BACHELOR'S THESIS / ENGINEERING 2024

Efficient Algorithms for Linearly Solvable Markov Decision Processes

David Pérez Carrasco

upf. Universitat Pompeu Fabra Barcelona

Efficient Algorithms for Linearly Solvable Markov Decision Processes

TREBALL DE FI DE GRAU DE

David Pérez Carrasco

Supervisor: Anders Jonsson

Degree in Mathematical Engineering in Data Science

Academic Year 2023-2024

Universitat Pompeu Fabra Barcelona

Escola d'Enginyeria

Acknowledgement

I would like to express my sincere gratitude to my supervisor Anders Jonsson, whose unwavering guidance, insightful feedback, and patient direction have been invaluable throughout this journey. His mentorship has shaped my work in profound ways. I am equally appreciative of my family and those who stood by my side, offering unwavering support and encouragement.

Abstract

In recent years, Reinforcement Learning (RL) has emerged as a powerful paradigm for sequential decision making under uncertainty. Within this framework, Markov Decision Processes (MDPs) serve as a fundamental model, defining the dynamics of state transitions and rewards. However, traditional RL algorithms, like Q-learning, often struggle with large or continuous state spaces due to computational complexity. Linearly-solvable Markov Decision Processes (LMDPs) offer a promising alternative, leveraging linear programming techniques for efficient planning and value function approximation.

The focus of this work is on evaluating and benchmarking state-of-the-art RL models against algorithms for continuous MDPs leveraging LMDPs, such as Z-learning. The aim is to improve the performance and scalability of these algorithms in larger and more intricate domains. We investigate efficient methods for optimal action selection and value function approximation within the linear framework. To enable a fair comparison with traditional MDP-based RL, we develop methods for embedding MDPs into LMDPs and vice versa. This allows us to benchmark traditional RL models against algorithms designed for continuous MDPs using LMDPs.

Furthermore, our research rigorously explores various factors that influence the learning behavior of algorithms in the context of Linearly-solvable MDPs. Particularly, we focus on analyzing the impact of different exploration strategies, aiming to uncover their effectiveness across diverse scenarios. By delving into these aspects, our study contributes valuable insights into the optimization and enhancement of reinforcement learning algorithms.

Resum

En els darrers anys, l'Aprenentatge per Reforç (RL) ha sorgit com un paradigma poderós per a la presa de decisions seqüencials sota incertesa. Dins d'aquest marc, els Processos de Decisió de Markov (MDP) serveixen com a model fonamental, definint la dinàmica de les transicions d'estat i les recompenses. No obstant això, els algorismes tradicionals de RL, com el "Q-learning", sovint tenen dificultats amb espais d'estats grans o continus a causa de la complexitat computacional. Els Processos de Decisió de Markov resolubles linealment (LMDP) ofereixen una alternativa prometedora, aprofitant tècniques de programació lineal per a la planificació eficient i l'aproximació de la funció de valor.

Aquest treball es centra en avaluar i comparar els models de RL més avançats amb algorismes per a MDPs continus que aprofiten els avantatges dels LMDPs, com ara el "Z-learning". L'objectiu és millorar el rendiment i l'escalabilitat d'aquests algorismes en dominis més grans i intricats. Investiguem mètodes eficients per a la selecció òptima d'accions i l'aproximació de la funció de valor dins del marc lineal. Per permetre una comparació justa amb el RL basat en MDPs tradicionals, desenvolupem mètodes per transformar MDPs en LMDPs i viceversa. Això ens permet comparar els models tradicionals de RL amb algorismes dissenyats per a MDPs continus utilitzant LMDPs.

A més, la nostra investigació explora rigorosament diversos factors que influeixen en el comportament d'aprenentatge dels algorismes en el context dels MDPs resolubles linealment. Particularment, ens centram en analitzar l'impacte de diferents estratègies d'exploració, amb l'objectiu de descobrir la seva efectivitat en diversos escenaris. En aprofundir en aquests aspectes, el nostre estudi aporta valuoses idees per a l'optimització i millora dels algorismes d'aprenentatge per reforç.

Resumen

En los últimos años, el Aprendizaje por Refuerzo (RL) ha surgido como un paradigma poderoso para la toma de decisiones secuenciales bajo incertidumbre. Dentro de este marco, los Procesos de Decisión de Markov (MDP) sirven como un modelo fundamental, definiendo la dinámica de las transiciones de estado y las recompensas. Sin embargo, los algoritmos tradicionales de RL, como el "Q-learning", a menudo enfrentan dificultades con espacios de estados grandes o continuos debido a la complejidad computacional. Los Procesos de Decisión de Markov resolubles linealmente (LMDP) ofrecen una alternativa prometedora, aprovechando técnicas de programación lineal para la planificación eficiente y la aproximación de la función de valor.

El enfoque de este trabajo se centra en evaluar y comparar modelos de RL de última generación con algoritmos para MDPs continuos que aprovechan las ventajas de los LMDPs, como el "Z-learning". El objetivo es mejorar el rendimiento y la escalabilidad de estos algoritmos en dominios más grandes e intrincados. Investigamos métodos eficientes para la selección óptima de acciones y la aproximación de la función de valor dentro del marco lineal. Para permitir una comparación justa con el RL basado en MDPs tradicionales, desarrollamos métodos para transformar MDPs en LMDPs y viceversa. Esto nos permite comparar los modelos tradicionales de RL con algoritmos diseñados para MDPs continuos utilizando LMDPs.

Además, nuestra investigación explora rigurosamente diversos factores que influyen en el comportamiento de aprendizaje de los algoritmos en el contexto de los MDPs resolubles linealmente. Particularmente, nos centramos en analizar el impacto de diferentes estrategias de exploración, con el objetivo de descubrir su efectividad en diversos escenarios. Al profundizar en estos aspectos, nuestro estudio aporta valiosas ideas para la optimización y mejora de los algoritmos de aprendizaje por refuerzo.

Contents

List of Figures

List of Tables

List of Algorithms

Introduction

Reinforcement Learning (RL) has rapidly advanced as a powerful methodology for sequential decision-making tasks under uncertainty. RL allows agents to learn optimal behaviors through interactions with their environment, receiving feedback in the form of rewards. This learning paradigm is particularly compelling because it mirrors the way humans and animals learn from their surroundings, making it suitable for a wide range of applications, from robotics and autonomous systems to game playing and financial modeling.

The greatest successes in reinforcement learning have been achieved in simulated environments, with video games often serving as ideal testing grounds due to their complexity and controlled settings. As Demis Hassabis, co-founder and CEO of Google's DeepMind, noted, "the reason games are used as a testing ground is that they're kind of like a microcosm of the real world" [Hassabis, 2016]. Building on this, specialized environments like Minigrid have become invaluable for RL research. Minigrid, a minimalistic gridworld, provides a versatile platform for evaluating RL algorithms, offering a range of tasks that facilitate rapid experimentation and robust benchmarking, essential for developing scalable and efficient RL models.

As we delve deeper into the mechanics of RL, it is essential to understand the foundational concepts underlying these algorithms. At the heart of RL lies the Markov Decision Process (MDP), a formal mathematical framework that models decision-making problems [Bellman, 1957, Puterman, 1994]. An MDP describes the environment in terms of states, actions, transitions, and rewards, providing a structured way to capture the dynamics of the system and the agent's objectives [Sutton and Barto, 2018]. Despite their foundational role, traditional RL algorithms such as Q-learning and SARSA often face substantial challenges when dealing with large or continuous state spaces. The primary issue is computational complexity: the need to explore and store values for a vast number of state-action pairs becomes impractical as the state space grows.

To tackle these scalability challenges, researchers have turned to Linearly-solvable Markov Decision Processes (LMDPs, [Kappen, 2005a, Todorov, 2006]).

LMDPs reformulate the RL problem to leverage linear programming techniques, transforming the typically nonlinear Bellman equation into a linear form. This linearization allows for more efficient computation of optimal policies and value functions, making it feasible to tackle larger and more intricate domains. By applying LMDPs, researchers aim to enhance the performance and scalability of RL algorithms, enabling them to operate effectively in environments like Minigrid and beyond [Todorov, 2009, Jonsson and Gómez, 2016].

Objectives and Scope

The primary objectives of this research are:

- To evaluate and benchmark the performance of traditional RL algorithms against those specifically designed for continuous MDPs using the LMDP framework. This involves a detailed analysis of their efficiency, scalability, and applicability to various domains, including simulated environments like Minigrid.
- To develop methods for embedding traditional MDPs into LMDPs and vice versa, allowing for a fair comparison of state-of-the-art RL models with algorithms designed for continuous MDPs using LMDPs.
- To investigate efficient techniques within the linear framework of LMDPs to enhance the decision-making capabilities of RL agents. This includes improving the algorithms' ability to approximate value functions accurately and select optimal actions effectively, while also analyzing efficient exploration strategies.

Structure of the Report

The initial chapters introduce the fundamentals of reinforcement learning and the MDP framework, followed by a dedicated chapter on LMDPs. Chapter 3 focuses on the development of classes and embeddings to enable a rigorous comparison between MDPs and LMDPs, with a special emphasis on enhancing embeddings. In Chapter 5, we present experimental results that highlight the comparative performance of Z-learning and Q-learning, as well as the impact of embedding accuracy and exploration strategies. The concluding chapter synthesizes the research findings and discusses potential avenues for future work, offering a trajectory for further exploration and development in the field.

Chapter 1

THE REINFORCEMENT LEARNING PARADIGM

Reinforcement Learning is a subfield of machine learning that focuses on how agents should take actions in an environment in order to maximize cumulative reward. Unlike traditional supervised learning, where models are trained on pre-labeled data, RL involves a process where an agent takes actions, observes the outcomes, and adjusts its behavior to maximize cumulative rewards. This dynamic learning method allows RL agents to tackle complex tasks and make decisions in uncertain and changing environments.

The methodology behind RL derives its success from its resemblance to the way both animals and humans learn through interaction with the environment. Unlike other machine learning methodologies, where the agent is given a clear objective, RL operates without prior knowledge of the goal or the steps needed to achieve it. Instead, the agent learns by receiving feedback from the environment in the form of rewards or penalties. This feedback informs the agent about the desirability of its actions, allowing it to adapt its behavior over time based on the consequences of its actions and the dynamics of the environment.

1.1 Markov Decision Processes (MDPs)

A Markov Decision Process (MDP) is a mathematical framework that models decision-making problems where outcomes are partly random and partly under the agent's control. MDPs belong to the class of mathematical problems used to describe the environment in the framework of reinforcement learning, specify the dynamics of the system, and set the goals for the agent. An MDP is formally defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$, is the state transition probability function, where $\Delta(\mathcal{S})$ is defined

as the set of probability distributions over $\mathcal{S}$, $\mathcal{R}$ is the reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and γ is the discount factor [Puterman, 1994, Sutton and Barto, 2018].

The set of states $\mathcal{S}$ describes all possible states within an environment besides an agent's experience. A set of actions, denoted $\mathcal{A}$, refers to all activities an agent can engage in. The state transition probability function, denoted $\mathcal{P}(s'|s, a)$, is the probability that the setting will transit to state s' provided the agent selects action a in state s. For the kind of problems we will use, MDPs contain a set of terminal states $\mathcal{T}$, from which there are no possible transitions. We denote $\mathcal{S}^-$ to the set of non-terminal states. However, the notation of the rest of the chapter will consider the general formulation of an MDP.

The reward function is a crucial component that defines the goal of the agent in the RL environment. It provides immediate feedback on the agent's actions, guiding the agent toward desirable outcomes. After each action a from a given state s, the environment delivers a reward, $\mathcal{R}(s, a)$, which is a scalar value indicating the immediate benefit or cost of the action taken. The goal of the agent is to maximize the long-term cumulative reward, often referred to as the return, which can be a sum or a discounted sum of rewards over time. This guides the agent towards actions that yield higher rewards in the long run.

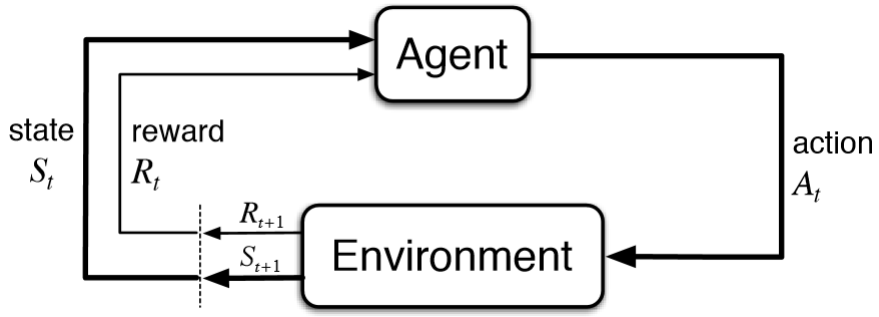

Figure 1.1: Basic agent behavior scheme in a Markov Decision Process.

The discount factor $\gamma \in [0, 1]$ is a parameter that measures the trade off between immediate and long-term rewards. The closer γ is to 0, the more short-sighted an agent is, while the closer γ is to 1, the more far-sighted an agent is, since the infinite horizon will more and more reflect the valuation of future rewards. Discounting is vital for the goal achievement as it balances immediate rewards and long-term cumulative rewards. Without discounting (i.e., setting the discount factor $\gamma = 1$), the agent becomes infinitely far-sighted. This means that the agent values future rewards just as much as immediate rewards.

As a result, the agent may focus on accumulating rewards over an extended period rather than prioritizing the fastest or most efficient path to the goal. Figure (1.1) illustrates the behavior of an RL agent in an MDP. Based on the information it has about the current state S_t, the agent selects an action. This action transitions the agent to a new state S_{t+1} and results in a reward R_{t+1}. Through this iterative process, the agent learns which actions lead to higher cumulative rewards, continually refining its understanding of the environment and improving its decision-making strategy.

MDPs are an abstraction made under the explicit assumption of the Markov property, which reflects that all future states depend only on the present one and not on the state-action pairs the agent exploits to get there, commonly referred to as the trajectory. Such memoryless property dramatically simplifies the problem of modeling decision-making. The agent faces the task of reinforcement learning of learning an optimal deterministic policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that maximizes the expected sum of accumulated rewards over time, in other words, a return. The return can be formalized as the sum of discounted rewards,

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{1.1}$$

where R_{t+k+1} is the reward received $k+1$ time steps after time t.

There are different algorithms devised to find the solutions to MDPs, varying from dynamic programming algorithms such as Value and Policy Iteration to model-free algorithms like Q-learning [Watkins and Dayan, 1992] and SARSA and model-based like Dyna-Q [Sutton, 1991]. The objective of all these algorithms is to estimate the value function, a measure of the expected return for every state, and find the optimal policy based on it. The Bellman equations are the fundamental recursive relations underlying an extensive range of RL algorithms. These recursive relations define a general constraint on a solution to the associated dynamic programming functional equations.

In other words, MDPs are a tool needed to design and solve a reinforcement learning decision-making problem. MDPs help structure a way to model the interactions between an agent and its environment to ease the process of developing algorithms in the learning space to make the best decisions under uncertainty. By leveraging the Markov property and defining the dynamics and rewards of the environment, MDPs help develop policies that guide the agent to attain his long-term objectives by trading off the immediate and cumulative rewards.

1.2 Core components of RL

Any reinforcement learning system has common subelements. In addition to the agent and the environment, the main components are a *policy*, a *reward signal*, a *value function*, and an optional *model* of the environment. The reward signal has already been defined as the reward function of an MDP.

1.2.1 Policy

A policy π is a strategy employed by the agent to determine its actions at any given state. It serves as the agent's decision-making mechanism, guiding its behavior to achieve the best possible outcome based on the information available. Policies can be categorized as:

- **Deterministic Policy**: A direct mapping where each state is mapped to a single action. Mathematically, it is defined as a function $\pi : \mathcal{S} \to \mathcal{A}$.
- **Stochastic Policy**: A probabilistic mapping where each state is mapped to a probability distribution over actions. It is defined as $\pi : \mathcal{S} \to \Delta(\mathcal{A})$, which represents the probability of taking action a in state s. This approach allows for exploration and can handle uncertainty better.

1.2.2 Value Function

The main problem for an RL agent, is that it cannot inherently know how close it is to achieving the goal, even if it knows the current state and the immediate reward associated with it. To provide some guidance to the agent, an approximation of how good the current state S_t is, based on the agent's prior experience, becomes crucial for learning and policy improvement.

Value functions are used to estimate how good it is for the agent to be in a given state (or to perform a certain action in a state), in terms of the return (equation 1.1). They provide a measure of the long-term benefit of states and actions:

- **State-Value Function** $v_\pi(s)$: This function gives the expected return starting from state s and following a specific policy π. It is mathematically expressed as:

$$v_\pi(s) \doteq \mathbf{E}_\pi \left[G_t | S_t = s \right] = \mathbf{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right], \text{ for all } s \in \mathcal{S}, \tag{1.2}$$

  where γ is the discount factor.

- **Action-Value Function** $q_\pi(s, a)$: This function provides the expected return of taking action a in state s and then following policy π. It is defined as:

$$\begin{aligned} q_\pi(s, a) &\doteq \mathbf{E}_\pi \left[G_t | S_t = s, A_t = a \right] \\ &= \mathbf{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right], \text{for all } s \in \mathcal{S}, a \in \mathcal{A}. \end{aligned} \tag{1.3}$$

Without value function approximations, agents would base their action decisions solely on immediate rewards, which, in most cases, could cause the agent to get stuck and never reach the goal. This discrepancy arises because a state can have a high reward but a low value, or vice versa. Immediate outcomes may not always align with long-term desirability. For instance, a state S_t might yield a smaller reward than another state but serve as a faster path to the goal, making it more desirable overall.

Thus, action choices are made based on value judgments. We seek actions that bring about states of highest value, not just the highest reward, because these actions maximize cumulative rewards over the long run.

1.2.3 Model

A model in RL is an optional component that represents the agent's understanding of the environment. It can predict the next state and reward given a state and action. Models are particularly useful in planning and depending on whether RL methods are model-based or not, they can be categorized as:

- **Model-Based Methods**: These methods utilize a model to simulate the environment's dynamics for planning and decision-making. They can predict future states and rewards, allowing the agent to evaluate the potential outcomes of actions without taking them in the real environment. Examples include algorithms like Dyna-Q [Sutton, 1991] and planning methods such as Value Iteration (discussed in Section 1.4).

- **Model-Free Methods**: These methods do not use a model of the environment and learn directly from interactions with it. They estimate value functions or policies based on the observed rewards and transitions. Some of the most popular examples include Q-learning [Watkins and Dayan, 1992] and SARSA [Rummery et al., 1994].

1.3 Bellman Equations

As mentioned earlier, the learning process of a reinforcement learning agent depends on its interactions with the environment and the value function estimations. A state's value should be calculated correctly by considering all possible transitions from that state and the values of reaching states. It is described in the Bellman Equation for the State-value Function as follows:

$$\begin{aligned} v_\pi(s) &\doteq \mathbf{E}_\pi[G_t \mid S_t = s] \\ &= \mathbf{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \quad \text{(by (1.2))} \\ &= \sum_a \pi(a \mid s) \left[\mathcal{R}(s,a) + \gamma \sum_{s'} \mathcal{P}(s' \mid s,a) v_\pi(s') \right], \quad \text{for all } s \in \mathcal{S}. \end{aligned} \tag{1.4}$$

This equation holds for any policy π and any state s. Similarly, the action-value function q_π is computed using the same recursive process applied to each state-action pair (s, a).

The optimal policy is characterized by the optimal state-value function v^* and the optimal action-value function q^*. Although there may be several different optimal policies, they all share these optimal value functions. The Bellman optimality equation expresses that the value of a state under an optimal policy must be equal to the expected return of the best action from that state, and similarly for the best actions from subsequent states [Bertsekas and Tsitsiklis, 1996, Sutton and Barto, 2018]:

$$v^*(s) = \max_a \left(\mathcal{R}(s,a) + \gamma \sum_{s'} \mathcal{P}(s' \mid s,a) v^*(s') \right), \quad \text{for all } s \in \mathcal{S}. \tag{1.5}$$

Furthermore, the Bellman optimality equation for the action-value function is defined as follows:

$$q^*(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s'} \mathcal{P}(s' \mid s,a) \max_{a'} q^*(s',a'), \quad \text{for all } s \in \mathcal{S}, a \in \mathcal{A}. \tag{1.6}$$

These Bellman equations are central to many reinforcement learning algorithms. By repeatedly applying these equations, agents can evaluate and improve their policies, ultimately aiming to find the optimal policy for maximizing cumulative rewards. Notably, the state-value function $v_\pi(s)$ relates to the action-value function $q_\pi(s,a)$ as $v_\pi(s) = \max_a q_\pi(s,a)$, and similarly, $v^*(s) = \max_a q^*(s,a)$.

Reinforcement learning agents aim to find optimal policies that maximize cumulative rewards. However, finding such optimal policies using the Bellman equations directly is computationally expensive and impractical for most real-world problems. Therefore, reinforcement learning research focuses on finding accurate approximations of these optimal policies. Among the various methods available, dynamic programming (DP) methods play a crucial role. While we provide a detailed discussion of DP methods, including policy evaluation, policy improvement, and policy iteration in Appendix (A.1), we focus here on the value iteration method, which is central to this thesis.

1.4 Value Iteration

Value iteration is a dynamic programming method that iteratively applies the Bellman optimality equation to update the value function until it converges to the optimal value function v_*. The optimal policy can then be derived from this optimal value function. The value iteration algorithm is described as follows:

Algorithm 1 Value Iteration

input: discount rate $\gamma \in [0, 1]$, a small positive number θ, MDP with $\mathcal{R}$, $\mathcal{P}$, $\mathcal{S}$, $\mathcal{A}$, $\mathcal{S}^-$, $\mathcal{T}$
output: $V : \mathcal{S} \to \mathbb{R}$
initialize $V(s) \leftarrow 0$ for all $s \in \mathcal{S}^-$, $V(s) \leftarrow \mathcal{R}(s, a)$ for all $s \in \mathcal{T}$, for any $a \in \mathcal{A}$
repeat
    $\Delta \leftarrow 0$
    for each state $s \in \mathcal{S}^-$ **do**
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_{a \in \mathcal{A}}(\mathcal{R}(s, a) + \gamma \sum_{s'} \mathcal{P}(s'|s, a)V(s'))$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end for
until $\Delta < \theta$

The final value function v^* represents the optimal value of each state. It has been shown that value iteration converges to an optimal policy in a finite number of iterations for discounted finite MDPs [Bertsekas, 1987, Puterman, 1994, Sutton and Barto, 2018]. The optimal policy π^* can be derived by selecting the action that maximizes the expected return from each non-terminal state:

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} \left(\mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, a)v^*(s') \right) \tag{1.7}$$

While value iteration effectively approximates the optimal value functions and policy, it requires a complete model of the environment, including transition probabilities and reward functions, which is often challenging to obtain in practice. Additionally, it is computationally expensive for large state and action spaces. Hence, exploring alternative methods that do not require an explicit model of the environment and are more computationally efficient is crucial.

1.5 Q-learning

Q-learning is a widely used model-free reinforcement learning algorithm that belongs to the class of Temporal Difference (TD) methods. TD methods update their estimates based on a single step, making them efficient and suitable for online learning. Q-learning, in particular, is off-policy, meaning it learns the value of the optimal policy independently of the agent's actions [Watkins, 1989]. This allows Q-learning to evaluate the optimal policy while potentially following a different, exploratory policy.

Q-learning aims to learn the optimal action-value function $Q^*(s, a)$ that gives the maximum expected future rewards for taking an action a in state s. The Q-learning update rule is given by:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] \tag{1.8}$$

Here, α is the learning rate, γ is the discount factor, r_{t+1} is the reward received after taking action a_t in state s_t, and s_{t+1} is the next state.

To balance exploration and exploitation, Q-learning commonly uses the ϵ-greedy approach. With probability ϵ, the agent chooses a random action (exploration), and with probability $1 - \epsilon$, it chooses the action that has the highest estimated reward (exploitation). This strategy ensures that the agent explores enough of the state space to find the optimal policy but still exploits its current knowledge to maximize rewards.

Algorithm (2) illustrates the iterative process of Q-learning, which learns the optimal policy while using another policy to interact with the environment. Q-learning has been shown to converge to the optimal value function $q^*(s, a)$ with probability 1, given that all state-action pairs continue to be updated over time [Watkins and Dayan, 1992, Tsitsiklis, 1994, Jaakkola et al., 1994].

Algorithm 2 Q-learning

input: discount rate $\gamma \in [0, 1]$, exploration factor $\epsilon \in (0, 1]$, MDP with $\mathcal{R}$, $\mathcal{P}$, $\mathcal{S}$, $\mathcal{A}$, $\mathcal{S}^-$, $\mathcal{T}$
output: $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$
initialize $Q(s, a) \leftarrow 0$ for all $s \in \mathcal{S}^-$, $a \in \mathcal{A}$, $Q(s, a) \leftarrow \mathcal{R}(s, a)$ for all $s \in \mathcal{T}$, $a \in \mathcal{A}$
repeat
    $s_t \leftarrow s_0$ (sample state from initial state distribution)
    while $s \notin \mathcal{T}$ **do**
        Select action a_t using policy derived from Q (e.g., ϵ-greedy)
        Take action a_t, observe r_{t+1}, s_{t+1}
        $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \right.$
                $\left. - Q(s_t, a_t) \right]$
        $s_t \leftarrow s_{t+1}$
    end while
until convergence

1.6 Challenges with Traditional MDP-Based Reinforcement Learning

Reinforcement learning with traditional MDPs presents challenges such as the exploration-exploitation dilemma and scalability issues. These highlight the need for more efficient approaches, like Linearly-solvable Markov Decision Processes, discussed in the following chapter.

1.6.1 Exploration vs. Exploitation Dilemma

A key challenge in RL is balancing exploration and exploitation. Exploration involves trying new actions to gather information about the environment, which is essential for finding the optimal policy. However, it can lead to short-term suboptimal rewards. Exploitation, on the other hand, uses the agent's current knowledge to maximize immediate rewards based on prior experience. Several strategies have been developed to address this dilemma:

- **ϵ-Greedy Strategy**: In this approach, the agent chooses a random action with probability ϵ and the action with the highest estimated reward with probability $1 - \epsilon$.

$$\pi(a|s) = \begin{cases} \text{random action} & \text{with probability } \epsilon \\ \arg\max_a Q(s, a) & \text{with probability } 1 - \epsilon \end{cases} \tag{1.9}$$

- **Upper Confidence Bound (UCB)**: This strategy selects actions based on an upper confidence bound that considers both the estimated reward and the uncertainty of that estimate. It encourages the agent to explore less-visited actions by accounting for the potential of high rewards.

1.6.2 Scalability and Computational Complexity

Scalability is a major issue as state and action spaces increase, leading to higher computational complexity and memory requirements.

- **State Space Explosion**: As the number of states and actions increases, the number of state-action pairs that need to be considered grows exponentially. This requires vast amounts of memory to store the value functions and extensive computational resources to update them.
- **Computational Demands**: In large state spaces, updating the value functions or Q-values for every state-action pair becomes highly computationally intensive. Each interaction with the environment necessitates updates, which significantly slows down the learning process and increases the computational burden.

Furthermore, solving the Bellman equation (1.5) in large instances is computationally demanding. Traditional MDP-based methods like value iteration and policy iteration become slow as state spaces grow, requiring extensive updates for each state and leading to slow convergence rates.rates.

These challenges highlight the limitations of traditional MDP-based RL methods in handling large, complex environments. The next chapter will explore LMDPs, which offer promising solutions to these challenges by leveraging linear programming techniques and addressing the inefficiencies of traditional MDP approaches.

Chapter 2

LINEARLY-SOLVABLE MARKOV DECISION PROCESSES

Linearly-solvable Markov Decision Processes (LMDPs) represent a specialized subclass of MDPs that achieves more efficient computation of optimal policies and value functions [Kappen, 2005b, Todorov, 2006]. By modifying the dynamics and leveraging linear programming techniques, LMDPs streamline the traditionally complex processes involved in solving reinforcement learning problems.

Unlike traditional Markov Decision Processes, which rely on solving nonlinear Bellman equations, LMDPs reformulate these equations into a linear form. This transformation significantly enhances computational efficiency and scalability, making LMDPs particularly advantageous in handling large and complex environments. Within this family of MDPs, Todorov reformulates the traditional maximization problem of finding an optimal policy (as in equation 1.5) into a minimization problem over the control space [Todorov, 2006], which is convex and analytically tractable due to the properties of LMDPs.

Recent applications of LMDPs span various domains, showcasing their versatility and robustness. Leveraging their role in reinforcement learning as probabilistic inference [Kappen et al., 2012, Levine, 2018], LMDPs have been utilized in multi-agent systems, demonstrating benefits in coordination and emergent behaviors [Gómez et al., 2020]. In robotics, their application has optimized navigation tasks in complex environments [Gómez et al., 2014, Williams et al., 2017]. Additionally, employing KL divergence minimization has enhanced performance in standard benchmarks such as Gridworld and Cart-Pole [Rawlik et al., 2012].

In this chapter, we provide an overview of the theoretical foundations and formulations of LMDPs, following closely the derivations from [Todorov, 2006] and [Jonsson and Gómez, 2016]. To ensure consistency and clarity, we will adopt a unified notation integrating elements from these references.

2.1 LMDP Definition

An LMDP shares several components with a traditional MDP, but the definitions of these components differ, resulting in unique dynamics. These dynamics can simulate the same environment as an MDP (see Section 1.1) but with greater efficiency. Formally, an LMDP is defined by a tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R} \rangle$ where

- $\mathcal{S}$ is a set of states.
- $\mathcal{P} : \mathcal{S} \to \Delta(\mathcal{S})$ is an uncontrolled transition probability distribution satisfying $\sum_{s'} \mathcal{P}(s'|s) = 1$ for each state $s \in \mathcal{S}$. In contrast to MDPs, where the state transition probability function is $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$, LMDPs do not explicitly use actions, instead, state transitions are directly governed by the transition probability distribution.
- $\mathcal{R} : \mathcal{S} \to \mathbb{R}$ is an expected reward function. Unlike in MDPs, where the reward function depends on actions, the reward is an inherent property of the states themselves, independent of transitions.

As noted before, in the problems we consider, there exists a non-empty subset of terminal states $\mathcal{T} \subset \mathcal{S}$. Terminal states mean the conclusion of an RL episode, meaning that there are no possible transitions from a terminal state. This implies that $\mathcal{P} : \mathcal{S}^- \to \Delta(\mathcal{S})$, where $\mathcal{S}^- \subset \mathcal{S}$ is the set of non-terminal states. For both MDPs or LMDPs, $v_\pi(s) = \mathcal{R}(s)$ for all $s \in \mathcal{T}$ and any policy π. Since there are no transitions from terminal states, the symbolic actions in an MDP have no effect. Consequently, the reward at a terminal state remains the same for all the actions, leading to $v_\pi(s) = \mathcal{R}(s, a)$ for any a in an MDP.

The absence of actions means there is no explicit policy as in traditional MDPs. Hence, the interaction between the agent and the environment is solely defined by the transition probability distribution. Instead of a policy, LMDPs utilize a control variable $\mathbf{u} \in \mathbb{R}^{|\mathcal{S}|}$, which is a real-valued vector with dimensionality equal to the number of states.

The control $\mathbf{u}$ distinguishes between uncontrolled transition probabilities, determined solely by the environment's dynamics, and the controlled transition probabilities, which result from modifying the uncontrolled transition probabilities under the influence of $\mathbf{u}$. This is analogous to how a policy π governs transitions in MDPs. Specifically, controlled transition probabilities are defined as:

$$\mathcal{P}_{\mathbf{u}}(s'|s) = \mathcal{P}(s'|s)e^{u_{s'}} \ . \tag{2.1}$$

There are two implicit constraints in equation (2.1). First, for a given state $s' \in \mathcal{S}$, $\mathcal{P}_{\mathbf{u}}(s'|s)$ can be non-zero only if $\mathcal{P}(s'|s)$ is non-zero. Second, $\sum_{s' \in \mathcal{S}} \mathcal{P}_{\mathbf{u}}(s'|s) = 1$ for all $s \in \mathcal{S}^-$.

Since the control vector has a direct effect on the transition probabilities, the reasonable way to measure its magnitude is by measuring the difference between the controlled and uncontrolled transition probabilities, which are measured using Kullback-Leibler (KL) divergence. In particular, the reward $R(s, \mathbf{u})$ for applying control $\mathbf{u}$ in state s is:

$$\begin{aligned} R(s, \mathbf{u}) &= \mathcal{R}(s) - \lambda \cdot KL\left(\mathcal{P}_{\mathbf{u}}(\cdot|s) \middle\| \mathcal{P}(\cdot|s)\right) \\ &= \mathcal{R}(s) - \lambda \cdot \sum_{s' \in S: \mathcal{P}(s'|s) \neq 0} \mathcal{P}_{\mathbf{u}}(s'|s) \log \frac{\mathcal{P}_{\mathbf{u}}(s'|s)}{\mathcal{P}(s'|s)} \,. \end{aligned} \tag{2.2}$$

In this case, $\mathcal{R}(s)$ is the (non-positive) reward associated with state s, where $\mathcal{R}(s)$ is strictly negative for all $s \in \mathcal{S}^-$ [Infante et al., 2022], but in alternative formulations, a (non-negative) cost could be used instead [Todorov, 2006]. The KL divergence penalizes controls that significantly differ from $\mathcal{P}$, which usually behaves like a random walk, as it is determined by the environment dynamics. The temperature parameter λ has a direct effect on the control as well. High temperature result in more stochastic solutions since deviating from the uncontrolled dynamics is penalized, while small temperature lead to deterministic policies.

Substituting 2.2 into 1.5 and dividing the value function by the temperature parameter, we obtain the Bellman equation for the LMDP:

$$\begin{aligned} \frac{1}{\lambda} v(s) &= \frac{1}{\lambda} \max_{\mathbf{u} \in \mathcal{U}(s)} \left(R(s, \mathbf{u}) + \sum_{s' \in S} \mathcal{P}_{\mathbf{u}}(s'|s) v(s') \right) \\ &= \frac{1}{\lambda} \mathcal{R}(s) + \max_{\mathbf{u} \in \mathcal{U}(s)} \sum_{s' \in \mathcal{S}} \mathcal{P}_{\mathbf{u}}(s'|s) \left[\frac{1}{\lambda} v(s') - \log \frac{\mathcal{P}_{\mathbf{u}}(s'|s)}{\mathcal{P}(s'|s)} \right] \,. \end{aligned} \tag{2.3}$$

Where $\mathcal{U}(s)$ consists of all control inputs for any state $s \in \mathcal{S}$ that satisfy the constraints in equation (2.1). Due to the lack of a discount factor, the terminal states are absorbing, which means the episode finishes once reaching them. Formally, it strictly implies $\mathcal{P}(t|t) = 1$ for each $t \in T$.

We reformulate (2.3) using the exponential transformation $z(s) = e^{v(s)/\lambda}$:

$$\begin{aligned} \frac{1}{\lambda} v(s) &= \frac{1}{\lambda} \mathcal{R}(s) + \max_{\mathbf{u} \in \mathcal{U}(s)} \sum_{s' \in \mathcal{S}} \mathcal{P}_{\mathbf{u}}(s'|s) \left[-\log \frac{\mathcal{P}_{\mathbf{u}}(s'|s)}{\mathcal{P}(s'|s) z(s')} \right] \\ &= \frac{1}{\lambda} \mathcal{R}(s) - \min_{\mathbf{u} \in \mathcal{U}(s)} \sum_{s' \in \mathcal{S}} \mathcal{P}_{\mathbf{u}}(s'|s) \left[\log \frac{\mathcal{P}_{\mathbf{u}}(s'|s)}{\mathcal{P}(s'|s) z(s')} \right] \,. \end{aligned} \tag{2.4}$$

Now, we insert a normalization term $\sum s' \mathcal{P}(s'|s) z(s')$ into the Bellman equation to obtain a KL divergence on the right side:

$$\begin{aligned}\frac{1}{\lambda} v(s) &= \frac{1}{\lambda}\mathcal{R}(s) - \min_{\mathbf{u}\in\mathcal{U}(s)} \sum_{s'\in\mathcal{S}} \mathcal{P}_{\mathbf{u}}(s'|s) \left[\log \frac{\mathcal{P}_{\mathbf{u}(s'|s)} \sum_{s''\in\mathcal{S}} \mathcal{P}(s''|s) z(s'')}{\mathcal{P}(s'|s) z(s') \sum_{s''\in\mathcal{S}} \mathcal{P}(s''|s) z(s'')}\right] \\ &= \frac{1}{\lambda}\mathcal{R}(s) + \log \sum_{s'\in\mathcal{S}} \mathcal{P}(s'|s) z(s') \\ &\quad - \min_{\mathbf{u}\in\mathcal{U}(s)} KL\left(\mathcal{P}_{\mathbf{u}}(\cdot|s) \middle\| \frac{\mathcal{P}(\cdot|s) z(\cdot)}{\sum_{s'\in\mathcal{S}} \mathcal{P}(s'|s) z(s')}\right) .\end{aligned} \tag{2.5}$$

The KL term is cancelled when the controlled and the uncontrolled transition probability distributions are equal, resulting in the following formulation of the optimal controlled transition probability:

$$\mathcal{P}_{\mathbf{u}^*}(s'|s) = \frac{\mathcal{P}(s'|s) z(s')}{\sum_{s''\in\mathcal{S}} \mathcal{P}(s''|s) z(s'')} . \tag{2.6}$$

Therefore, the optimality Bellman equation, after applying the exponential transformation, is defined as follows:

$$z(s) = e^{\mathcal{R}(s)/\lambda} \sum_{s'\in\mathcal{S}} \mathcal{P}(s'|s) z(s') . \tag{2.7}$$

Defining the vector $\mathbf{z}$ with elements $z(s)$, and the diagonal matrix G with the terms $e^{\mathcal{R}(s)/\lambda}$ on its main diagonal, as well as considering the already defined uncontrolled transition probability matrix $\mathcal{P}$, we can write equation (2.7) in matrix form as:

$$\mathbf{z} = G\mathcal{P}\mathbf{z} . \tag{2.8}$$

Thus within LMDPs we have transformed the Bellman optimality equation into a system of linear equations [Todorov, 2006, Jonsson and Gómez, 2016].

2.2 Power Iteration

Since the equation (2.8) is linear, we can solve its eigenvector problem through iterative methods. We can infer that $\mathbf{z}$ is an eigenvector of $G\mathcal{P}$ with eigenvalue 1. Furthermore, as we are assuming $\mathcal{R}(s)$ is bounded in $(-\infty, 0)$ for $s \in \mathcal{S}^-$ and in $(-\infty, 0]$ for $s \in \mathcal{T}$, $z(s)$ will be positive for all $s \in \mathcal{S}$, specifically bounded in $[0, 1)$ for non-terminal states and $[0, 1]$ for terminal states, where $z(s) = e^{\mathcal{R}(s)/\lambda}$ for $s \in \mathcal{T}$ [Molina et al., 2023].

Power Iteration is a method used to find the dominant eigenvector of a matrix, which in this context is the vector $\mathbf{z}$ that solves the system of linear equations (2.8) for $\mathcal{S}^{-}$. The algorithm iteratively multiplies an initial vector $\mathbf{z}^{(0)}$ by the matrix $G\mathcal{P}$ and normalizes the result until convergence. The steps are as follows:

1. Initialize $\mathbf{z}^{(0)}$ to an arbitrary positive vector. In our case, $\mathbf{z}^{(0)} = 1$.

2. For $k = 0, 1, 2, \ldots$ until convergence:

$$\mathbf{z}^{(k+1)} = G\mathcal{P}\mathbf{z}^{(k)} \; . \tag{2.9}$$

3. Upon convergence, $\mathbf{z}^{(k)}$ approximates the solution to $\mathbf{z} = G\mathcal{P}\mathbf{z}$, as demonstrated in [Todorov, 2006].

Power Iteration is simple and efficient, particularly for large sparse matrices, making it suitable for solving the linear equations in LMDPs. In infinite-horizon discounted-reward problems, equation (2.8) can be formulated as:

$$\mathbf{z} = G\mathcal{P}\mathbf{z}^{\gamma} \; . \tag{2.10}$$

Where $\gamma < 1$ is the discount factor. This equation is nonlinear, but the analogous iterative method (2.9 has been shown to converge swiftly [Todorov, 2006].

2.3 Z-learning

Power iteration shares the same limitation with value iteration: they require a model of the environment. To handle the cases when a model is not available we require stochastic approximations of the value function. Todorov proposed an on-line learning algorithm for LMDPs called Z-learning, corresponding to the analogous Q-learning for MDPs [Todorov, 2006, Todorov, 2009].

Instead of computing the entire matrix $G\mathcal{P}$, Z-learning updates the $\mathbf{z}$ values iteratively based on observed transitions and rewards. The estimations are updated at each time step as:

$$\hat{z}(s_t) \leftarrow (1-\alpha)\hat{z}(s_t) + \alpha e^{r_t/\lambda}\hat{z}(s_{t+1}), \tag{2.11}$$

where s_t is the state at time step t, with its corresponding reward $r_t = \mathcal{R}(s_t)$, and s_{t+1} is the next state reached after the transition. This update rule assumes the uncontrolled transition probability distribution is used for sampling the states, which is known as Naive Z-learning [Jonsson and Gómez, 2016] and is generally no better than a random walk. A better approach is to use an optimal controlled transition probability distribution $\hat{\mathcal{P}}_{\mathbf{u}}(s'|s)$ derived from $\hat{z}(s)$ to use the learning from the agent for sampling:

$$\hat{z}(s_t) = (1-\alpha)\hat{z}(s_t) + \alpha e^{r_t/\lambda} \frac{\mathcal{P}\left(s_{t+1}|s_t\right)}{\hat{\mathcal{P}}_{\mathbf{u}}\left(s_{t+1}|s_t\right)} \hat{z}(s_{t+1}), \tag{2.12}$$

where $\mathcal{P}\left(s_{t+1}|s_t\right)/\mathcal{P}_{\mathbf{u}}\left(s_{t+1}|s_t\right)$ serves as an importance weight and requires access to the uncontrolled dynamics [Infante et al., 2022]. It can be further reformulated in order to simply use the immediate value function:

$$\begin{aligned}
\hat{z}(s_t) &\leftarrow (1-\alpha)\,\hat{z}(s_t) + \alpha e^{r_t/\lambda} \frac{\mathcal{P}\left(s_{t+1}|s_t\right)}{\hat{\mathcal{P}}_{\mathbf{u}}\left(s_{t+1}|s_t\right)} \hat{z}\left(s_{t+1}\right) \\
&= (1-\alpha)\,\hat{z}(s_t) + \alpha e^{r_t/\lambda} \frac{\mathcal{P}\left(s_{t+1}|s_t\right)\hat{z}(s_{t+1})}{\mathcal{P}\left(s_{t+1}|s_t\right)\hat{z}(s_{t+1})} \sum_{s' \in S} \mathcal{P}\left(s'|s_t\right)\hat{z}(s') \\
&= (1-\alpha)\,\hat{z}(s_t) + \alpha e^{r_t/\lambda} \sum_{s' \in S} \mathcal{P}\left(s'|s_t\right)\hat{z}(s')
\end{aligned} \tag{2.13}$$

Hence, the update rule moves $\hat{z}(s_t)$ in the direction of the expected value of $\hat{z}(s_t)$, which is the desired estimate [Jonsson and Gómez, 2016], as described in Algorithm (3).

Algorithm 3 Z-learning

input: learning rate $\alpha \in (0, 1]$, temperature parameter $\lambda > 0$, LMDP with $\mathcal{R}$, $\mathcal{P}$, $\mathcal{S}$, $\mathcal{S}^-$, $\mathcal{T}$
output: $\hat{Z} : S \to \mathbb{R}$
initialize $\hat{Z}(s) \leftarrow 1$, for all $s \in \mathcal{S}^-$, $\hat{Z}(s) \leftarrow e^{\mathcal{R}(s)/\lambda}$, for all $s \in \mathcal{T}$, $\hat{\mathcal{P}}_{\mathbf{u}} \leftarrow \mathcal{P}$
repeat
    $s_t \leftarrow s_0$ (sample state from initial state distribution)
    while $s_t \notin \mathcal{T}$ **do**
        Take reward r_t from the current state s_t.
        $G\left[z\right](s_t) \leftarrow \sum_{s' \in \mathcal{S}} \mathcal{P}\left(s' \mid s\right)\hat{Z}(s')$
        $\hat{Z}(s_t) \leftarrow \hat{Z}(s_t) + \alpha\left[e^{r_t/\lambda} G[z](s_t) - \hat{Z}(s_t)\right]$
        Update $\hat{\mathcal{P}}_{\mathbf{u}}$ derived from $\hat{Z}$
        Sample a next state s_{t+1} according to $\hat{\mathcal{P}}_{\mathbf{u}}$
        $s_t \leftarrow s_{t+1}$
    end while
until convergence

Chapter 3

METHODOLOGY

In this chapter, we detail the methodology employed to assess the efficiency and scalability of LMDPs in contrast to MDPs. The experimental evaluation was carried across diverse environments, including Minigrid and Gridworld. We also formulate embedding techniques between LMDPS and MDPs to generate precise approximations in both directions enabling a fair comparison.

3.1 Sequential Decision Domains

The primary objective of this work was to create a robust simulator capable of representing sequential decision problems through both deterministic and stochastic actions. This involved developing and integrating various sequential decision domains, such as Gridworld, with existing reinforcement learning frameworks like Gymnasium and Minigrid. By doing so, we aimed to facilitate the implementation and evaluation of Reinforcement Learning algorithms within these domains, thereby providing a platform for comprehensive benchmarking and analysis.

To achieve this, we developed both MDP and LMDP frameworks as abstract classes, ensuring they could be adapted to the dynamics of any sequential decision problem. This required creating generic, flexible implementations of MDPs and LMDPs that could accommodate the unique characteristics and dynamics of diverse RL problems. Our approach focused on ensuring versatility and scalability, making it possible to represent a wide array of RL problems, from simple grid-based environments to more complex, high-dimensional state spaces.

3.1.1 MDP Representation

To develop an abstract class for MDPs, we followed the standard MDP framework, defining it as a tuple $(\mathcal{S}, \mathcal{S}^-, \mathcal{T}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{T}$ is the subset of terminal states and $\mathcal{S} = \mathcal{S}^- \cup \mathcal{T}$. This class is designed to be flexible, accepting the number of states $|\mathcal{S}|$, non-terminal states $|\mathcal{S}^-|$, and actions $|\mathcal{A}|$ as input parameters.

The abstract MDP class includes transition probabilities $\mathcal{P}$, represented as a matrix with dimensions $|\mathcal{S}^-| \times |\mathcal{A}| \times |\mathcal{S}|$, and reward functions $\mathcal{R}$, with dimensions $|\mathcal{S}| \times |\mathcal{A}|$. Both $\mathcal{P}$ and $\mathcal{R}$ are initialized as zero-valued matrices, allowing specialized domains to define their dynamics and rewards. States are represented as integers, simplifying the handling of dynamics and rewards. A mapping function ensures correct indexing for multidimensional states. The discount rate γ is included as an attribute of the MDP and serves as the default for various methods and algorithms, with flexibility for different discount rates as needed. Specialized classes derived from the abstract class manage the state set $\mathcal{S}$, which can be domain-specific. The abstract class focuses on providing a foundational structure, leaving domain-specific characteristics to be defined in subclasses.

Managing terminal states $\mathcal{T}$ initially involved separate sets for $\mathcal{S}$, $\mathcal{S}^-$, and $\mathcal{T}$, leading to inefficiencies. An index-based strategy was adopted, where terminal states are indexed sequentially after non-terminal states in $\mathcal{P}$ and $\mathcal{R}$. This simplifies terminal state detection—any state index higher than $|\mathcal{S}^-|$ is terminal. For challenging domains where state order is significant, a mapping function is used to manage state ordering and maintain accuracy. This strategy avoids inefficiencies and simplifies computational tasks for methods requiring constant access to $\mathcal{S}^-$. The generic class includes an initial state s_0, set to index 0 by default, useful for online algorithms starting from a defined initial state.

The generic MDP class includes essential methods for computing value functions, transitioning to next states, and embedding methods for constructing equivalent LMDPs. These methods provide a universal framework, enabling researchers to benchmark and compare different MDP settings without adapting these methods for each unique problem.

A key method in the generic class is the state transition function, named *act*, determining the next state s' from a given state s and action a using the transition probabilities $\mathcal{P}$. It supports both deterministic and stochastic actions by employing a random choice generator over the distribution $\mathcal{P}(s'|s, a)$. For deterministic actions, where $\mathcal{P}(s'|s, a) = 1$, the next state is straightforwardly selected. For stochastic actions, the next state is sampled according to the transition distribution. The reward is directly obtained from $\mathcal{R}(s, a)$, and the terminal status of s' is determined according to $s' \geq |\mathcal{S}^-|$.

Value iteration is another critical method included in the generic MDP class, enabling the computation of the optimal value function for any reinforcement learning problem. Implemented in a vectorized manner, as described in Algorithm (8), this method significantly enhances computational efficiency compared to traditional loop-based implementations. The method takes the stopping criteria parameter ϵ and the discount rate γ as arguments, defaulting to the inherent MDP discount rate. This vectorized approach demonstrates superior performance (detailed in Appendix B.1), making it an invaluable tool for solving large-scale MDPs efficiently.

Finally, the embedding function constructs an equivalent LMDP, based on the procedure outlined in [Todorov, 2006]. During this research, several cases of MDPs where Todorov's method could not achieve an optimal approximation were encountered. To address these limitations, alternative proposals are incorporated into this method, as described in the following sections.

The first step of the embedding method is to determine whether the actions in the MDP are stochastic or deterministic. Once the nature of the dynamics is identified, the method proceeds with the appropriate embedding technique detailed later in this chapter. Implementing the embedding method at the generic MDP class level, rather than at domain-specific levels, ensures a fair comparison between algorithms by providing a consistent and exact approximation across different frameworks. This universal applicability makes the method robust and versatile, suitable for any type of dynamics, problem size, or state space.

3.1.2 LMDP Representation

The LMDP class mirrors the MDP structure and encapsulates the key components as a tuple: $(\mathcal{S}, \mathcal{S}^-, \mathcal{T}, \mathcal{P}, \mathcal{R}, \lambda)$ as defined in the previous chapter. It initializes with parameters $|\mathcal{S}|$, $|\mathcal{S}^-|$, s_0, and λ, serving as the default temperature parameter. Transition probabilities $\mathcal{P}$ and reward functions $\mathcal{R}$ are zero-valued matrices, allowing specialized subclasses to define specific dynamics and rewards.

In LMDPs, actions are not explicitly defined. The state transition method *act* uses a transition probability distribution P, which defaults to the LMDP's inherent $\mathcal{P}$ but can utilize any provided probability distribution. This enables transitions based on the optimal or any approximated policy. This method samples the next state s' using a random choice generator, retrieves the reward $\mathcal{R}(s)$, and checks if s' is terminal, similarly to the MDP method..

The power iteration method, analogous to value iteration in MDPs, is implemented in a vectorized manner, as illustrated in Algorithm (9).

Initially, convergence was determined by the difference between successive $\mathbf{z}^{(k)}$ values, but for a more equitable comparison between MDPs and LMDPs, transforming $\mathbf{z}$ into v using $v = \lambda \log \mathbf{z}$ provided a more comparable metric. Additionally, the implementation leverages sparse matrix operations to optimize performance. Sparse matrices, such as transition probability matrices $\mathcal{P}$ and diagonal reward matrices G, contain a significant proportion of zero values, allowing for efficient computational optimization. This optimization results in an algorithm that is over 100 times faster, as shown in Table (B.2), significantly enhancing the efficiency of handling LMDPs, especially in large-scale settings where computing the optimal value function would otherwise be computationally prohibitive.

Another essential method is computing the optimal controlled transition probabilities $\mathcal{P}_{\mathbf{u}^*}$. This method accepts $\mathbf{z}$ as an argument, allowing flexibility to compute controlled probabilities for any $\mathbf{z}$ approximation. It then calculates $\mathcal{P}_{\mathbf{u}^*}$ using the uncontrolled transition probabilities $\mathcal{P}$ as in equation (2.6). Additionally, the class includes a method for transforming $\mathbf{z}$ into the value function v for subsequent comparisons with MDP methodologies.

Finally, the LMDP class incorporates an embedding method to construct an equivalent MDP, following the procedure outlined in [Todorov, 2006]. Implemented in a vectorized approach, this ensures consistent and fair comparisons between MDPs and LMDPs across different dynamics, problem sizes, or state space types.

3.2 Embedding of MDPs

The initial step in benchmarking the efficiency of algorithms for LMDPs against methods for traditional MDPs is to develop a precise embedding of an MDP into an analogous LMDP, to allow a fair and direct comparison. To achieve this, we employ an LP-relaxation approach in integer programming as described by [Todorov, 2006]. Additionally, this work incorporates several extensions to enhance the approximation across a diverse range of settings.

3.2.1 LMDP embedding of stochastic MDPs

Most of the reinforcement learning problems defined as Markov Decision Processes consist of stochastic dynamics. This is the closest simulation to real-world problems, where taking a specific action does not always lead to the same subsequent state due to inherent environmental uncertainties.

Often, each action has an expected next state, but there is a small probability that it could lead to a different state. This can be observed in Grid environments with slippery conditions, where an action intended to move the agent to a specific state might result in the agent slippering to an adjacent state. In other situations, actions consist of a probability distribution over the possible next states, similar to the dynamics in Linearly-solvable Markov Decision Process.

To facilitate the unification of MDP and LMDP notation, we denote the transition probabilities in the traditional MDP with $\tilde{\mathcal{P}}(s'|s,a)$ and the reward function with $\tilde{\mathcal{R}}(s,a)$. Accordingly, the LMDP notations will remain unmodified, with the the reward function as $\mathcal{R}_(s)$, the uncontrolled transition probabilities as $\mathcal{P}(s'|s)$, the controlled transition probabilities as $\mathcal{P}_{\mathbf{u}}(s'|s)$.

In the case of computing an LMDP embedding of an MDP, we aim to obtain an LMDP such that for each (s,a) the transition $\mathcal{P}_{\mathbf{u}^a}(\cdot|s) = \tilde{\mathcal{P}}(\cdot|s,a)$ has a reward $\mathcal{R}_{\mathbf{u}^a}(s) = \tilde{\mathcal{R}}(s,a)$, where $\mathbf{u}^a$ is the control vector corresponding to the symbolic action a. Namely, for each discrete action a in the traditional MDP we want a corresponding continuous action with the same reward and probability distribution. This requirement can be formally expressed as:

$$\mathcal{R}(s) - \sum_{s' \in \mathcal{S}} \tilde{\mathcal{P}}(s'|s,a) \log \frac{\tilde{\mathcal{P}}(s'|s,a)}{\mathcal{P}(s'|s)} = \tilde{\mathcal{R}}(s,a), \text{ for all } s \in \mathcal{S}^-, a \in \mathcal{A}. \quad (3.1)$$

This will be solved separately for each state s as a system of $|\mathcal{A}|$ equations, where $\tilde{\mathcal{P}}$ and $\tilde{\mathcal{R}}$ are known by the MDP itself and $\mathcal{P}$ and $\mathcal{R}$ are not. By fixing s we can define the vectors $\mathbf{m}$, $\mathbf{b}$ and the matrix D as follows:

$$\begin{aligned} m_{s'} &= \log \mathcal{P}(s'|s) \\ b_a &= \tilde{\mathcal{R}}(s,a) + \sum_{s' \in \mathcal{S}} \tilde{\mathcal{P}}(s'|s,a) \log \tilde{\mathcal{P}}(s'|s,a) \\ D_{as'} &= \tilde{\mathcal{P}}(s'|s,a) \end{aligned} \quad (3.2)$$

Hence, the system of equations becomes linear:

$$\mathcal{R}\mathbf{1} + D\mathbf{m} = \mathbf{b}, \quad (3.3)$$

where D, $\mathbf{b}$ are given, $\mathcal{R}$, $\mathbf{m}$ are unknown and $\mathbf{1}$ is a column vector of 1's. To focus on the possible next states it is necessary to remove the zero columns of D. Additionally, for an exact embedding to exist, one of the assumptions is that if any element $D(a,s')$ is zero, the entire column corresponding to s' must be 0.

If this assumption does not hold, we can replace the 0 elements with a small ϵ and renormalize the matrix. However, this situation is unlikely because it implies that a transition from s to s' has zero probability for one action but nonzero probability for the remaining actions, which is uncommon in MDPs with stochastic dynamics.

Since we need $\mathcal{P}$ to be a normalized probability distribution, we require:

$$\sum_{s' \in \mathcal{S}} e^{m_{s'}} = 1 \tag{3.4}$$

Despite this equation being nonlinear, it can be transformed into a linear problem. By definition of the MDP, we know that D is a stochastic matrix, which results in $D\mathbf{1} = \mathbf{1}$, simplifying equation (3.3) as:

$$D\left(\mathcal{R}\mathbf{1} + \mathbf{m}\right) = \mathbf{b}, \tag{3.5}$$

which can be solved for $\mathbf{c} = \mathcal{R}\mathbf{1} + \mathbf{m}$ using a linear solver. However, this does not apply in all cases. An MDP, whether characterized by deterministic or stochastic dynamics, may have variable transition dimensionalities. Specifically, the number of possible next states from a given state can vary, leading to a situation where some states have more potential transitions than others. This variability in the transition structure has a direct effect in the previous equations, since it determines the nature of D, it must be dealt with to accurately model and solve the MDP. We could manage this by removing the transitions that lead to blocked actions for each state, but this would result in an inhomogeneous transition system, complicating the process. Therefore, it is necessary to maintain a consistent number of symbolic actions across all states.

For example, consider a simple grid environment the agent can move to the adjacent cell on the right, left, up, or down. In such an environment, corner states may have only three possible states, since the two transitions leading towards the grid limits will make the agent remain on the current cell, while other states have four. Additionally, a grid environment with obstacles may result in several states having different possible transitions. Ordinary states have four possible transitions (to immediate neighbors), while states near obstacles or grid boundaries have fewer transitions. Figure (3.1) illustrates this grid world example, where black squares represent obstacles and the grayscale represents the distribution of transitions across the states.

In cases where the number of possible transitions does not match the number of potential next states, matrix D becomes non-square, which prevents the use of a traditional linear solver for solving equation (3.5).

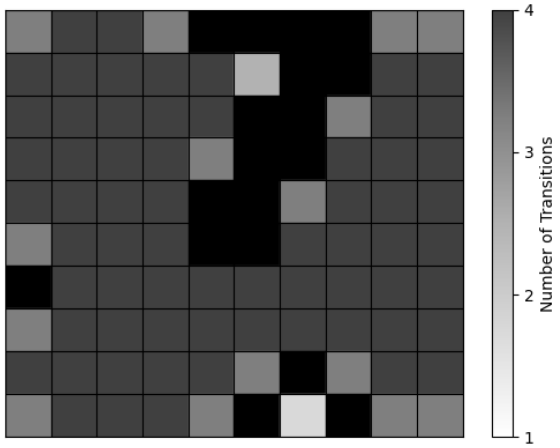

Figure 3.1: State transitions distribution in a grid world example.

Specifically, there are two possible scenarios where D is a non-square matrix for a given state s:

1. **More Symbolic Actions than Possible Next States**: This scenario makes D column rank deficient. In this case, D has more rows than columns, as there are more actions than possible next states. This results in a system with more equations than unknown variables, making it impossible to find an exact embedding.

2. **More Possible Next States than Symbolic Actions**: This scenario, though less common, makes D row rank deficient. Here, D has more columns than rows, resulting in a non-unique solution for $\mathbf{c}$. In this case, we should exploit this freedom in choosing $\mathbf{c}$ to improve the approximation of the MDP.

There are two possible approaches for dealing with such cases:

1. **Least Squares Solver:** Finds the solution for $\mathbf{c}$ that minimizes the sum of squares of the residuals $||D\mathbf{c} - \mathbf{b}||$. When D is full rank, the least squares method provides an exact solution with zero residuals.

2. **Moore-Penrose pseudoinverse:** When D is column-rank-deficient, meaning there is no exact solution, it provides the least-squares approximation. When there is no unique solution, it provides the minimum norm solution, which is the solution with the smallest Euclidean norm. Specifically, we can find $\mathbf{c} = D^{\dagger}\mathbf{b}$ to solve equation (3.5).

Now that we have found $\mathbf{c}$ for all the possible cases, $\mathbf{m} = \mathbf{c} - \mathcal{R}\mathbf{1}$ is a solution to (3.3) for any $\mathcal{R}$. Hence we can choose $\mathcal{R}$ to make $\mathbf{m}$ satisfy (3.4):

$$\mathcal{R} = \log \sum_{s' \in \mathcal{S}} e^{-c_{s'}} \tag{3.6}$$

The approach from [Todorov, 2006] did not consider absorbing states, applying the process uniformly to all states $s \in \mathcal{S}$. However, in our case, terminal states need a different approach. Since they are absorbing states, we only need to determine the appropriate $\mathcal{R}$. Specifically, we adopt the method from [Todorov, 2009], which embeds LMDP $\mathcal{R}$ into MDP $\tilde{\mathcal{R}}$ by assigning identical rewards to all actions, and apply it in the opposite direction, defining:

$$\mathcal{R}(s) = \frac{\sum_{a' \in \mathcal{A}} \tilde{\mathcal{R}}(s, a)}{|\mathcal{A}|} \tag{3.7}$$

Algorithm 4 Vectorized Embedding of stochastic MDP into LMDP

input: MDP $\mathcal{M}$ with $\mathcal{A}$, $\mathcal{S}$, $\mathcal{T}$, $\mathcal{S}^-$, $\tilde{\mathcal{R}}$ and stochastic $\tilde{\mathcal{P}}$, small ϵ
output: LMDP $\mathcal{L}$ with $\mathcal{R}$ and $\mathcal{P}$
initialize $D = \mathcal{P}$
Identify the number of potential next states $N(s)$ from each state $s \in \mathcal{S}^-$
Obtain the set $\mathcal{N}$ by collecting the unique values of $N(s)$ for all states $s \in \mathcal{S}^-$
for $n \in \mathcal{N}$ **do**
    Identify state space $\mathcal{S}^n$ of states s with $N(s) = n$
    Obtain D' for $\mathcal{S}^n$
    Remove zero columns from D', keeping only possible transitions
    Replace zeros in remaining columns with ϵ and renormalize
    $\mathbf{B} \leftarrow \mathcal{R}[\mathcal{S}^n] + \sum_{s' \in \mathcal{S}} D' \log(D')$
    Calculate pseudo-inverse $D^\dagger$ of D
    $\mathbf{C} \leftarrow D^\dagger \mathbf{B}$
    $\mathbf{R} \leftarrow \log(\sum_{s' \in \mathcal{S}} e^{-\mathbf{C})}$
    $\mathbf{M} \leftarrow \mathbf{C} - \mathbf{R}$
    $\mathcal{R}[\mathcal{S}^n] \leftarrow \mathbf{R}$
    $\mathcal{P}[\mathcal{S}^n] \leftarrow e^{\mathbf{M}}$
end for
$\mathcal{R}(\mathcal{T}) \leftarrow \left(\sum_{a \in \mathcal{A}} \tilde{\mathcal{R}}(\mathcal{T}, a)\right) / |\mathcal{A}|$
return $\mathcal{L}$ with $\mathcal{R}$ and $\mathcal{P}$

As one of the main objectives of this work is to develop efficient and scalable methods for LMDPs, the efficiency of the embedding techniques is crucial. Accordingly, we have implemented a vectorized version of Todorov's embedding theory [Todorov, 2009], using the Moore-Penrose pseudoinverse and avoiding a loop over all states. This approach, detailed in algorithm (4), offers significant improvements in efficiency as outlined in Appendix (E).

3.2.2 LMDP embedding of Deterministic MDPs

While a significant portion of the problems addressed by MDPs involve stochastic dynamics, this is not universally true for all settings. Many problems exhibit deterministic dynamics, where each discrete action leads the agent to a single, predictable outcome, leaving no room for uncertainty in the actions.

The dynamics of these deterministic MDPs result in a practical and straightforward understanding of the problem, making these settings simpler to solve. However, this practicality do not extend to embedding techniques, which become more complex due to the deterministic nature of the dynamics. Since the assumptions from the previous section are not met with these dynamics, an exact embedding cannot be constructed, thus an approximation method is required. The embedding technique described in previous section cannot be directly applied to deterministic MDPs for several reasons.

Firstly, equation (3.2) incorporates the entropy of the transition probability distribution in the MDP. When actions are deterministic, there is no entropy at all, making it impossible to define the equations in that manner. Several methods were explored within this work to address this issue. One straightforward approach could be removing the entropy factor from equation (3.2), and following the same procedure. However, without the entropy addition, vector $\mathbf{b}$ becomes negative, contrary to expectations, leading to an incorrect and unmanageable construction of the LMDP, as it implies that the reward function will be positive.

Upon examining the reward constructions under this method, we found significant observations. Analysis of different MDP with varying transition dynamics dimensionalities indicates that the constructed $\mathcal{R}$ differs from the original reward $\tilde{\mathcal{R}}$ of the MDP by at most $\log N(s)$, where $N(s)$ is the number of possible next states from state s. This has two important implications. First, the difference between the embedded rewards $\mathcal{R}$ and the original rewards $\tilde{\mathcal{R}}$ increases as $N(s)$ increases, which implicitly follows from (3.2). Second, when the original rewards are small in absolute terms (e.g., -1), $\log N(s)$ may be greater than the actual $\tilde{\mathcal{R}}$, causing a sign switch when entropy is lacking.

To address this issue, Todorov's proposal of scaling all rewards $\tilde{\mathcal{R}}$ by a negative constant[1] might seem like a suitable solution. This allows us to construct non-positive rewards, making it possible to build the LMDP. However, this approach essentially shifts the reward values by the constant, which, although nonproblematic, results in a suboptimal approximation. Furthermore, it would be preferable to develop an embedding that does not modify the inherent dynamics of the MDP and works for any reward function definition $\mathcal{R} : \mathcal{S} \to \mathbb{R}$.

An alternative method involves considering a stochastic "policy" to translate the MDP to an LMDP. This method results in:

$$\mathcal{R}(s) \leftarrow \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \tilde{\mathcal{R}}(s, a), \text{ for all } s \in \mathcal{S} \tag{3.8}$$

$$\mathcal{P}(s'|s) \leftarrow \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \tilde{\mathcal{P}}(s'|s, a), \text{ for all } s \in \mathcal{S}^{-}, s' \in \mathcal{S} \tag{3.9}$$

This method provides a suitable approximation of $\mathcal{P}$, as deterministic actions are transformed into a stochastic distribution naturally. Nevertheless, the reward averaging leads to a suboptimal approximation. Referring to equation (2.2), the optimal approximation would be:

$$\mathcal{R}(s) = \tilde{\mathcal{R}}(s, a) + \lambda KL\left(\mathcal{P}_{\mathbf{u}} \middle\| \mathcal{P}\right) \tag{3.10}$$

However, obtaining $\mathcal{P}_{\mathbf{u}}$ requires a method such as power iteration (2.9), which in turn requires knowledge of $\mathcal{R}(s)$, creating a recursive dependency. To resolve this, an alternative iterative method is needed. One approach is to use the dynamics defined the dynamics in equations (3.8) and (3.9) to obtain the value function through power iteration, then use this value to compute $\mathcal{P}_{\mathbf{u}}$, and finally update $\mathcal{R}$ as in equation (3.10). This approach, described in algorithm (5), led to better approximations than previous alternatives.

This procedure is expected to enhance the approximations if more iterations were considered, updating the reward $\mathcal{R}$ with the new computations of $\mathcal{P}_{\mathbf{u}}$ at each step. However, through empirical evaluation across multiple settings, it was observed that the best approximation was always found in the very first iteration. This unexpected finding led us to develop an alternative method to refine the embedding process.

[1] Theory from [Todorov, 2006] is cost-based, using non-negative costs instead of non-positive rewards. Thus, scaling by a positive integer in his approach equates to scaling by a negative integer in our case.

Algorithm 5 Deterministic MDP Embedding through stochastic policy averaging

1: **input:** MDP $\mathcal{M}$ with $\mathcal{A}$, $\mathcal{S}$, $\mathcal{T}$, $\mathcal{S}^-$, $\tilde{\mathcal{R}}$ and deterministic $\tilde{\mathcal{P}}$, temperature parameter λ
2: **output:** LMDP $\mathcal{L}$ with $\hat{\mathcal{R}}$ and $\mathcal{P}$
3: $\hat{\mathcal{R}}(\cdot) \leftarrow \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \tilde{\mathcal{R}}(\cdot, a)$
4: $\mathcal{P}(\cdot|\cdot) \leftarrow \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \tilde{\mathcal{P}}(\cdot|\cdot, a)$
5: Obtain Z from power iteration method for LMDP $\mathcal{L}$ with $\mathcal{P}$ and $\hat{\mathcal{R}}$
6: $\mathcal{P}_{\mathbf{u}^*} \leftarrow \mathcal{P}Z / \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|\cdot)Z(s')$
7: $\hat{\mathcal{R}} \leftarrow \hat{\mathcal{R}} + \lambda KL(\mathcal{P}_{\mathbf{u}^*} \| \mathcal{P})$
8: **return** $\mathcal{L}$ with $\hat{\mathcal{R}}$ and $\mathcal{P}$

Given that the best approximation comes from the first iteration (i.e., from the exact process in algorithm (5)), we hypothesize that there is a factor K such as the optimal approximation of the embedded LMDP reward will be:

$$\mathcal{R}(s) = K \cdot \hat{\mathcal{R}}(s) \text{ for all } s \in \mathcal{S}, \tag{3.11}$$

where $\hat{\mathcal{R}}(s)$ is the initial approximation of $\mathcal{R}$, returned from (5). The optimal value of factor K can be found using a search algorithm, with the value function approximation error between the embedded LMDP and the original MDP as the objective function to be minimized. Binary search was initially employed, resulting in noticeably better results than all the methods commented so far.

However, although the value function is monotonically decreasing as a function of K, the approximation error with the optimal value function does not follow a monotonic pattern. Instead, it exhibits a unimodal distribution, which means there is a single local minimum within the range of K. This characteristic makes the problem more suitable for ternary search rather than binary search

Ternary search is preferred in this context because it is specifically designed for finding the minimum or maximum of a unimodal function. It divides the range into three parts and systematically narrows the search interval based on comparisons, making it more efficient and reliable than binary search for this type of problem. Algorithm (6) describes the full approach developed within this work, which clearly outperforms the previous embedding techniques, as well as the baseline embedding proposal, which is not suitable for this class of MDPs.

The improvements achieved by our method are illustrated Figure 3.2, where various discussed approaches are compared in terms of the value function V approximation. The four scatter plots compare the optimal value function from the original MDP with the value function from the embedded LMDP at 3,600 state-space points.

Algorithm 6 Deterministic MDP Embedding through ternary search

input: MDP $\mathcal{M}$ with $\mathcal{A}$, $\mathcal{S}$, $\mathcal{T}$, $\mathcal{S}^-$, $\tilde{\mathcal{R}}$ and deterministic $\tilde{\mathcal{P}}$, temperature parameter λ, small ϵ, optimal value function V^* from $\mathcal{M}$
output: LMDP $\mathcal{L}$ with $\mathcal{R}$ and $\mathcal{P}$
$\hat{\mathcal{R}}(\cdot) \leftarrow \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \tilde{\mathcal{R}}(\cdot, a)$
$\mathcal{P}(\cdot|\cdot) \leftarrow \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \tilde{\mathcal{P}}(\cdot|\cdot, a)$
Obtain Z from power iteration method for LMDP $\mathcal{L}$ with $\mathcal{P}$ and $\hat{\mathcal{R}}$
$\mathcal{P}_{\mathbf{u}^*} \leftarrow \mathcal{P}Z / \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|\cdot)Z(s')$
$\hat{\mathcal{R}} \leftarrow \hat{\mathcal{R}} + \lambda KL(\mathcal{P} \| \mathcal{P}_{\mathbf{u}^*})$
Set $K_{min} = 0$ and $K_{max} = 1$
while $K_{max} - K_{min} \geq \epsilon$ **do**
    $m_1 = K_{min} + (K_{max} - K_{min})/3$
    $R_1 = m_1 \cdot \hat{\mathcal{R}}$
    Obtain Z_1 from power iteration method for LMDP $\mathcal{L}$ with $\mathcal{P}$ and R_1
    $V_1 = \lambda \log Z_1$
    $MSE_1 = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} (V_1 - V^*)^2$
    $m_2 = K_{max} - (K_{max} - K_{min})/3$
    $R_2 = m_2 \cdot \hat{\mathcal{R}}$
    Obtain Z_2 from power iteration method for LMDP $\mathcal{L}$ with $\mathcal{P}$ and R_2
    $V_2 = \lambda \log Z_2$
    $MSE_2 = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} (V_2 - V^*)^2$
    if $MSE_1 > MSE_2$ **then**
        $K_{min} = m_1$
    else
        $K_{max} = m_2$
    end if
end while
$\mathcal{R} = K_{min} * \hat{\mathcal{R}}$
return $\mathcal{L}$ with $\mathcal{P}$ and $\mathcal{R}$

The best results are obtained with both the Ternary Search-based (TS) embedding approach, described in the previous algorithm, and the Binary Search-based (BS) approach. Both methods achieve an R^2 of 0.9997, indicating that either method could be chosen. However, since this work aims to enable a fair comparison between MDPs and LMDPs, ternary search was chosen for its slightly superior performance.

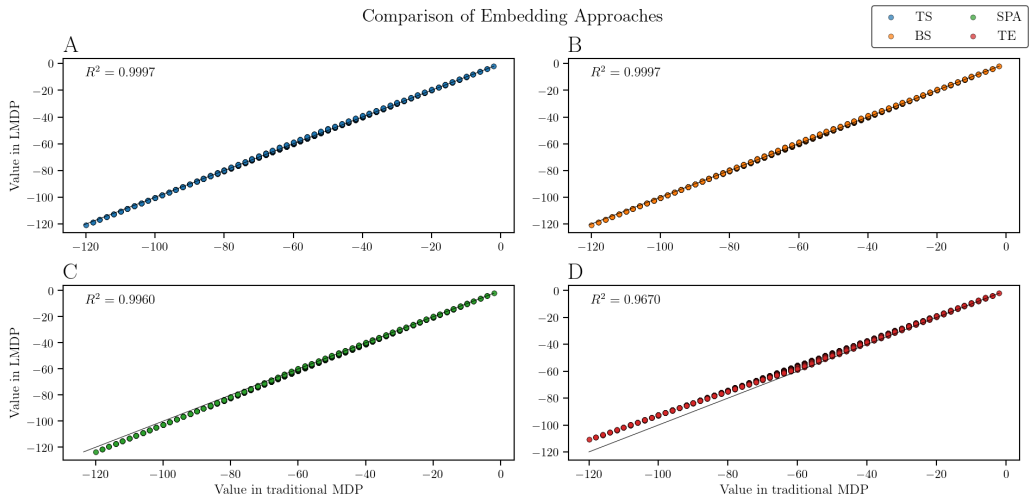

Figure 3.2: Embedding Approximations by different approaches.

Specifically, the ternary search-based embedding achieved a Mean Squared Error (MSE) of 0.1525, while the binary search approach yielded an MSE of 0.1527. Although this difference is practically negligible, given the lack of an exact embedding for deterministic cases and the overall less accurate approximations compared to stochastic cases, the best available approximation was selected. A significant distinction can be observed between the developed method and Todorov's original embedding (TE) approach. Todorov's method achieved an R^2 of 0.967 and an MSE of 19.8, indicating that our method offers a 99.23% improvement over the baseline. Additionally, the stochastic policy averaging (SPA) method also outperformed Todorov's approach, achieving an R^2 of 0.996 and an MSE of 2.4. However, the final method developed in this work outperformed SPA by 93.75%, substantiating our previous discussion. It is evident that the approximation accuracy diminishes for larger state values, as depicted in Figures (3.2C) and especially in (3.2D), in comparison to both (3.2A) and (3.2B). This performance discrepancy becomes more pronounced in larger settings, as highlighted in Figure (F.1).

To provide a more comprehensive analysis of the performance of the different embedding approaches, we have plotted the Mean Squared Error as a function of the number of states in Figure (3.3). This plot, presented in a logarithmic scale, corroborates our previous findings, illustrating the superior accuracy of the ternary search-based embedding method (TS) compared to the other approaches. As the number of states increases, the TS method consistently maintains a lower MSE, demonstrating its robustness and scalability. The stochastic policy averaging method (SPA) also shows significant improvement over Todorov's embedding (TE), yet it is still outperformed by the TS method by a considerable margin.

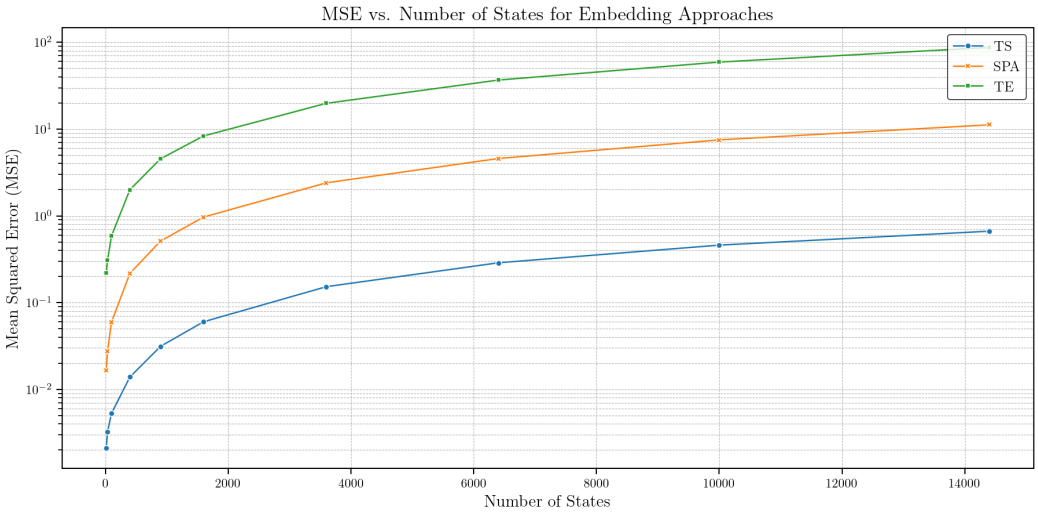

Figure 3.3: MSE vs. Number of States for Embedding Approaches.

This further supports the claim that our method provides a more precise approximation, especially in larger state spaces, highlighting its potential for more complex MDPs. The observed trends emphasize the importance of selecting an appropriate embedding technique, particularly for applications involving extensive state spaces where precision is critical. The performance difference becomes more pronounced with increasing problem size, underscoring the efficiency and effectiveness of the TS method for large-scale reinforcement learning tasks.

3.3 Embeddings of LMDP into MDP

The goal of this work is to facilitate a fair comparison between methods for MDPs and LMDPs, enabling benchmarking and evaluation of their effectiveness across both frameworks. Consequently, one of the primary objectives is to establish a standard method for comparing these two paradigms. While the aim is not to construct LMDPs from MDPs, it is essential to create pairs of problems that allow for fair comparisons between different methods. Therefore, an embedding from LMDP to MDP is also required. To achieve this, we utilize Todorov's proposal [Todorov, 2006, Todorov, 2009], which is more straightforward than the embedding from MDPs to LMDPs.

The initial step is to compute the optimal controlled transition probability distribution $\mathcal{P}\mathbf{u}^*$ through power iteration (see equation 2.9). We then construct an MDP with one action a that has the same transition probabilities as $\mathcal{P}_{\mathbf{u}^*}$. For each state s, we also define $N(s)-1$ other symbolic actions, whose transition probabilities are derived from $\mathcal{P}_{\mathbf{u}^*}(\cdot \mid s)$ by circular shifting. This ensures that all actions have the same probability distribution, but peaked at different states. This method is applicable to both deterministic and stochastic-dynamics based LMDPs, since $\mathcal{P}_{\mathbf{u}^*}$ can be constructed stochastically, resulting in a stochastic MDP.

Algorithm 7 Vectorized Embedding of LMDP into MDP

input: LMDP $\mathcal{L}$ with $\mathcal{R}$ and $\mathcal{P}$, temperature parameter λ
output: MDP $\mathcal{M}$ with $\tilde{\mathcal{P}}$ and $\tilde{\mathcal{R}}$
Obtain Z from power iteration method for LMDP $\mathcal{L}$ with $\mathcal{R}$ and $\mathcal{P}$
$\mathcal{P}_{\mathbf{u}^*} \leftarrow \mathcal{P}Z / \sum_{s' \in \mathcal{S}} \mathcal{P}(s''|\cdot)Z(s')$
$\max(a) \leftarrow \max N(s)$ for all $s \in \mathcal{S}^-$
$\tilde{\mathcal{R}}(\cdot, a) \leftarrow \mathcal{R}(\cdot) - \lambda KL\left(\mathcal{P}_{\mathbf{u}^*}(\cdot \mid \cdot) \| \mathcal{P}(\cdot \mid \cdot)\right)$ for all actions a
Identify the number of potential next states $N(s)$ from each state $s \in \mathcal{S}^-$
Obtain the set $\mathcal{N}$ by collecting the unique values of $N(s)$ for all states $s \in \mathcal{S}^-$
for $n \in \mathcal{N}$ **do**
    Identify state space $\mathcal{S}^n$ of states s with $N(s) = n$
    Identify state space $\mathcal{S}'^n$ of potential next states s' of states s in $\mathcal{S}^n$
    for action a in $\max(a)$ **do**
        $\tilde{\mathcal{P}}(\mathcal{S}'^n \mid \mathcal{S}^n, a) \leftarrow \mathcal{P}_{\mathbf{u}^*}(\mathcal{S}'^n \mid \mathcal{S}^n)$ shifted a times.
    end for
end for
return $\mathcal{M}$ with $\tilde{\mathcal{R}}$ and $\tilde{\mathcal{P}}$

Each one of these actions will involve a reward $\tilde{\mathcal{R}}(s,a)$ defined oppositely to equation (3.10), ensuring that all actions have the same reward for a given state s. This results in an MDP that is guaranteed to have an identical optimal value function to the original LMDP.

Additionally, instead of defining $N(s)$ different actions for each state s, we define $\max(a) \leftarrow \max N(s)$ for all $s \in \mathcal{S}^-$ to avoid inhomogeneous transition probability dimensionalities. For states s where $N(s) < \max(a)$, the remaining $\max(a) - N(s)$ actions will have the same transition as the prior actions, following the circular shifting procedure. This causes row-rank deficiency in the transition probability matrix, as several rows will have linear dependencies. However, due to the approach followed in Algorithm (4), an exact approximation can still be constructed. Direct matrix operations cannot be performed if the LMDP has different transition dimensionalities, similar to the embedding in the opposite direction. Therefore, the same strategy is used, iterating through subsets of states according to their transition dimensionalities $N(s)$.

This embedding approach is already optimal for constructing approximations in both stochastic and deterministic settings of LMDPs. However, the iterative solution over all states can be optimized for efficiency, as shown in Algorithm (7), leading to significant efficiency improvements as highlighted in Appendix (E.2).

Chapter 4

EXPERIMENTATION

In this chapter, we describe the various environments utilized for experimentation within the Grid World domain and present the corresponding results. The methodologies developed in this work, including the extended embedding techniques and the implementations of MDP and LMDP frameworks, are publicly accessible[1].

4.1 Grid World Domain

The environments used for our experiments are based on the Grid World domain. Both environments feature non-positive rewards and obstacles, but differ in their complexity and dynamics.

The first environment is a Grid problem with straightforward dynamics. Here, the agent can move to any of its neighboring cells. The transitions from each state are equiprobable to each neighboring state, except for states adjacent to the grid boundaries or objects. In these cases, the transition probability for the missing direction is assigned to the state itself, creating non-uniform transition dynamics. As discussed in the embeddings chapter, this results in heterogeneous transition probabilities when decomposing non-zero values.

The rewards in this environment are non-positive, eliminating the need for discounting since negative rewards naturally encourage the minimization of episode length. The goal state has a reward of 0, and all non-terminal states receive a reward of -1. Figure (4.1) illustrates an example of this environment, including the optimal value function for each state. In this figure, black cells denote obstacles.

[1] https://github.com/davidperezcarrasco/Efficient-Algorithms-for-Linearly-Solvable-Markov-Decision-Processes

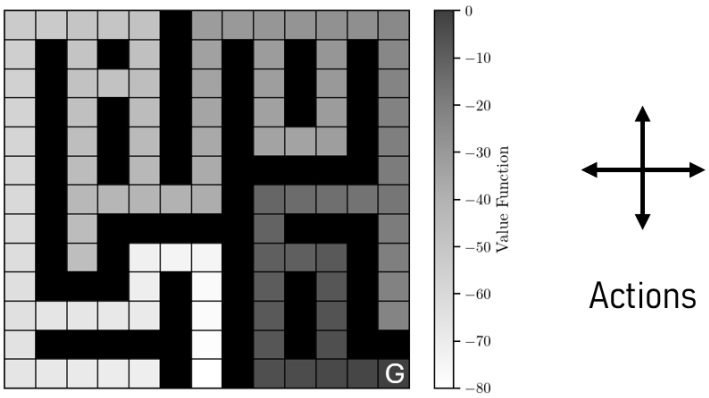

Figure 4.1: Simple Grid Environment

The second environment extends the simple grid environment by incorporating the agent's orientation. In this setting, the agent can face upwards, downwards, leftwards, or rightwards, resulting in a state representation that includes both the agent's position and its orientation. This extension increases the state space by a factor of four compared to a simple grid of the same size.

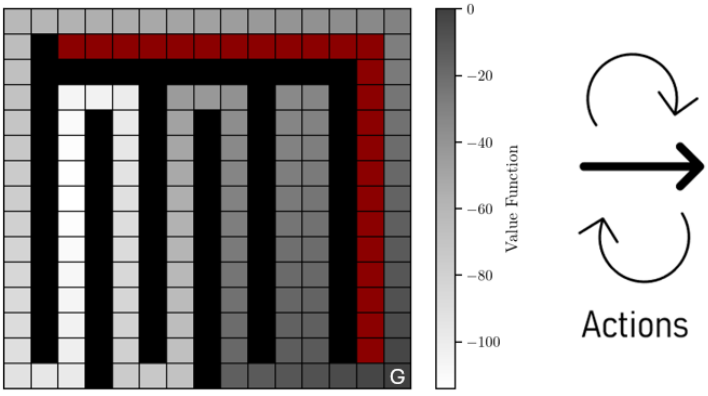

Figure 4.2: Minigrid Environment with directional states

The actions available to the agent in this domain include rotating left, rotating right, and moving forward, adhering to the dynamics of the Gymnasium Minigrid framework. This environment also features undesirable terminal states with significantly higher negative rewards to ensure that these states are less desirable than any other state, regardless of their distance from the goal. Unlike the simple grid environment, this directional grid environment does not encounter transition dimensionality issues because the agent always has directional states available. If the neighboring cell contains an obstacle, the agent remains in its current state. Figure (4.2) provides an example of this environment.

This example environment emulates the "Cliff-Walking" problem commonly discussed in reinforcement learning literature [Sutton and Barto, 2018]. The agent can choose between two main paths: one that is extremely dangerous but expedient and another that is safer but longer. The red cells denote undesirable terminal states, which have significantly higher negative rewards to discourage the agent from reaching them. In our illustrations, these terminal states are excluded from the value function calculations to prevent distortion of the value function scale. The specific settings for these environments are detailed in Appendix (H).

4.2 Experiments and Results

For the comparative analysis of LMDPs with traditional MDPs, we employed both Q-learning and Z-learning algorithms (12, 13). The Z-learning training routine utilized a learning rate decay strategy $\alpha_k = \frac{c}{c+k}$, where $t(k)$ represents the number of episodes to which the sample k belongs [Todorov, 2006]. The constant c was meticulously optimized for each algorithm. As detailed in Appendix (D), the optimal values were determined to be $c = 500,000$ for MDPs and $c = 20,000$ for LMDPs. Discounting strategies were also considered, as showcased, but since we are using non-positive rewards, there already exists an incentive for the agent to minimize the episode's length, leading to an optimal choice $\epsilon = 1$.

4.2.1 Optimization of Exploration-Exploitation Trade-off

One of the pivotal aspects of reinforcement learning is the exploration-exploitation trade-off. Effective exploration is critical for an agent's learning success as it directly impacts the ability to discover optimal policies. To simulate and compare the behaviors of Z-learning with Q-learning more accurately, we adopted an ϵ-greedy approach with an ϵ-decay mechanism. Various ϵ values and decay coefficients were tested to identify the most effective strategy for Q-learning, thereby enabling a fair comparison with Z-learning. These explorations and their findings are comprehensively discussed in Appendix (D). The optimal configuration determined for the agent's exploration-exploitation strategy involves an initial ϵ value set to 1, coupled with a high decay rate. This setup ensures ample exploration during the early stages, gradually transitioning towards exploitation as the learning process progresses.

The tuning of exploration parameters unveiled a clear trade-off between the speed of convergence and the precision of the policy approximation. As illustrated in Figure (4.3), lower ϵ-decay values lead to significantly faster convergence, albeit at the expense of reduced policy accuracy.

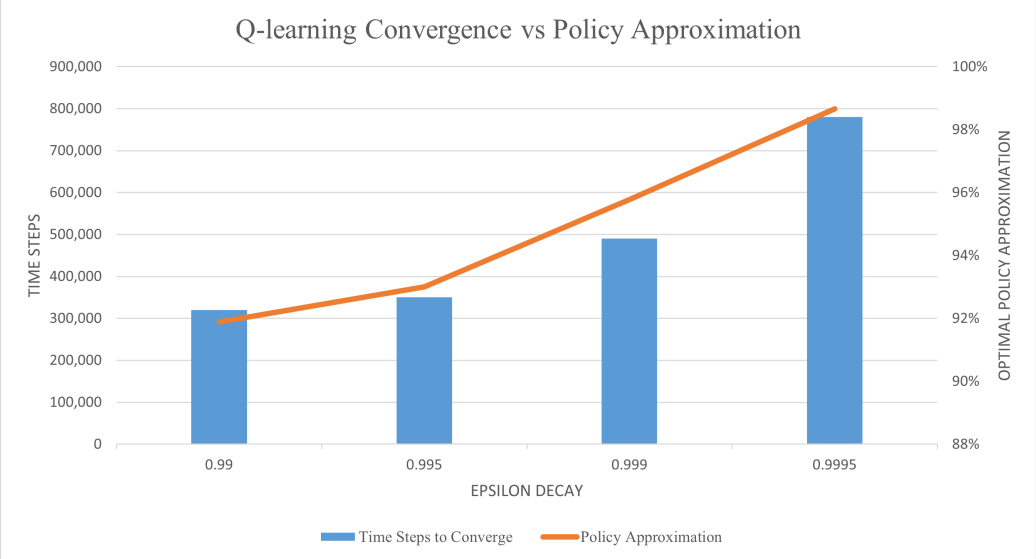

Figure 4.3: Q-learning convergence and policy approximation by epsilon decay

Specifically, an ϵ-decay of 0.99 achieved convergence more than twice as fast compared to an ϵ-decay of 0.9995, but with approximately 6% lower precision. These experiments were conducted in a basic Minigrid environment devoid of obstacles, which simplified the agent's task. However, in more complex environments with numerous obstacles, the policy error was markedly higher for strategies with low exploration. Figure (4.4) presents the primary differences between the ϵ-decay values in terms of both episodic reward and approximation error. The episodic reward, measured by aggregating all rewards from the visited states within an episode, is plotted on a logarithmic scale. The approximation error, calculated as the Mean Squared Error between the optimal value function v^* and its approximation at each time step, is depicted on a symmetrical logarithmic scale.

It is evident that larger values of ϵ-decay, indicative of prolonged exploration, result in extremely accurate approximations, achieving an error scale unattainable by other approaches. However, this strategy impedes the episodic reward from converging within a reasonable timeframe. Conversely, smaller values of ϵ-decay, indicative of brief exploration, facilitate noticeably faster convergence but fail to achieve desirable precision in the value function approximation. Consequently, it is imperative to consider exploration strategies that strike an optimal balance between convergence rate and approximation precision. To achieve efficient methods with accurate approximations, most experiments determined ϵ-decay within the range of 0.999 to 0.9995.

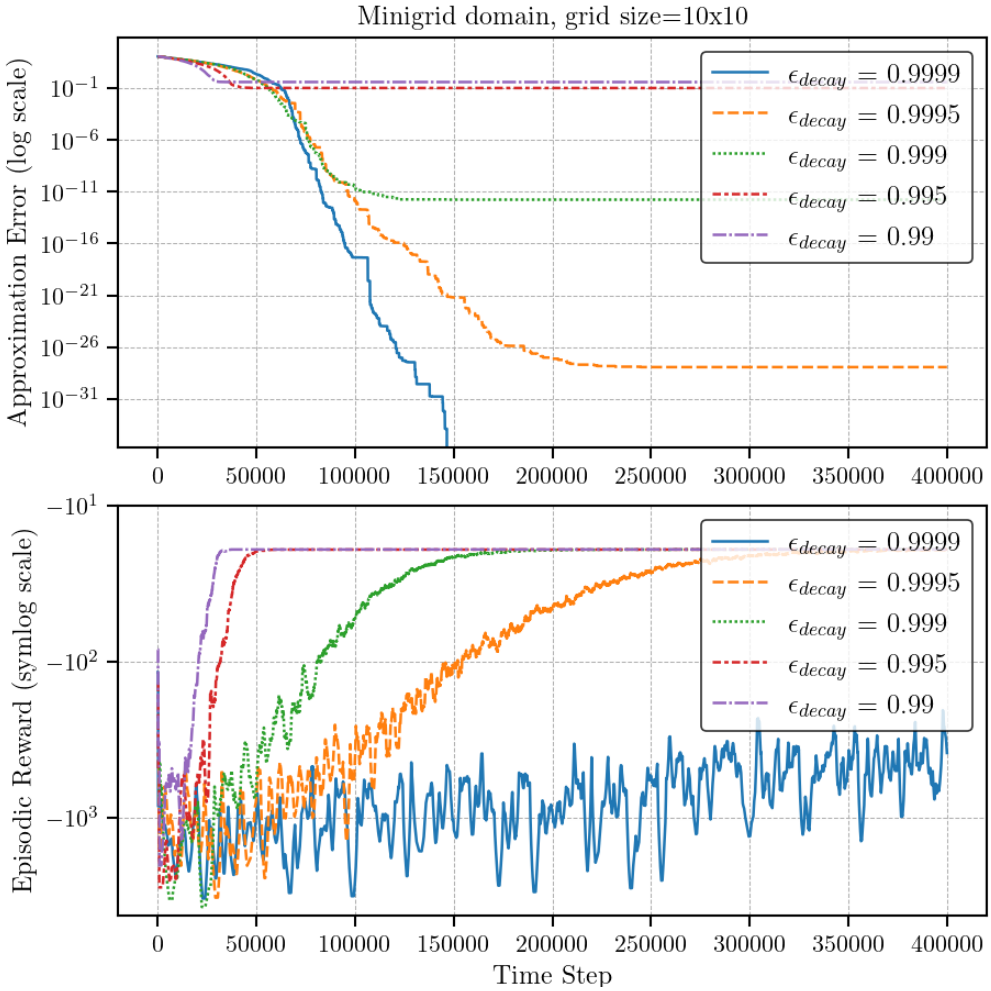

Figure 4.4: Comparison of approximation error (top) and episodic reward (bottom) for different ϵ-decay values.

Nevertheless, in specific challenging settings, such as maze-like environments, a faster exploration decay was necessary to achieve convergence. The lower bound for the exploration rate parameter, $\min_\epsilon$, was set to zero in all experiments. However, this bound was not reached in scenarios with higher exploration decay rates. Another crucial factor in the exploration dilemma is the randomness at the episode restart. Allowing random restarts for the agent helps it reach more unseen states, thereby enhancing the value approximations. However, for specific goal-oriented settings, such as maze environments or the hill cliff emulation, a single initial state was required for all episodes. To ensure a more consistent and controlled analysis, all experiments were conducted with a single initial state for all episodes.

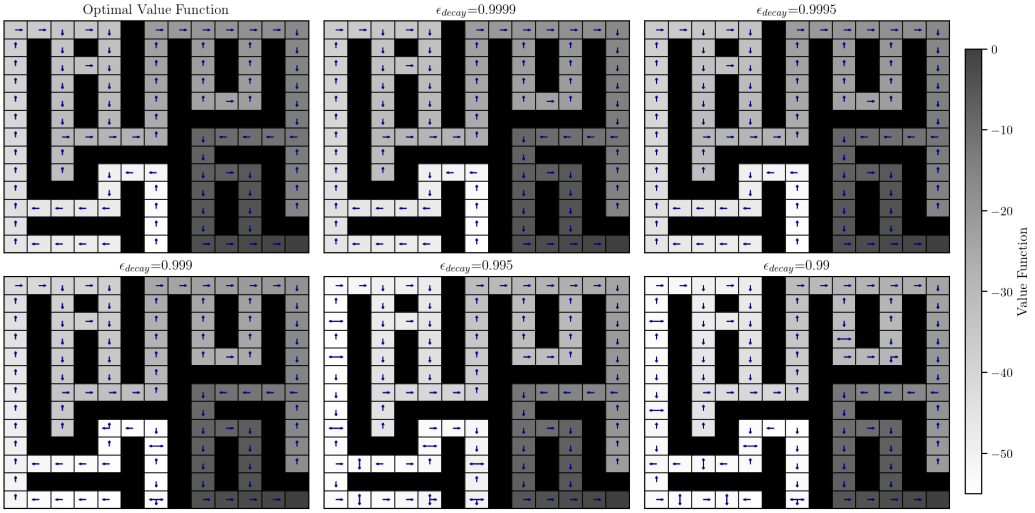

Figure 4.5: Value function and policy approximations for different ϵ-decay values in a grid world maze domain.

Figure (4.5) presents the value function approximations resulting from various ϵ-decay rates in Q-learning, along with the optimal value function v^*. Additionally, it displays both the optimal and greedy policies. The figure clearly demonstrates that higher ϵ-decay rates yield more accurate value function approximations and converge towards the optimal policy. In contrast, lower ϵ-decay rates produce suboptimal policy approximations, despite the rapid convergence observed in Figure (4.4).

4.2.2 Comparative Analysis of Z-learning and Q-learning

To benchmark LMDPs against standard MDPs, we utilized the embeddings previously defined to ensure a fair comparison. Both Z-learning and Q-learning methods were employed as described earlier in this thesis. The results of our experiments are depicted in the following figures, demonstrating the performance of Z-learning compared to Q-learning.

The first set of experiments was conducted in a 20x20 Minigrid environment, consisting on 9 different interconnected rooms. This setting was suitable for the benchmarking of the efficiency adn scalability of both frameworks, as it consisted in around 1,500 states. Figure (4.6) presents the value function approximation error (logarithmic scale) and episodic reward (symmetrical logarithmic scale) for both methods in this environment. The results highlight the distinct advantages of Z-learning over Q-learning.

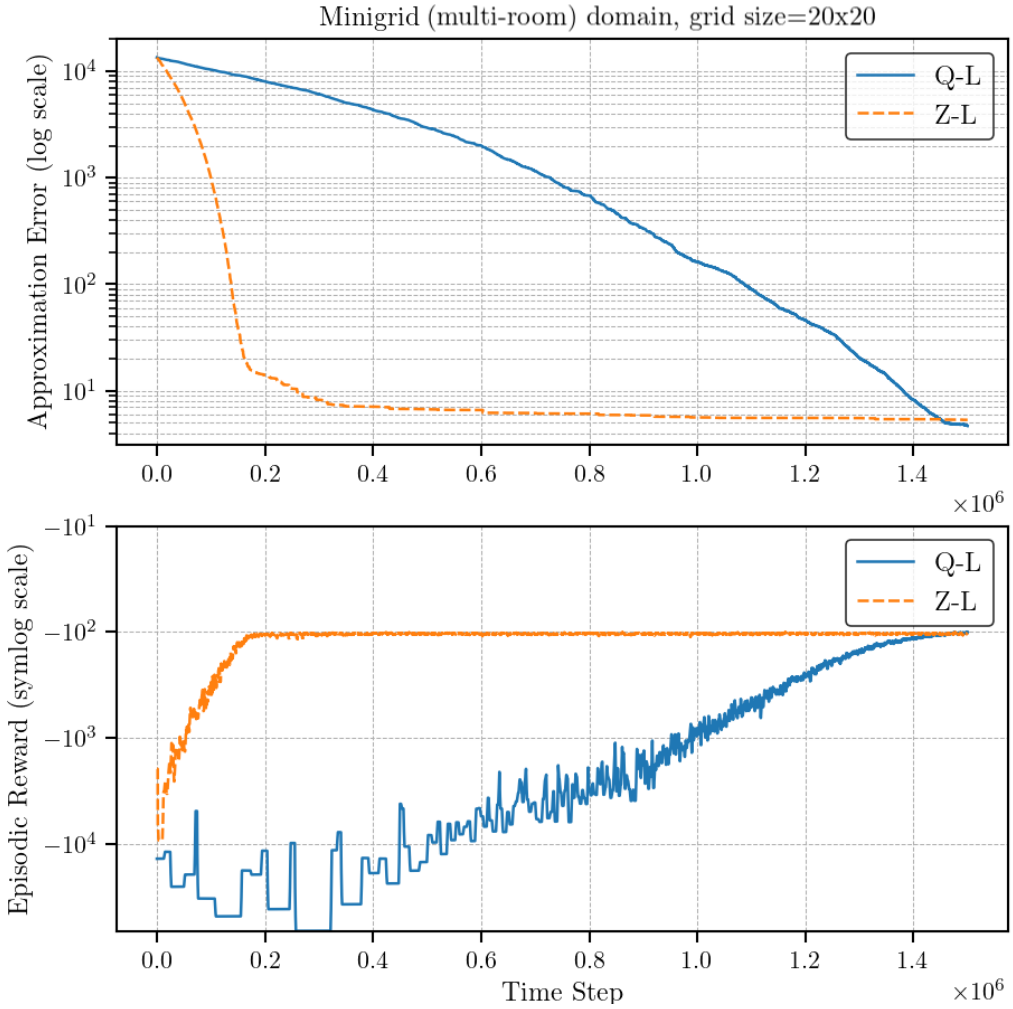

Figure 4.6: Comparison of approximation error (top) and episodic reward (bottom) between Q-learning and Z-learning in a Minigrid multi-room domain.

Although both approaches ultimately achieve optimal episodic reward values, Z-learning converges significantly faster, requiring only one-tenth of the time needed by Q-learning. This efficiency is also reflected in the approximation error, where Z-learning attains a highly accurate value function approximation much quicker than Q-learning. Consequently, Z-learning demonstrates a 90% reduction in computational cost compared to Q-learning. This substantial decrease in computational demand underscores the enhanced efficiency and scalability of LMDPs, particularly in larger domains.

Figure (4.7) presents the value function and policy approximations for both Q-learning and Z-learning in the 20x20 Minigrid environment. Since LMDPs do not have explicit policies but instead utilize a controlled state transition probability distribution $\mathcal{P}_{\mathbf{u}}$, a comparative approach was employed to evaluate both MDPs and LMDPs on a common metric. The optimal exponential value function derived from Z-learning was converted to the regular value function using the transformation $v(s) \leftarrow \lambda z(s)$. Subsequently, for each state, the neighboring transition with the highest $v(s)$ was selected as the optimal policy, thereby simulating the policy determination process in an MDP.

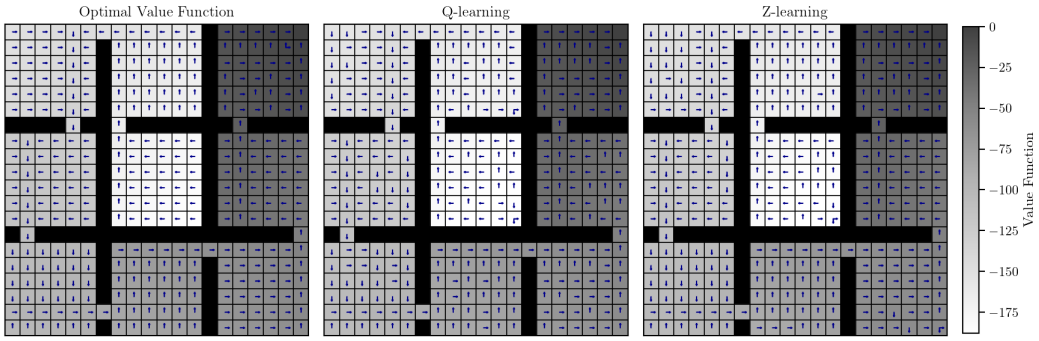

Figure 4.7: Value function and policy approximations for Q-learning and Z-learning in a Minigrid multi-room domain.

The results demonstrate that both Q-learning and Z-learning yield accurate value function approximations, resulting in nearly optimal policies. Notably, there are discrepancies in states that necessitate extensive exploration to reach. This limitation could be mitigated by increasing the initial sampling randomness. However, incorporating random sampling introduces variability in paths across episodes, complicating the benchmarking of episodic rewards. Despite these challenges, the comparative analysis reveals that both methods are capable of producing effective value function approximations, though Z-learning offers distinct computational advantages.

Another set of experiments was conducted in the Minigrid Hill Cliff emulation domain. This environment is particularly insightful for evaluating the online learning capabilities of MDPs and LMDPs in scenarios that include non-desirable terminal states. Figure (4.8)illustrates the episodic rewards and approximation errors achieved by Z-learning and Q-learning over time. The results indicate that Z-learning converges significantly faster than Q-learning, requiring nearly one-tenth of the iterations needed by Q-learning.

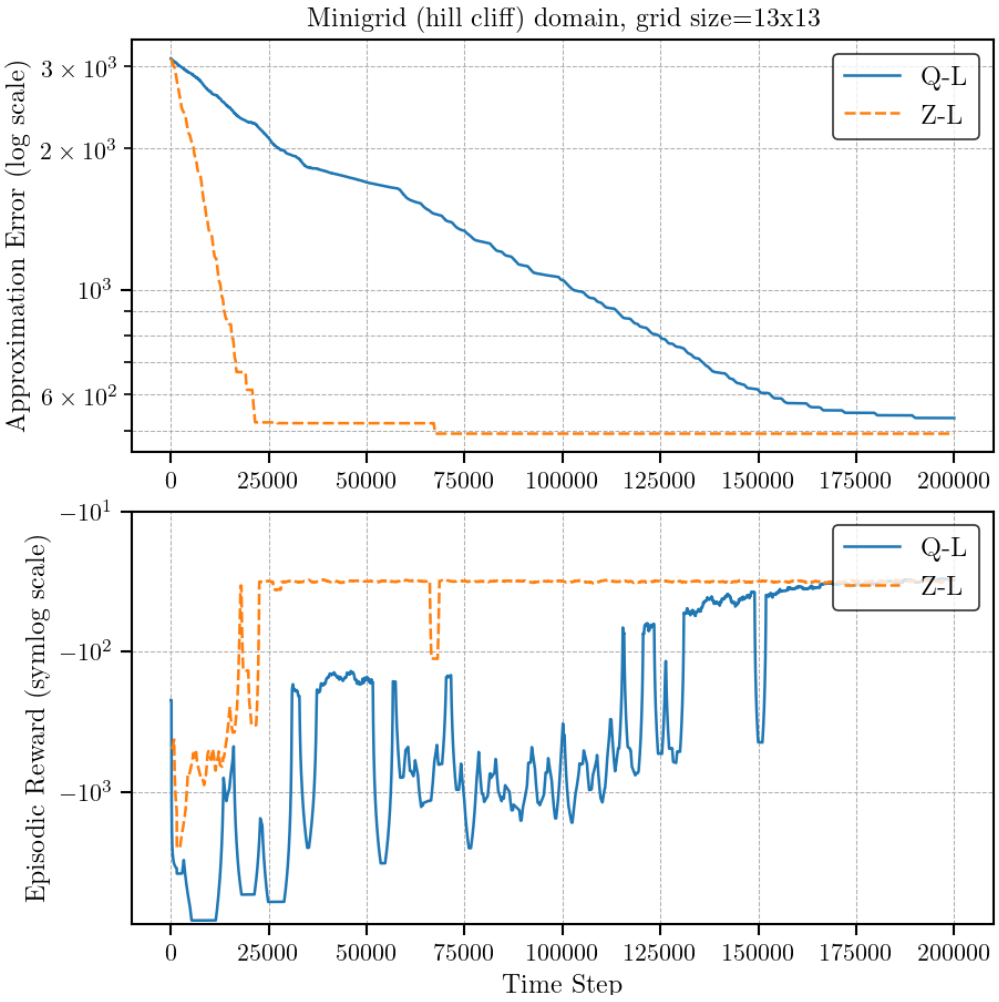

Figure 4.8: Comparison of approximation error (top) and episodic reward (bottom) between Q-learning and Z-learning in a Minigrid hill-cliff domain.

Furthermore, it is evident that Q-learning struggles to reach the same level of approximation error as Z-learning in this domain. This disparity is primarily due to the exploration strategy of the MDP algorithm. In environments with a high risk of encountering non-desirable terminal states, such as the Hill Cliff domain, it becomes exceedingly challenging to find optimal state value approximations. While an alternative exploration strategy could potentially improve the approximation error, it would significantly slow down the convergence rate. Given the high risk associated with increased exploration in this setting, achieving convergence would be incredibly difficult and thus, such a strategy was not considered desirable.

In contrast, Z-learning samples from the optimally controlled transition probability that is updated at each step. Since it is based on the z-function, which equates to the exponential reward in terminal states, terminal states with high negative rewards are very unlikely to be selected from the beginning of the online process. This makes the LMDP exploration behavior more suitable for scenarios like the Hill Cliff domain, where avoiding non-desirable terminal states is crucial.

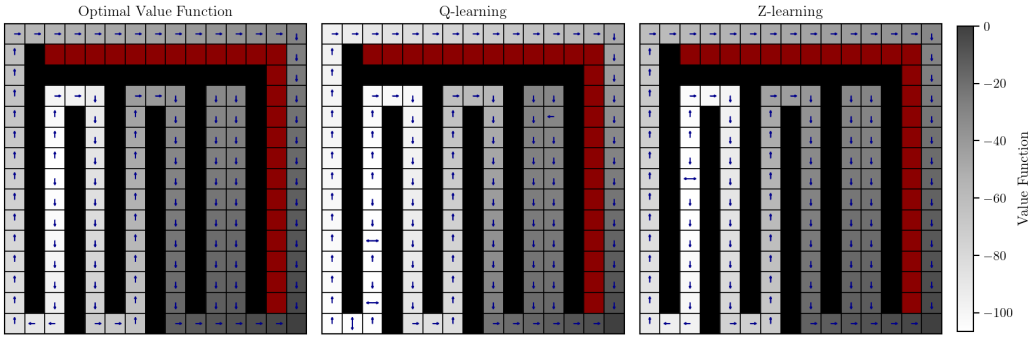

Figure 4.9: Value function and policy approximations for Q-learning and Z-learning in a Minigrid hill-cliff domain.

Figure (4.9) presents the value function approximations of Q-learning and Z-learning in the Hill Cliff domain, alongside the optimal value function. In this scenario, the approximations are less accurate compared to the previous environment, primarily due to the presence of non-desirable terminal states. This discrepancy is evident when comparing the approximation errors in Figure (4.9) with those in Figure (4.7). Thus, both Q-learning and Z-learning face challenges in accurately approximating values and policies for states that require extensive exploration.

Notably, Q-learning exhibits a slightly worse approximation than Z-learning, as indicated by the increased number of incorrect policy approximations. This is consistent with the observed difference in approximation errors shown in Figure (4.8). Interestingly, both methods particularly struggle with accurately approximating the values of states near the “cliff” due to the inherent high risk associated with these states. This high risk often leads to failed episodic terminations when these states are reached, complicating the accurate computation of their values.

Chapter 5

CONCLUSIONS AND FUTURE WORK

This thesis has presented an in-depth analysis of Linear Markov Decision Processes and their distinct advantages over traditional Markov Decision Processes. Our research has demonstrated that LMDPs, particularly when utilizing Z-learning algorithms, offer substantial improvements in scalability and computational efficiency. Experimental evaluations conducted in the Minigrid Hill Cliff and Multi-Room domains have underscored Z-learning's capability to achieve faster convergence and more precise value function approximations compared to Q-learning. These findings emphasize the efficacy of employing linear programming techniques for optimal action selection in sequential decision-making problems.

A key contribution of this work is the formulation of MDP-LMDP embeddings that are applicable across a wide range of settings and problem dynamics. This development is highly significant as it allows for the precise construction of equivalent problems within these two frameworks, facilitating exact and fair comparisons. The ability to accurately translate problems between MDPs and LMDPs is a critical advancement, offering a valuable tool for researchers and practitioners aiming to leverage the strengths of both approaches. This extension ensures that performance evaluations accurately reflect the inherent capabilities of each model, thereby providing a robust framework for future research endeavors.

Additionally, this thesis has conducted an exhaustive analysis of the influence of various design parameters on the performance of these algorithms. Through meticulous optimization, we have identified the optimal selection of parameters that enhance the decision-making capabilities of reinforcement learning agents, particularly in navigating the exploration-exploitation trade-off. The insights gained from this analysis provide a solid foundation for fine-tuning these algorithms to achieve superior performance in diverse scenarios.

In conclusion, this thesis advances the theoretical understanding of LMDPs and their practical applications while providing a robust methodological framework for future research. The detailed embeddings and optimized parameter configurations presented herein are poised to contribute to the field of reinforcement learning, enabling more efficient and scalable solutions to complex decision-making problems. The implementations and findings of our work are publicly available, fostering continued research and development in this area.

5.1 Future Work

There are several areas for future research to further enhance the applicability and performance of Z-learning and related algorithms.

1. **Scalability to Larger Domains**: Future research should explore the scalability of Z-learning to even larger domains. This could involve optimizing the linear programming components and exploring parallel computation techniques to handle vast state spaces.

2. **Dynamic Environments**: Extending the current framework to dynamic environments where transition probabilities and reward functions change over time would be valuable. Developing adaptive algorithms that can quickly respond to environmental changes and maintain optimal performance will be crucial.

3. **Incorporating Deep Learning**: Integrating deep learning techniques with Z-learning could further improve scalability and performance. Using deep neural networks to approximate value functions and policy mappings could enable handling continuous state and action spaces.

Bibliography

[Bellman, 1957] Bellman, R. (1957). A Markovian Decision Process. *Indiana University Mathematics Journal*, 6(4):679–684.

[Bertsekas, 1987] Bertsekas, D. P. (1987). *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall, Englewood Cliffs, NJ.

[Bertsekas and Tsitsiklis, 1996] Bertsekas, D. P. and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*, volume 27. Athena Scientific.

[Gómez et al., 2014] Gómez, V., Kappen, H. J., Peters, J., and Neumann, G. (2014). Policy search for path integral control. In Calders, T., Esposito, F., Hüllermeier, E., and Meo, R., editors, *Machine Learning and Knowledge Discovery in Databases*, pages 482–497, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Gómez et al., 2020] Gómez, V., Thijssen, S., Symington, A., Hailes, S., and Kappen, H. J. (2020). Real-time stochastic optimal control for multi-agent quadrotor systems.

[Hassabis, 2016] Hassabis, D. (2016). From not working to neural networking. *The Economist*.

[Infante et al., 2022] Infante, G., Jonsson, A., and Gómez, V. (2022). Globally optimal hierarchical reinforcement learning for linearly-solvable markov decision processes. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(6):6970–6977.

[Jaakkola et al., 1994] Jaakkola, T., Jordan, M. I., and Singh, S. P. (1994). On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6:1185–1201.

[Jonsson and Gómez, 2016] Jonsson, A. and Gómez, V. (2016). Hierarchical linearly-solvable markov decision problems. In *Proceedings of the 26th International Conference on Automated Planning and Scheduling (ICAPS)*.

[Kappen, 2005a] Kappen, H. J. (2005a). Linear theory for control of nonlinear stochastic systems. *Phys. Rev. Lett.*, 95:200201.

[Kappen, 2005b] Kappen, H. J. (2005b). Linear theory for control of nonlinear stochastic systems. *Phys. Rev. Lett.*, 95:200201.

[Kappen et al., 2012] Kappen, H. J., Gómez, V., and Opper, M. (2012). Optimal control as a graphical model inference problem. *Machine Learning*, 87(2):159–182.

[Levine, 2018] Levine, S. (2018). Reinforcement learning and control as probabilistic inference: Tutorial and review.

[Molina et al., 2023] Molina, G. I., Jonsson, A., and Gómez, V. (2023). Optimal hierarchical average-reward linearly-solvable markov decision processes. In *Sixteenth European Workshop on Reinforcement Learning*.

[Puterman, 1994] Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, USA, 1st edition.

[Rawlik et al., 2012] Rawlik, K., Toussaint, M., and Vijayakumar, S. (2012). On stochastic optimal control and reinforcement learning by approximate inference. In *Proceedings of Robotics: Science and Systems*, Sydney, Australia.

[Rummery et al., 1994] Rummery, G., Niranjan, M., and of Cambridge. Engineering Department, U. (1994). *On-line Q-learning Using Connectionist Systems*. CUED/F-INFENG/TR. University of Cambridge, Department of Engineering.

[Sutton, 1991] Sutton, R. S. (1991). Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4):160–163.

[Sutton and Barto, 2018] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: an introduction*. Adaptive computation and machine learning series. The MIT Press, Cambridge, Massachusetts, second edition edition.

[Todorov, 2006] Todorov, E. (2006). Linearly-solvable markov decision problems. In Schölkopf, B., Platt, J., and Hoffman, T., editors, *Advances in Neural Information Processing Systems*, volume 19. MIT Press.

[Todorov, 2009] Todorov, E. (2009). Efficient computation of optimal actions. *Proceedings of the National Academy of Sciences of the United States of America*, 106:11478–83.

[Tsitsiklis, 1994] Tsitsiklis, J. N. (1994). Asynchronous stochastic approximation and q-learning. *Machine Learning*, 16(3):185–202.

[Watkins, 1989] Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis, University of Cambridge.

[Watkins and Dayan, 1992] Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3-4):279–292.

[Williams et al., 2017] Williams, G., Aldrich, A., and Theodorou, E. (2017). Model predictive path integral control: From theory to parallel computation. *Journal of Guidance, Control, and Dynamics*, 40:1–14.

Appendix A

SUPPLEMENTARY BACKGROUND AND THEORY

A.1 Dynamic Programming Methods

Dynamic programming methods are fundamental techniques for solving MDPs by leveraging the Bellman equations. These methods include policy evaluation, policy improvement, policy iteration, and value iteration.

A.1.1 Policy Evaluation

Policy evaluation involves computing the state-value function v_π for a given policy π. The goal is to determine the expected return for each state under the current policy. This is achieved by iteratively applying the Bellman expectation equation until convergence:

$$v_\pi^{(k+1)}(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s) \left[\mathcal{R}(s,a) + \gamma \sum_{s' \in S} \mathcal{P}(s'|s,a) v_\pi^{(k)}(s') \right] \tag{A.1}$$

The iterative process ends when the value function stabilizes, indicating that the estimates have converged to the true values under policy π.

A.1.2 Policy Improvement

Policy improvement is the process of refining a policy based on the value function. Given the state-value function v_π, a new policy π' is derived by choosing actions that maximize the expected return:

$$\pi^{(k+1)}(s) \leftarrow \arg\max_{a \in \mathcal{A}} \left(\mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) v_{\pi^{(k)}}(s') \right) \tag{A.2}$$

This process ensures that the new policy π' is at least as good as the original policy π.

A.1.3 Policy Iteration

Policy iteration combines policy evaluation and policy improvement in an iterative loop. Starting with an initial policy $\pi^{(0)}$, the algorithm alternates between evaluating the current policy and improving it:

- **Policy Evaluation**: Compute $v_{\pi^{(k)}}^{(k+1)}$.
- **Policy Improvement**: Update $\pi^{(k)}$ to $\pi^{(k+1)}$.

This loop continues until the policy stabilizes, indicating that the optimal policy has been found.

Appendix B

OPTIMIZATION OF ALGORITHMIC EFFICIENCY

B.1 Enhanced Efficiency in Value Iteration

Our value iteration implementation, described in Algorithm (8), leverages matrix operations and vectorized computation to significantly outperform traditional loop-based implementations, as presented in Algorithm (1). The efficiency of these two approaches is compared in Table (B.1), which highlights the dramatic reduction in execution times achieved through vectorization. The vectorized approach is particularly advantageous for large-scale sequential decision problems, where computational efficiency and scalability are crucial. By replacing iterative loops with matrix operations, our implementation takes full advantage of modern computational architectures optimized for parallel processing and efficient memory usage. This results in a performance improvement of nearly two orders of magnitude over the traditional loop-based method, demonstrating the scalability of our approach. For example, at a grid size of 50x50, the loop-based method requires approximately 100 seconds, whereas the vectorized method completes the task in just over 1 second, showcasing nearly a 100-fold increase in efficiency.

Table (B.1) clearly demonstrates that the vectorized implementation outperforms the loop-based approach across all tested grid sizes. As the grid size increases, the execution time of the loop-based method grows exponentially, while the vectorized method remains significantly faster. This substantial reduction in computation time not only accelerates the solution process but also enables handling more complex and larger environments, making the vectorized approach highly suitable for large-scale MDPs and LMDPs.

Grid Size	Number of States	Loop Time (s)	Vectorized Time (s)
2	4	0.00042	0.00007
3	9	0.00303	0.00194
5	25	0.01124	0.00034
10	100	0.11045	0.00123
15	225	0.49348	0.00433
20	400	1.56553	0.01392
30	900	9.18308	0.09197
40	1600	35.22223	0.37331
50	2500	99.86756	1.07719
60	3600	262.92955	2.99847
70	4900	505.89836	5.49093
80	6400	985.84623	10.65779
90	8100	1842.35975	19.35463
100	10000	3058.16183	33.91510

Table B.1: Value iteration execution times for different grid sizes using vectorized and loop approaches.

B.2 Efficiency Improvements in Power Iteration

Although LMDPs are inherently more efficient than MDPs, they can still be computationally intensive, especially when iterating over transition probabilities or reward functions in large settings. However, the sparse nature of the transition probability matrices allows for the application of efficient computation techniques, significantly reducing the computational cost, as demonstrated in Table (B.2). Both methods leverage matrix operations instead of loops, but the more efficient approach capitalizes on the sparse matrix condition, resulting in considerable performance gains compared to treating the matrices as dense. As shown in the table, for smaller environments, the non-sparse approach is faster due to the overhead associated with sparse matrix creation and manipulation, though this overhead is minimal. Simple grids of 15x15 or smaller, involving fewer than 300 states, can be efficiently computed without sparse matrices. In these cases, the overhead introduced by sparse matrices slower the computation while the setting is not big enough to leverage these efficient operations. This overhead is essentially negligible. For larger settings, the sparse matrix approach becomes significantly more efficient. In extreme cases, the sparse technique is up to 2000% faster, highlighting its importance. Without leveraging sparse matrices, computing the optimal value function for an LMDP with 6400 states would take over 25 minutes, rendering it impractical for benchmarking multiple algorithms. In contrast, with sparse

matrix optimization, this computation is reduced to a a second. The difference becomes even more pronounced in larger settings, where computing the optimal value function for a 10,000 state environment could take over 2 hours without sparse matrices, but only 2 seconds with them.

Grid Size	Number of States	Non-Sparse Time (s)	Sparse Time (s)
2	4	0.00439	0.00661
3	9	0.00079	0.00727
5	25	0.00206	0.00561
10	100	0.00660	0.01041
15	225	0.02661	0.01544
20	400	0.21025	0.02998
30	900	2.03092	0.05614
40	1600	13.63155	0.11217
50	2500	63.88911	0.22688
60	3600	218.97058	0.36187
70	4900	660.02274	0.62532
80	6400	1514.062	1.00146
90	8100	3533.87411	1.45586
100	10000	8723.56723	2.14425

Table B.2: Power iteration execution times for different grid sizes using sparse and non-sparse techniques.

Additionally, it is important to note that power iteration is used iteratively within some algorithms. For instance, in the embedding of deterministic MDPs, power iteration is called twice during each iteration of the ternary search to compare value function approximations. This iterative process can involve tens or hundreds of function calls, which would be infeasible without these efficiency improvements. Such computations could take hours or even days for a medium to large MDP setting, but with the optimized approach, they can be completed in just a few seconds. The results presented in Figure (3.3) would have been unattainable without these efficiency improvements. It is also important to not how the required times for power iteration, in the efficient approach, smaller than in the value iteration for MDPs. This reinforces the main point of the LMDPs allow for more efficient computation of value functions by leveraging linear programming. In small settings it may not be differential, but in larger cases power iteration is up to 10 times faster than value iteration, showcasing how linear Bellman Equations combined with matrix optimization techniques are quite more efficient than traditional Bellman Equations.

Appendix C

EXPLORATION-EXPLOTIATION OPTIMIZATION

As discussed throughout this work, the exploration-exploitation dilemma remains one of the most significant challenges in reinforcement learning. This challenge is partially addressed by the LMDP framework, which incorporates stochastic uncontrolled transition probabilities and ensures that even optimal controlled probabilities include transitions to non-optimal states, thus maintaining exploration. However, for MDPs, this challenge persists. In this section, we provide an in-depth analysis of our experiments and findings regarding this issue.

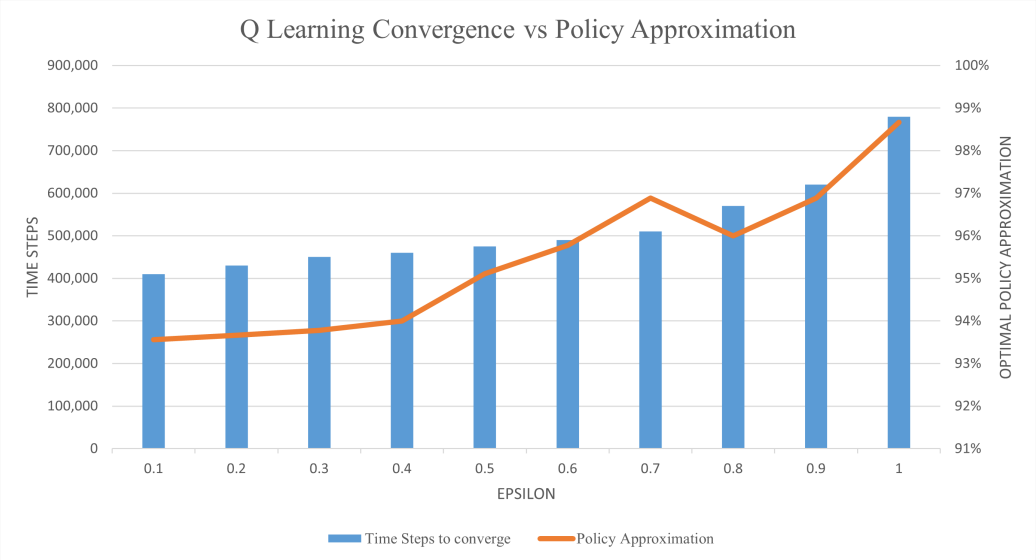

Figure C.1: Q-learning Convergence and Policy Approximation by epsilon

Figure (C.1) illustrates the convergence times and policy approximation errors for different values of ϵ with an ϵ-decay of 0.999. As observed in the main experimentation section, increased exploration leads to better approximations but slower convergence. Specifically, an initial ϵ of 1 requires twice the convergence time compared to an initial ϵ of 0.1, but it achieves a near-optimal policy. This work aims to find efficient methods for proper estimation, balancing convergence speed and optimal policy approximation. Our analysis suggests that using an initial ϵ of 1 with a slightly reduced ϵ-decay yields good results in terms of approximation error and convergence speed.

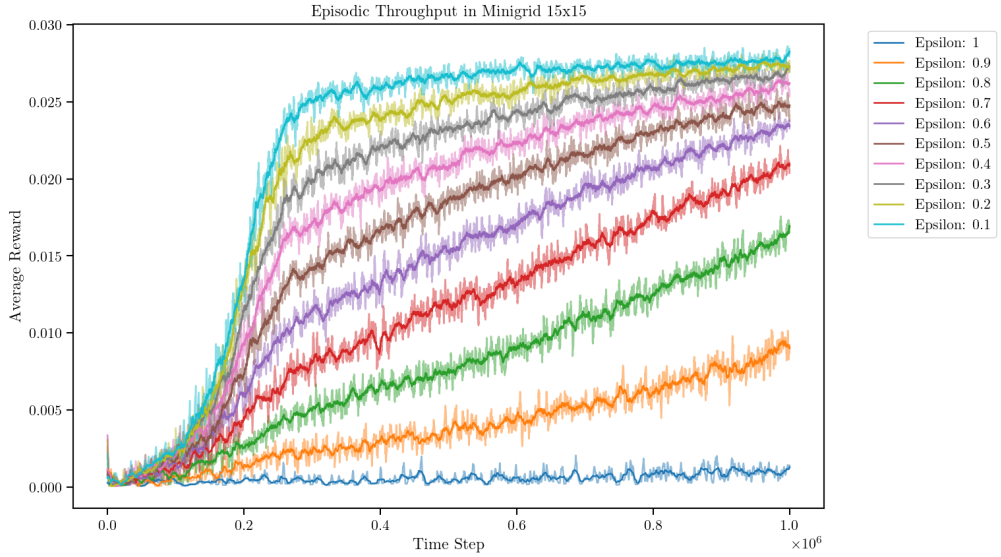

Figure C.2: Q-learning Convergence by ϵ

When fixing exploration, as in ϵ-greedy strategies with an exploration decay set to 1, the smallest ϵ values result in the fastest convergence, as shown in Figure (C.2). Here, Q-learning converges fastest with $\epsilon = 0.1$. However, despite the quick convergence, this approach yields the least precise approximation. It still does not match the convergence speed and precision of Z-learning, which outperforms Q-learning in value function approximation, highlighting the advantages of LMDPs over MDPs.

Figure (C.3) illustrates the convergence behavior of Q-learning with various ϵ-decay values, starting with an initial ϵ of 1, in a 10x10 grid world containing multiple randomly placed obstacles. The results reaffirm our earlier observations: strategies that enforce prolonged exploration yield more accurate approximations but slower convergence rates. Conversely, approaches that prioritize brief exploration achieve faster convergence but result in less precise approximations.

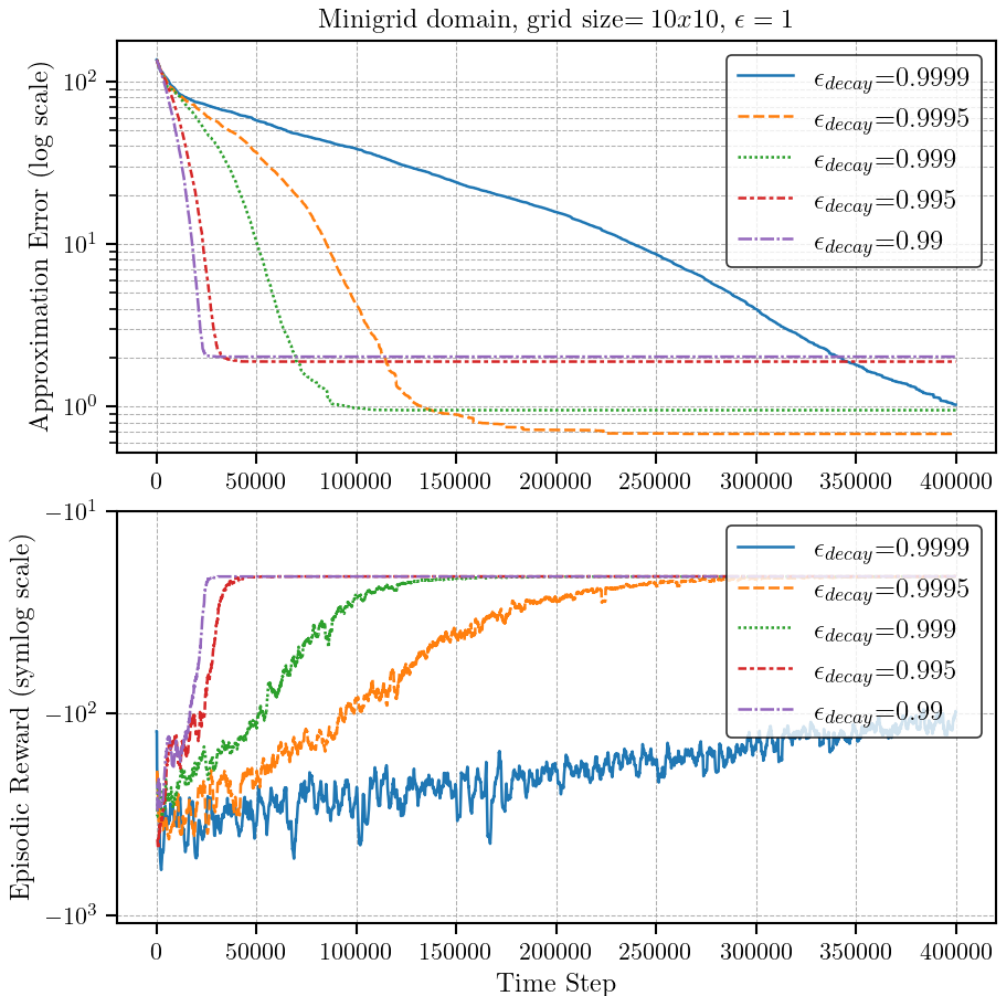

Figure C.3: Comparison of approximation error (top) and episodic reward (bottom) for different ϵ-decay values in a 10x10 grid world.

The following figures present a comparison of different exploration strategies within larger environments. Consistent with previous observations, higher ϵ-decay values continue to struggle with convergence, and these difficulties become more pronounced as the state space expands. Additionally, the overall differences in convergence rates between the various exploration strategies diminish as the problem size increases. This trend suggests that while larger ϵ-decay values provide better approximations, their slower convergence becomes a significant challenge in more extensive state spaces. Conversely, lower ϵ-decay values maintain their faster convergence but at the cost of approximation accuracy.

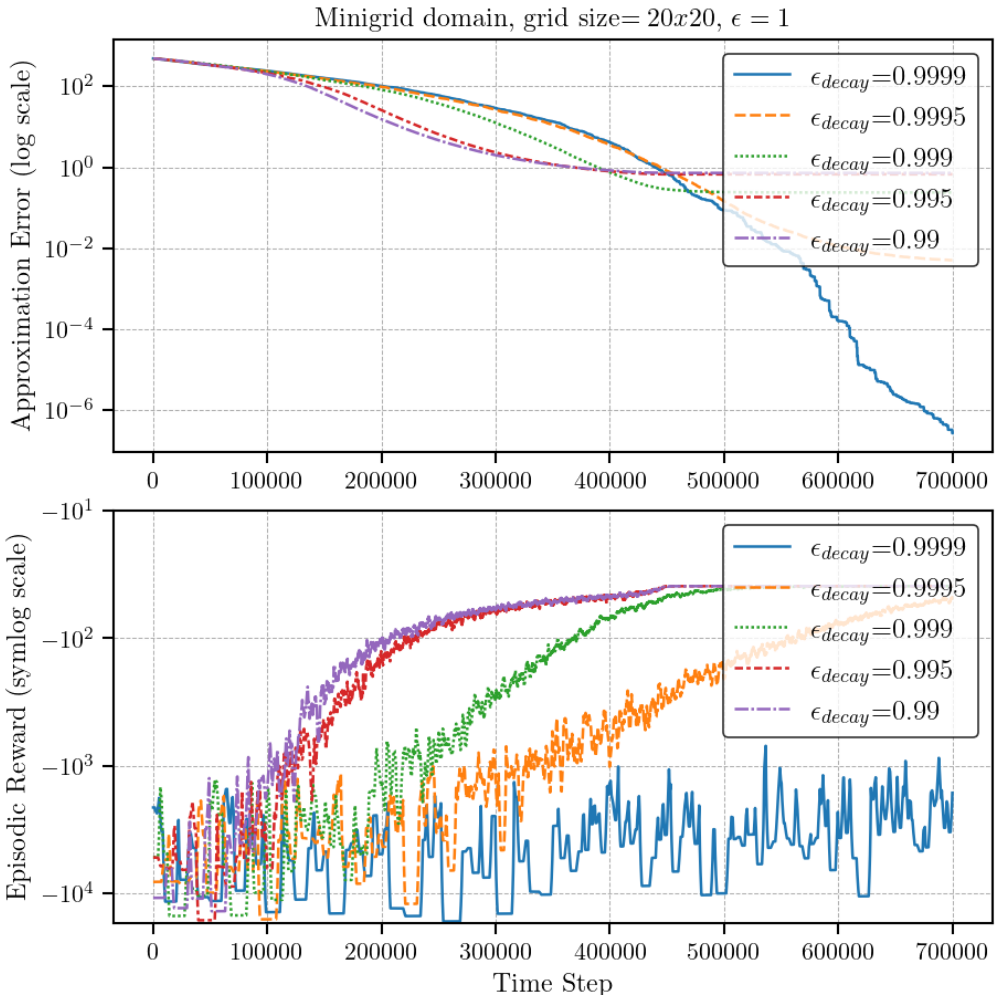

Figure C.4: Comparison of approximation error (top) and episodic reward (bottom) for different ϵ-decay values in a 20x20 grid world.

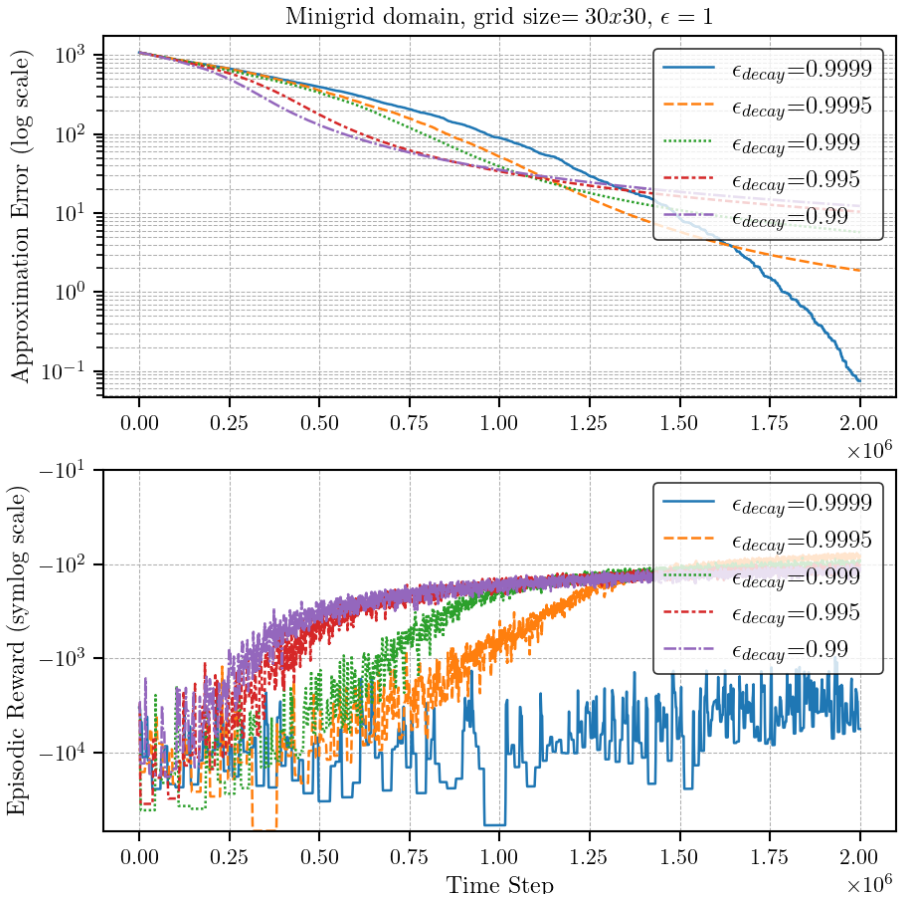

Figure C.5: Comparison of approximation error (top) and episodic reward (bottom) for different ϵ-decay values in a 30x30 grid world.

The next figures illustrate the convergence rates and approximation errors for the various exploration approaches across the diverse settings examined in this study. Additionally, they showcase the value function and policy approximations for each method in each setting, providing a comprehensive comparison of their performance. It is evident across a wide range of diverse settings that the same trend persists: high ϵ-decay values yield accurate approximations, whereas low ϵ-decay values result in faster convergence. Interestingly, in maze-like environments, ϵ-decay values that previously balanced convergence rate and approximation precision now require excessive time to converge. In these environments, only the fastest strategies, which employ brief exploration phases, achieve reasonable convergence. This is primarily because, in maze-like environments, enforcing exploration beyond the initial learning phase significantly hampers the on-line optimal convergence. Achieving the goal through an optimal path in such environments is highly dependent on minimal exploration.

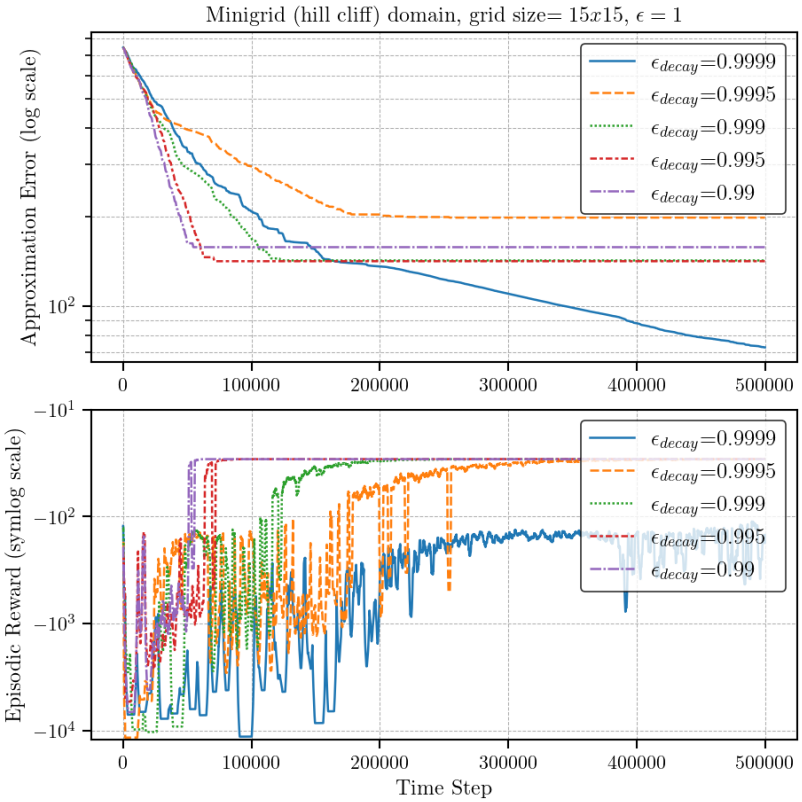

Figure C.6: Comparison of approximation error (top) and episodic reward (bottom) for different ϵ-decay values in a Minigrid hill-cliff domain.

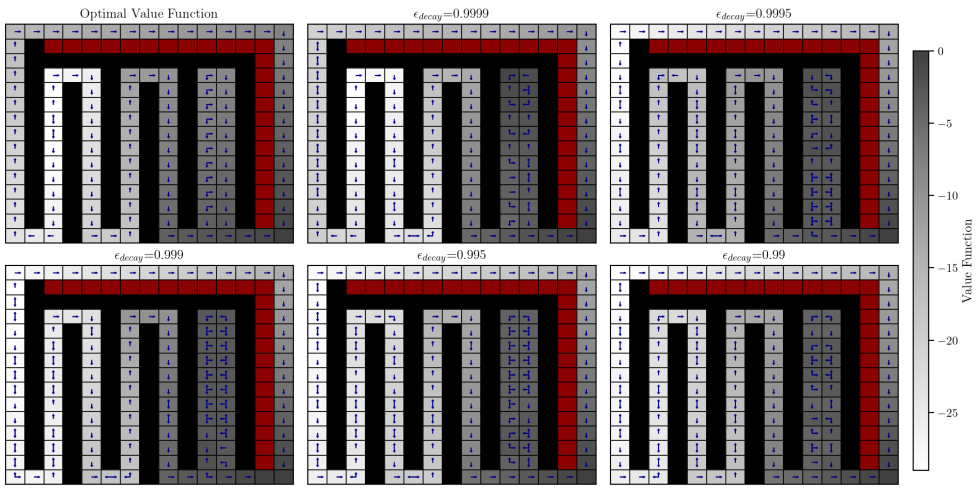

Figure C.7: Value function and policy approximations for different ϵ-decay values in a Minigrid hill-cliff domain.

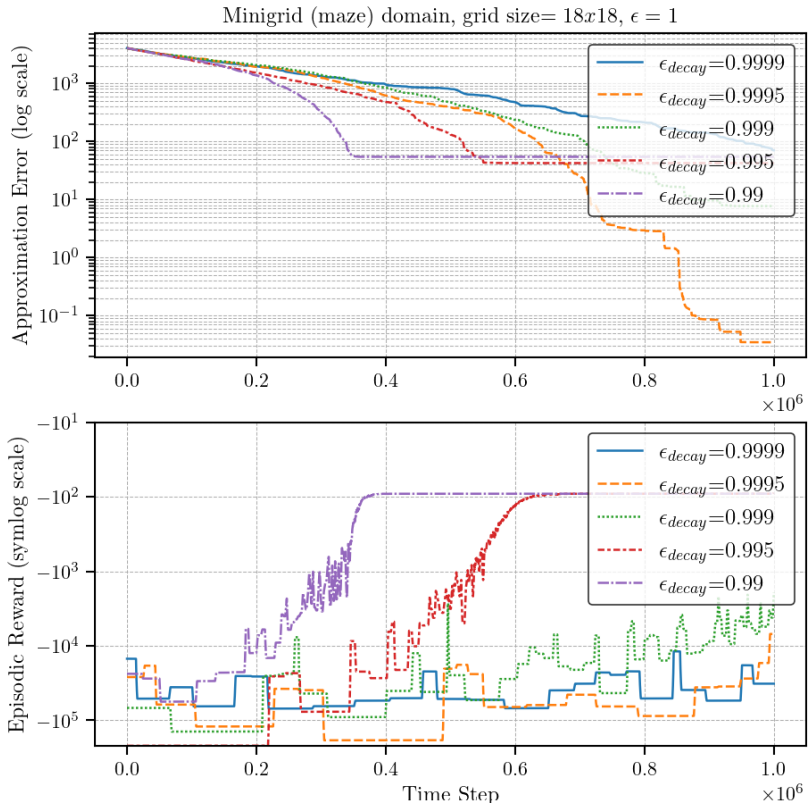

Figure C.8: Comparison of approximation error (top) and episodic reward (bottom) for different ϵ-decay values in a Minigrid 18x18 maze domain.

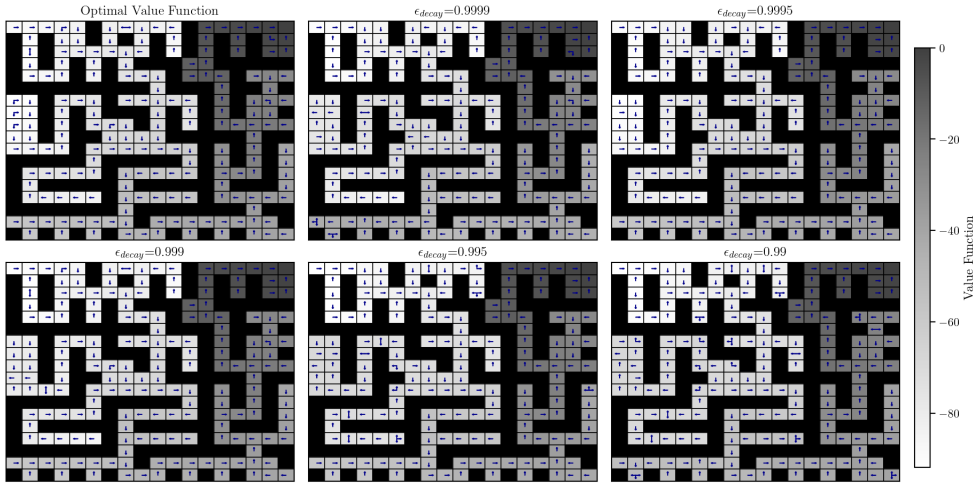

Figure C.9: Value function and policy approximations for different ϵ-decay values in a Minigrid 18x18 maze domain.

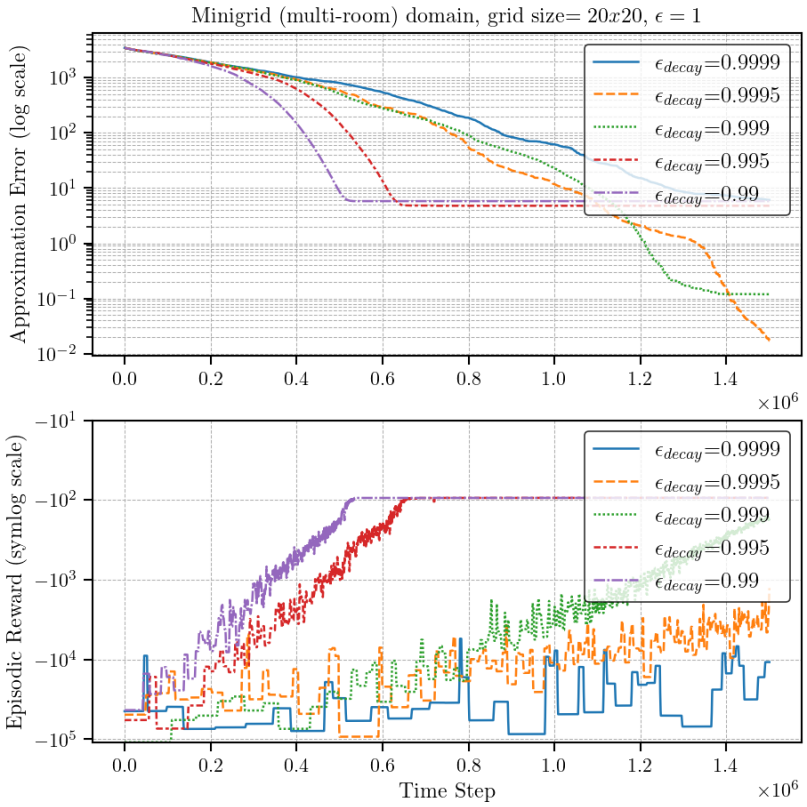

Figure C.10: Comparison of approximation error (top) and episodic reward (bottom) for different ϵ-decay values in a Minigrid 20x20 multi-room domain.

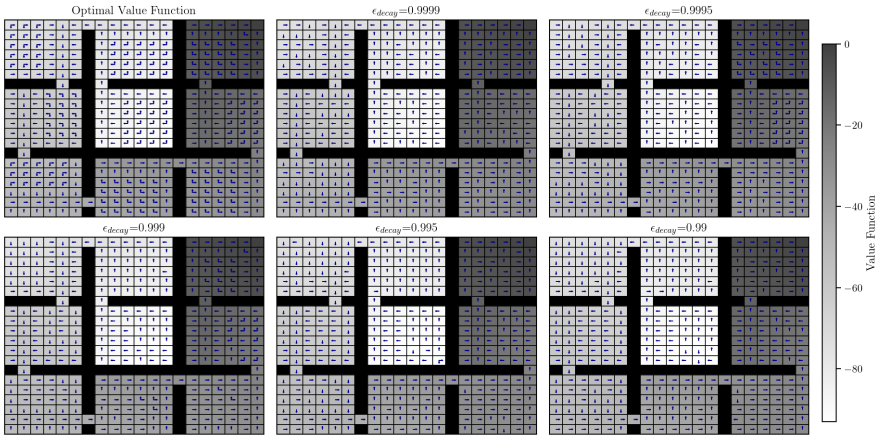

Figure C.11: Value function and policy approximations for different ϵ-decay values in a Minigrid 20x20 multi-room domain.

Appendix D

ALGORITHM HYPERPARAMETER OPTIMIZATION

Parameter tuning is a crucial aspect of reinforcement learning, as the performance of training routines can be significantly affected by slight modifications in the parameters. The selection of appropriate hyperparameters is essential for achieving optimal learning efficiency and effectiveness, and it can greatly influence the convergence rate and the accuracy of the resulting policy. In this work, we conducted an exhaustive parameter tuning for the Q-learning and Z-learning algorithms, ensuring that both methods were finely tuned to perform at their best across various environments.

While previous chapters have explored the exploration-exploitation parameters, specifically focusing on the ϵ-decay strategies, this chapter examines other key hyperparameters essential for enhancing reinforcement learning algorithms. The learning rate is crucial in determining how much new information is integrated versus retained. The temperature parameter in Z-learning, another critical hyperparameter, controls policy stochasticity and significantly impacts performance.

Our parameter tuning involved systematic experimentation and analysis. By evaluating the effects of different hyperparameter values on Q-learning and Z-learning, we identified the optimal settings that maximize learning performance. This analysis highlights the sensitivity of these algorithms to hyperparameter selection and provides insights into achieving the best results. The following sections present our findings on hyperparameter optimization, demonstrating the significant impact of fine-tuning on efficiency and scalability.

D.1 Q-learning Optimization

D.1.1 Learning Rate

The learning rate is a fundamental parameter in any machine learning model, particularly in reinforcement learning. It determines how much the agent learns from the current observation versus how much it retains from previous observations. Following the approach of [Todorov, 2006], we used a constant c to dynamically update the learning rate and conducted independent studies to observe the agent's learning behavior for different values of c in Q-learning and Z-learning.

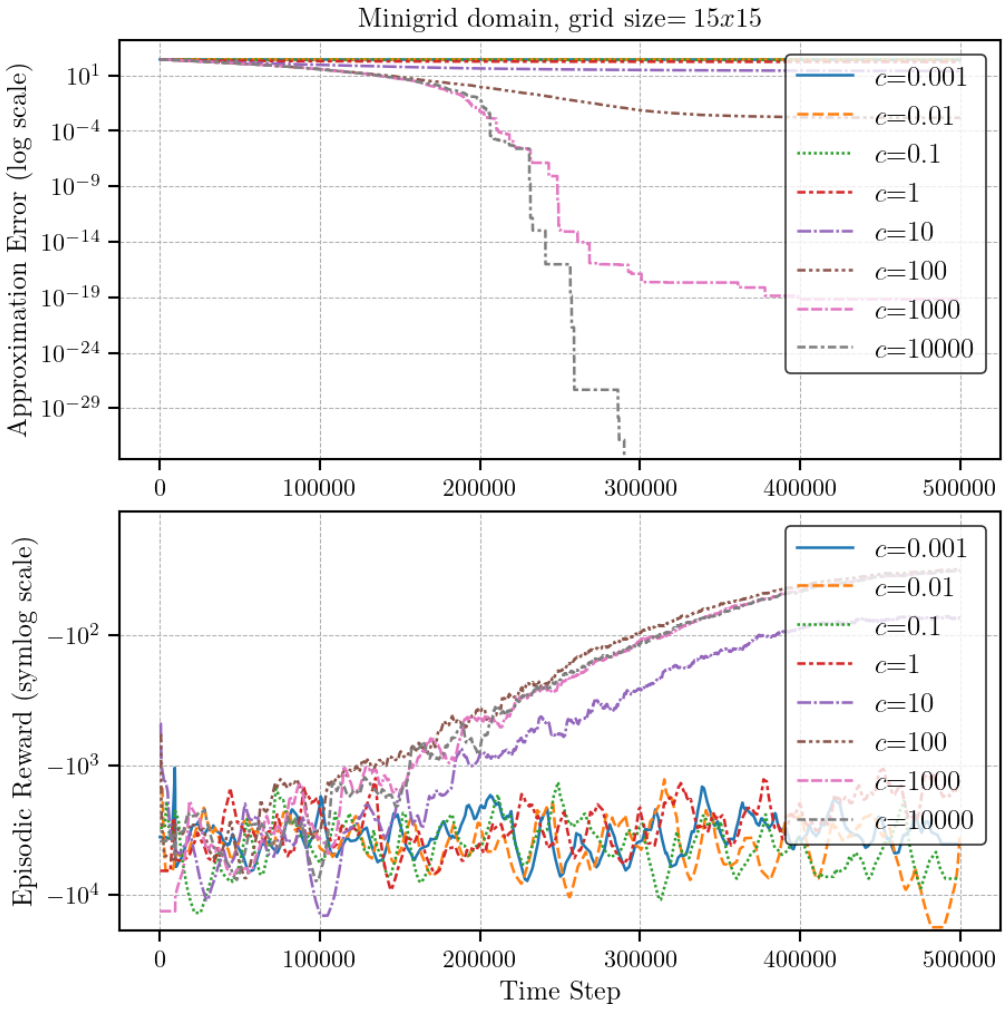

Figure D.1: Comparison of approximation error (top) and episodic reward (bottom) for different c values in a 15x15 empty grid world.

Figure (D.1) provides an initial comparison of a broad range of c values in an empty 15x15 grid. The upper plot illustrates the approximation error, represented as the mean squared error between the optimal value function and the approximations obtained from each approach at every time step. The lower plot depicts the episodic reward for each approach. The results indicate that only values of $c \geq 10$ achieve convergence to the maximum throughput, whereas decimal values show a lack of effective learning.

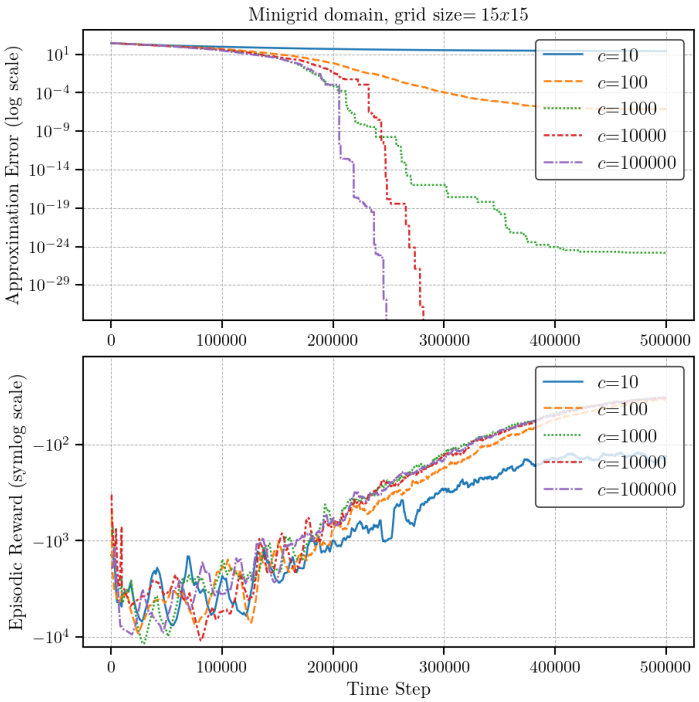

Figure D.2: Comparison of approximation error (top) and episodic reward (bottom) for different c values in a 15x15 empty grid world.

Figure (D.2) presents a comparison of non-decimal multiples of 10 for the parameter c in a 15x15 grid environment. It is evident that larger values of c lead to both superior approximations and faster convergence, while smaller values result in poorer approximations and require more time to converge. However, this trend does not hold universally across all settings, as these results are based on a simple empty grid. The subsequent figures illustrate the comparison of the best multiples of 10 for c in various settings, specifically within the different domains explored in this work. As shown in Appendix (C), each setting requires a unique optimal exploration strategy due to the problem's nature. Therefore, the optimal exploration strategy is employed for each setting, prioritizing convergence speed while ensuring an accurate value function approximation.

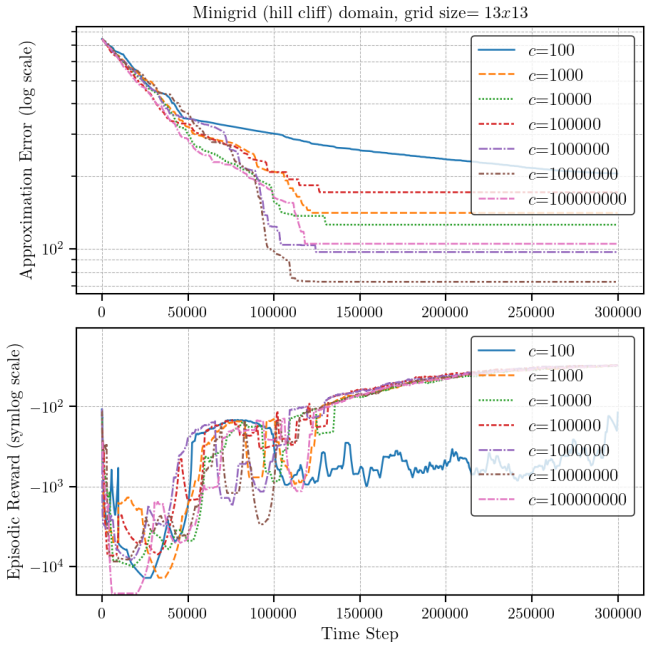

Figure D.3: Comparison of approximation error (top) and episodic reward (bottom) for different c values in a 13x13 hill-cliff emulation grid world.

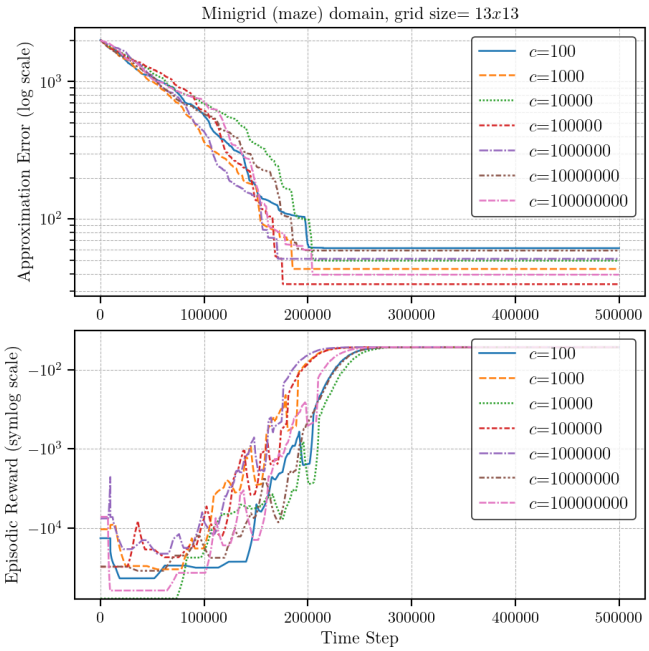

Figure D.4: Comparison of approximation error (top) and episodic reward (bottom) for different c values in a 13x13 maze-like grid world.

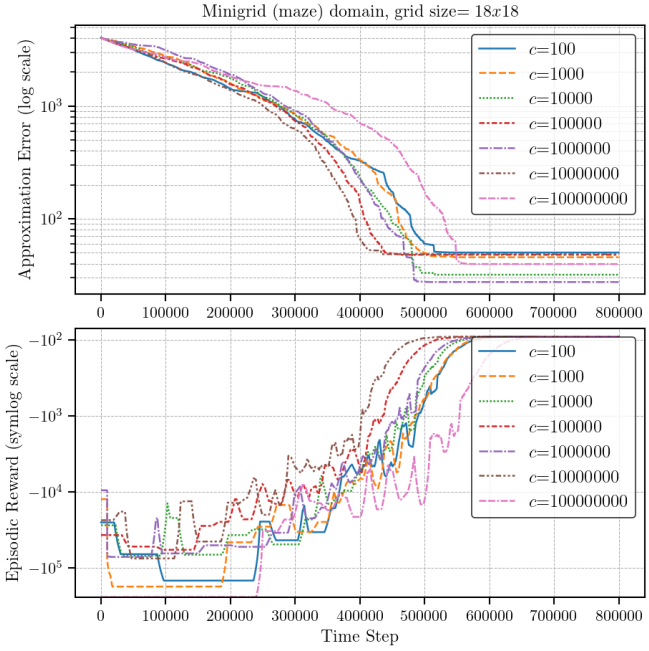

Figure D.5: Comparison of approximation error (top) and episodic reward (bottom) for different c values in a 18x18 maze-like grid world.

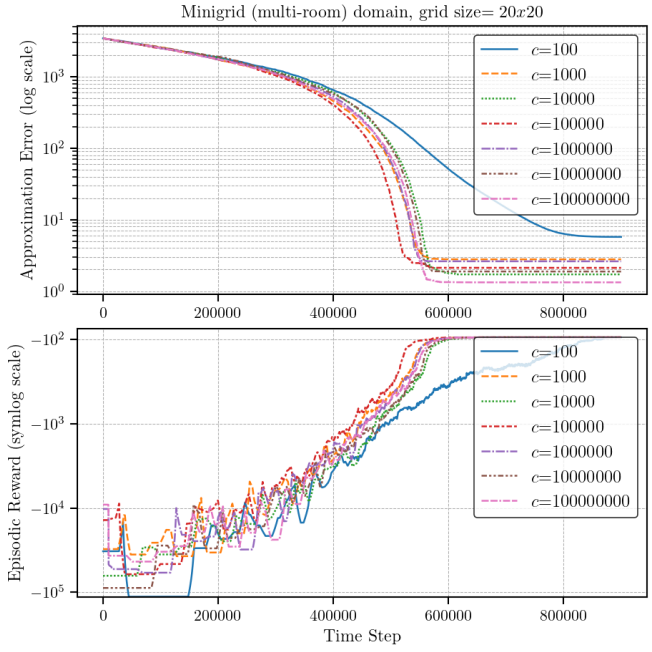

Figure D.6: Comparison of approximation error (top) and episodic reward (bottom) for different c values in a 18x18 multi-room grid world.

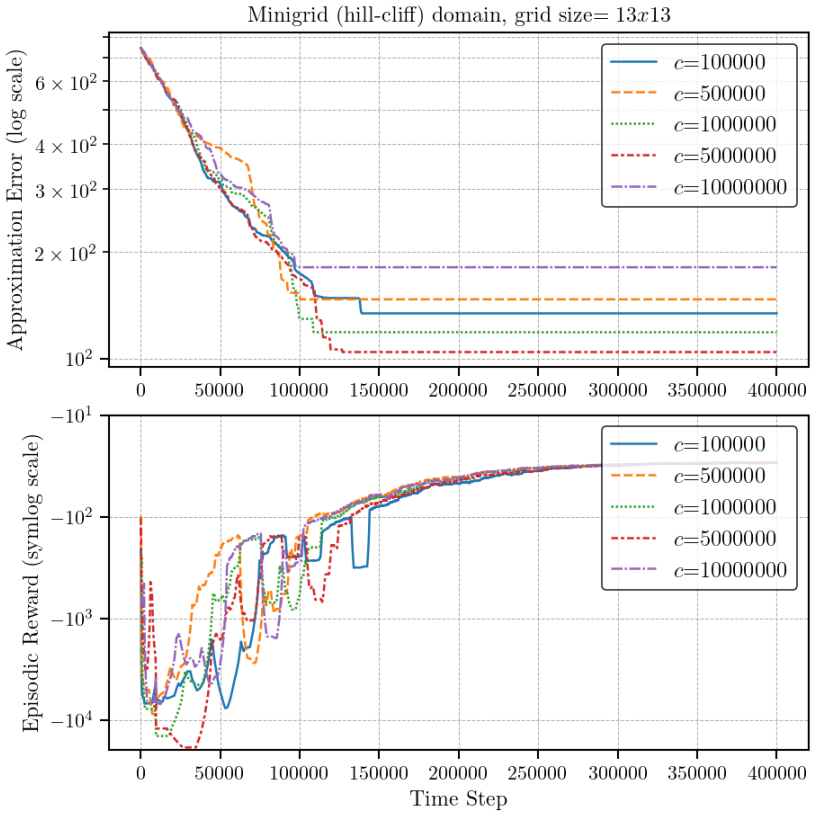

Figure D.7: Comparison of approximation error (top) and episodic reward (bottom) for different c values in a 13x13 hill-cliff emulation grid world.

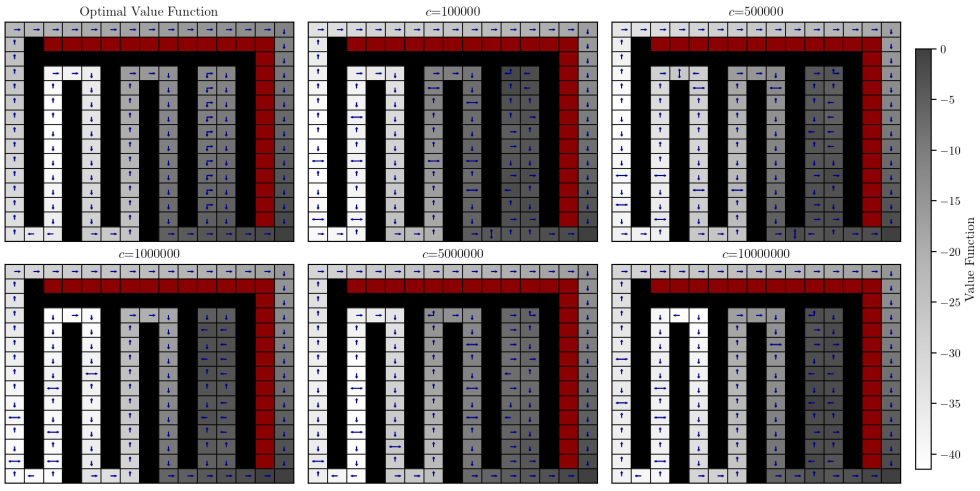

Figure D.8: Value function and policy approximations for different c values in a 13x13 hill-cliff emulation grid world.

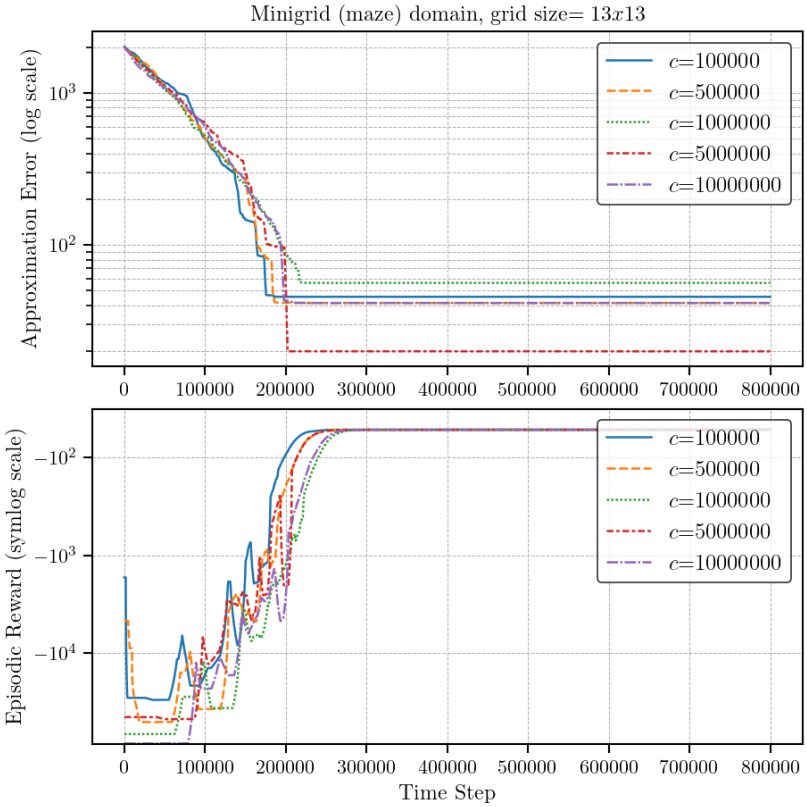

Figure D.9: Comparison of approximation error (top) and episodic reward (bottom) for different c values in a 13x13 maze-like grid world.

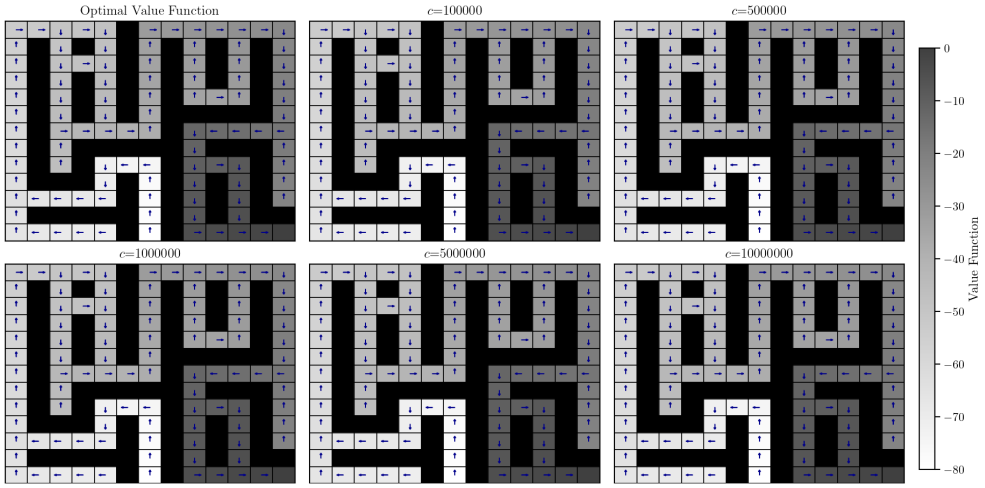

Figure D.10: Value function and policy approximations for different c values in a 13x13 maze-like grid world.

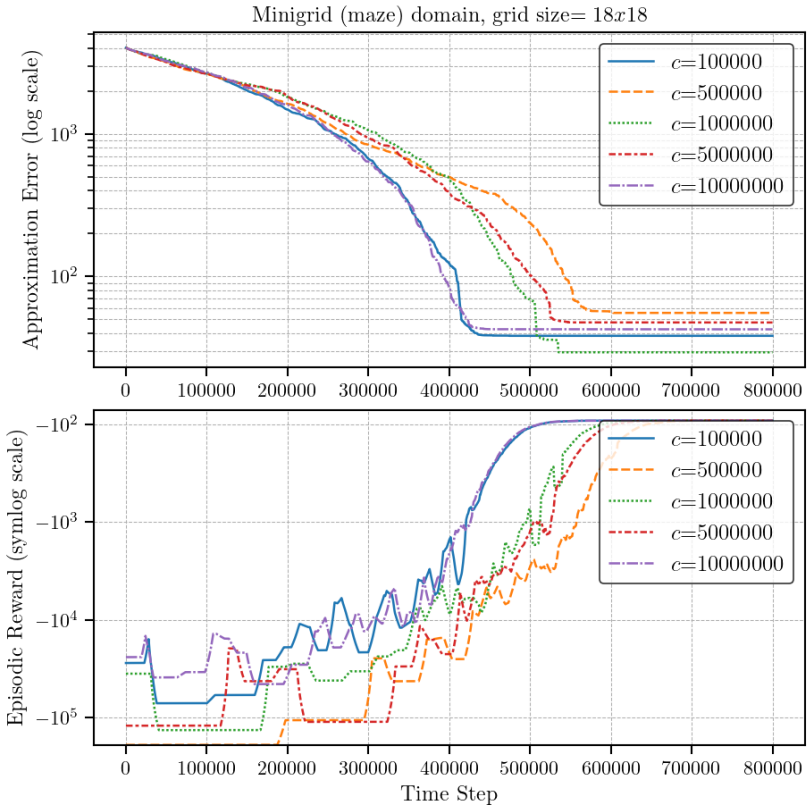

Figure D.11: Comparison of approximation error (top) and episodic reward (bottom) for different c values in a 18x18 maze-like grid world.

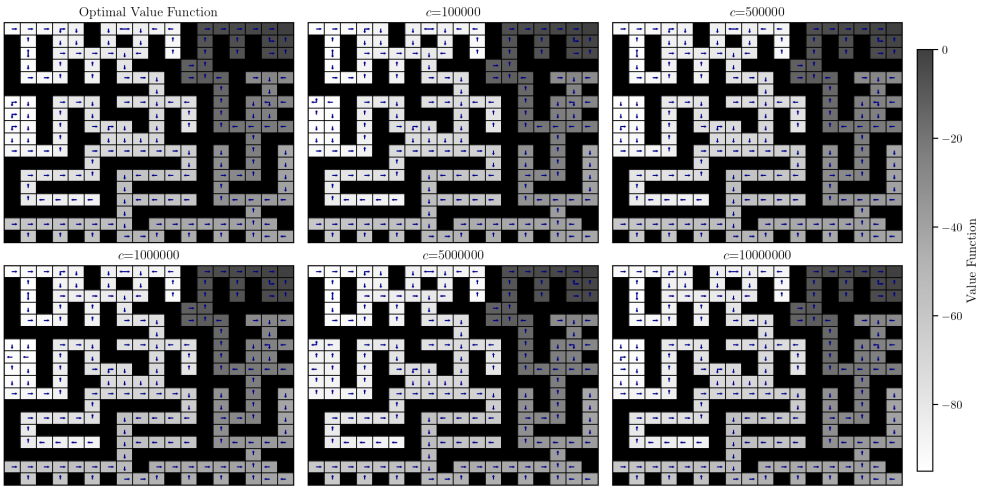

Figure D.12: Value function and policy approximations for different c values in a 18x18 maze-like grid world.

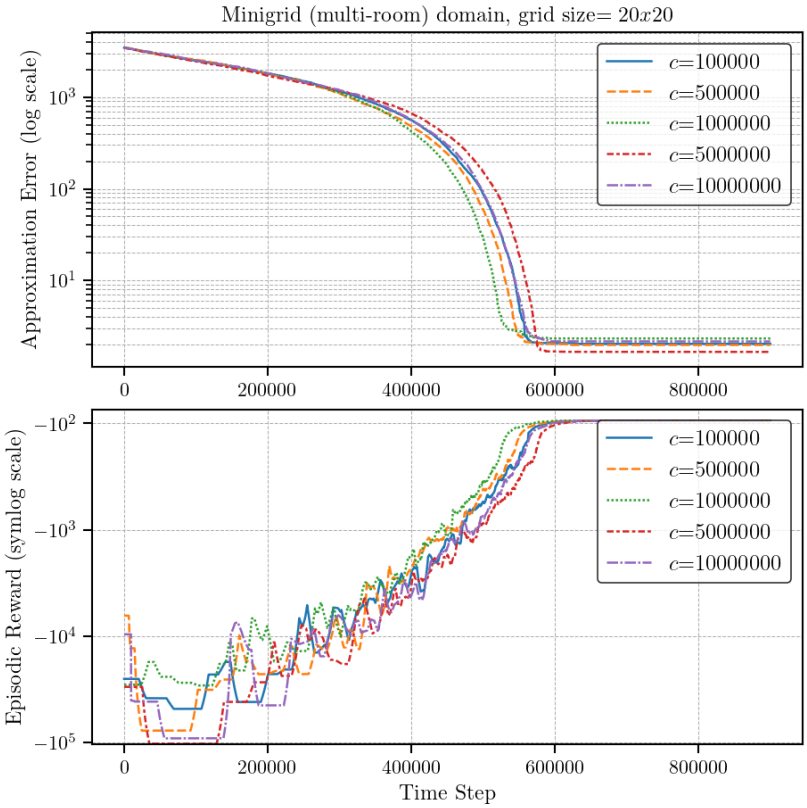

Figure D.13: Comparison of approximation error (top) and episodic reward (bottom) for different c values in a 18x18 multi-room grid world.

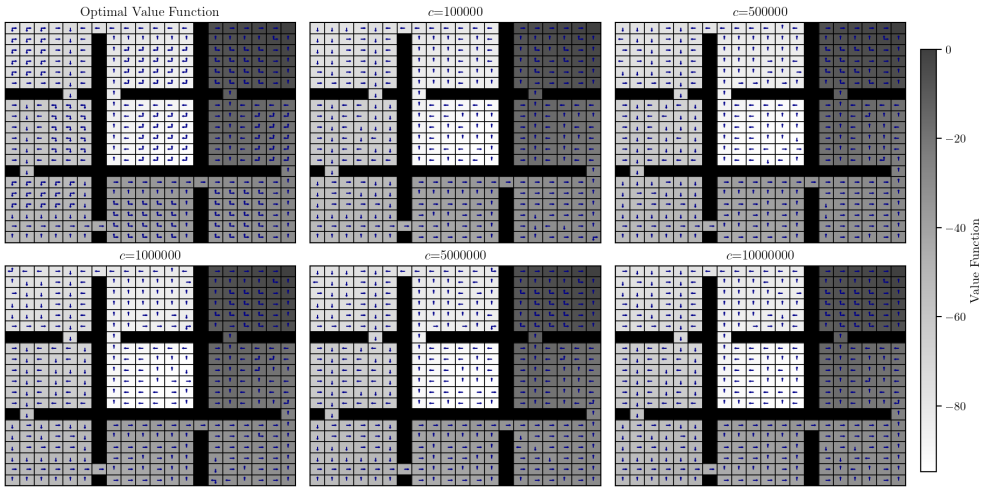

Figure D.14: Value function and policy approximations for different c values in a 18x18 multi-room grid world.

As illustrated in the previous images, the choice of the hyperparameter c significantly influences the behavior of the learning algorithms across different settings. In certain scenarios, the values of c that facilitate the fastest convergence may lead to less accurate approximations. Conversely, in other settings, this relationship does not necessarily hold. The optimal choice of c varies depending on the environment, which can be attributed to the impact of undesirable terminal states and obstacles on the value function approximation.

For instance, in maze-like environments, $c = 100,000$ performs reasonably well. However, in environments with undesirable terminal states, such as the hill-cliff emulation, values around $1,000,000$ offer the best trade-off between convergence rate and approximation accuracy. In settings without undesirable terminal states, but that are not maze-like, providing the agent with more freedom to reach the goal, a range of c values from $100,000$ to $5,000,000$ performs similarly. Therefore, for the comparative analysis between Q-learning and Z-learning, the optimal c value has been selected based on the specific characteristics of each domain to ensure a fair and accurate evaluation.

D.1.2 Discount Rate

The discount rate, denoted as γ, is a crucial parameter in reinforcement learning, as it governs the trade-off between immediate and long-term rewards. A higher discount rate emphasizes long-term rewards, encouraging the agent to prioritize future benefits over immediate gains, while a lower discount rate places more importance on immediate rewards. The choice of γ directly impacts the agent's decision-making process and the convergence of the learning algorithm. In essence, γ determines how much future rewards are valued in comparison to current rewards, thereby influencing the overall policy learned by the agent.

In typical reinforcement learning scenarios with positive rewards, tuning γ is essential for balancing exploration and exploitation, as well as for achieving a desirable trade-off between short-term and long-term gains. However, in our case, we are dealing with non-positive rewards, which fundamentally alters the dynamics of the problem. Non-positive rewards inherently incentivize the agent to minimize the episode length, as accumulating negative rewards leads to a lower overall return. Consequently, the optimal choice of γ in such scenarios is found to be 1 across all settings, regardless of the nature of the problem. This choice effectively removes the discounting effect, treating future rewards equally to immediate ones, which aligns with the objective of minimizing the negative reward accumulation over the course of an episode.

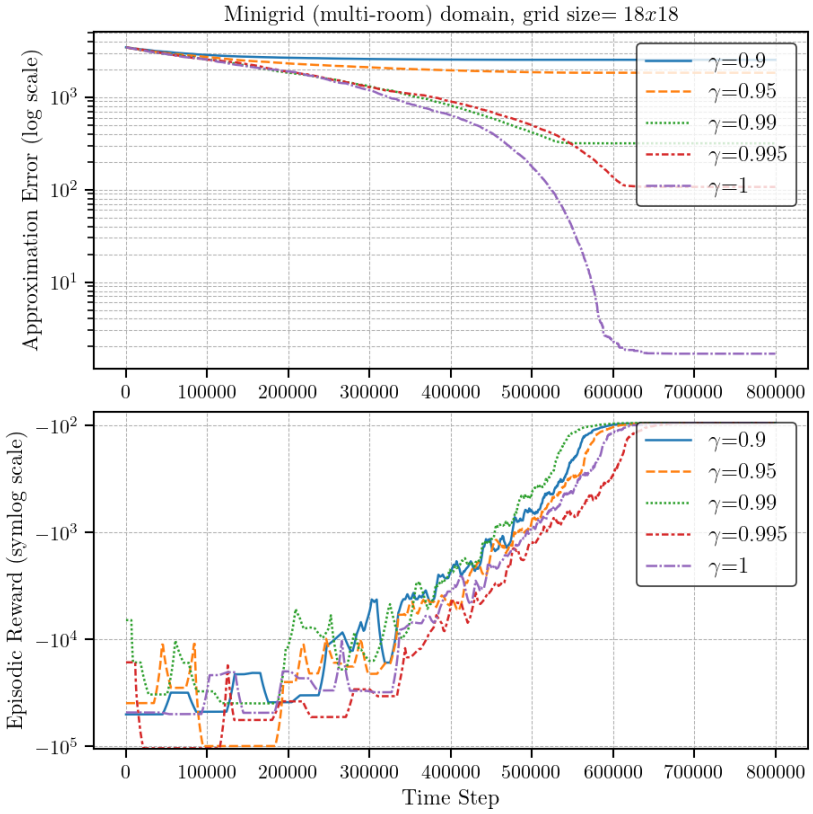

Figure D.15: Comparison of approximation error (top) and episodic reward (bottom) for different γ values in a 18x18 multi-room grid world.

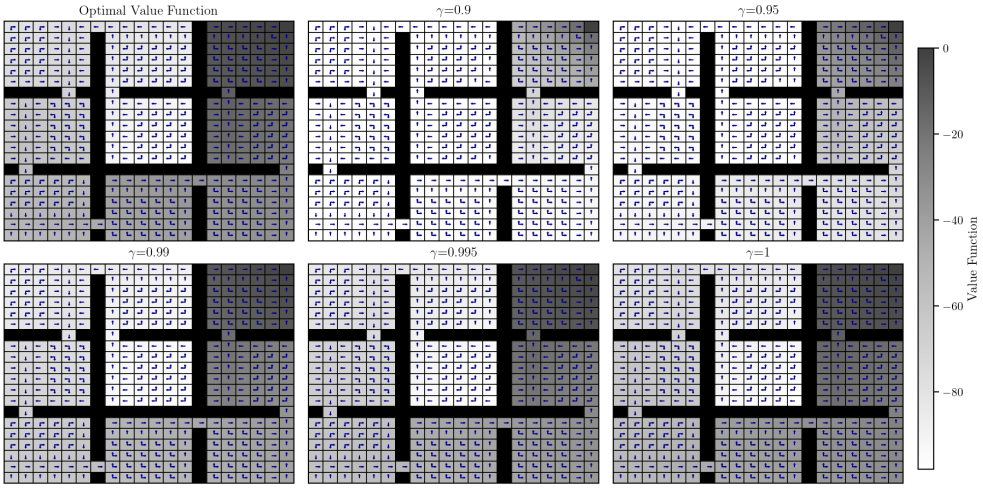

Figure D.16: Value function and policy approximations for different c values in a 18x18 multi-room grid world.

Figure (D.15) illustrates the comparison of various values of γ in a multi-room setting. It is evident that $\gamma = 1$ leads to significantly more accurate solutions, with the convergence rate remaining fairly consistent across different approaches. The disparity in approximation error is particularly pronounced when employing methods that emphasize exploration throughout the episode. In these methods, the value function approximations are notably less accurate compared to those using a discount rate of 1, as depicted in Figure (D.16).

Interestingly, this figure also shows that the approximations of the optimal transitions are nearly identical across different approaches. This discrepancy between the value function approximation and the similarity in optimally controlled transition probabilities arises because discounting reduces all value function values uniformly. Consequently, while the absolute accuracy of the value function is diminished with lower discount rates, the relative differences between states remain unchanged. This uniform scaling effect ensures that the policy derived from the value function remains optimal, even though the value function itself is less precise.

D.2 Z-learning Optimization

Optimizing Z-learning proved to be significantly more straightforward than optimizing Q-learning, primarily due to the inherent dynamics of Z-learning, which alleviate the exploration challenge. This simplification arises because Z-learning leverages the optimal controlled transition probabilities, thereby reducing the need for extensive exploration that characterizes traditional Q-learning approaches.

D.2.1 Learning Rate

For the learning rate optimization, we adhered to the methodology outlined in [Todorov, 2006]. Specifically, we optimized the parameter c independently for both Q-learning and Z-learning. Our goal was to identify the optimal values of c that would enhance the performance of Z-learning in terms of convergence speed and accuracy of the value function approximation.

During the optimization process, we observed notable disparities between Z-learning and Q-learning. Unlike Q-learning, where the learning rate has a pronounced impact on the balance between exploration and exploitation, Z-learning's learning dynamics are influenced differently by the choice of c. This is due to the formulation of Z-learning, which inherently integrates aspects of exploration through the optimal control framework, making the learning rate less sensitive to the pitfalls of under or over-exploration.

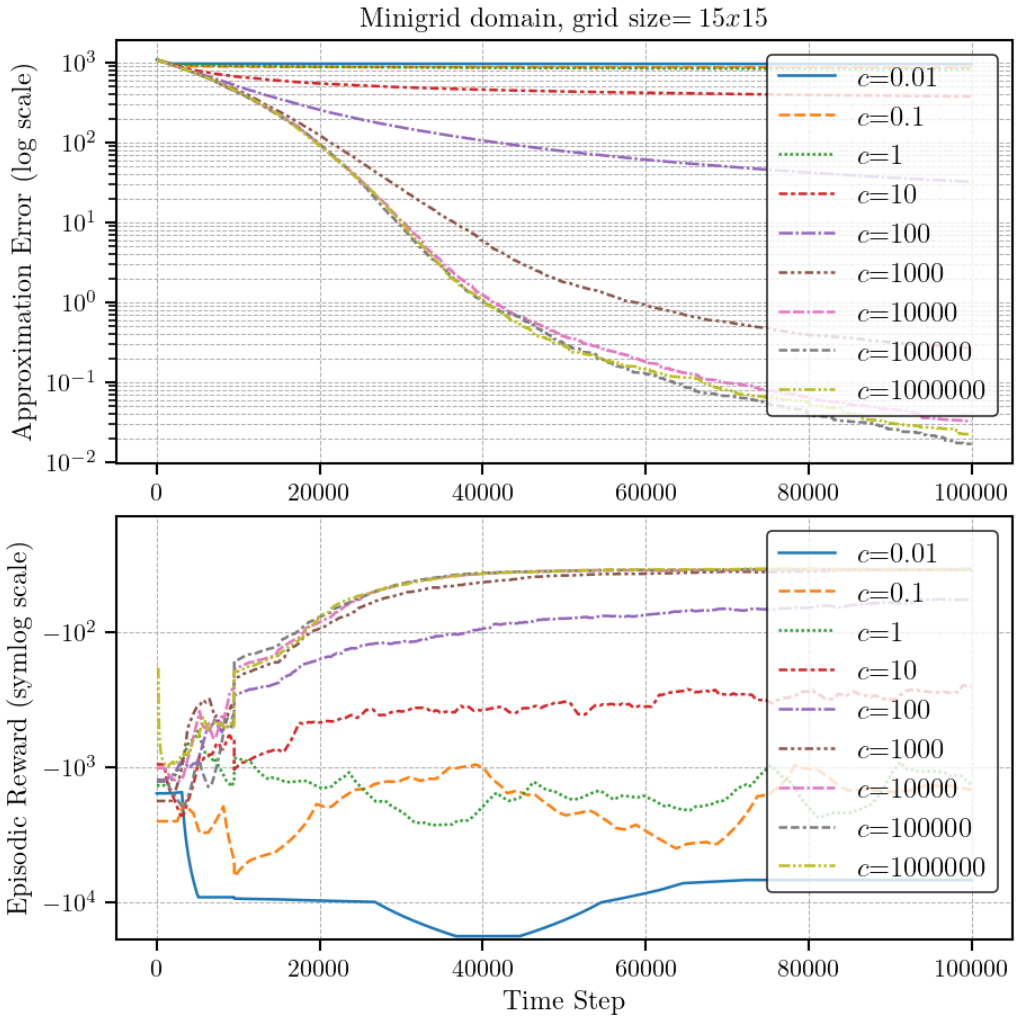

Figure D.17: Comparison of approximation error (top) and episodic reward (bottom) for different c values in a 15x15 empty grid world.

Figure (D.17) illustrates the comparison of Z-learning performance across multiples of 10 for the parameter c in an empty grid world with a grid size of 15. The results demonstrate a pattern similar to that observed in Q-learning. Specifically, decimal values fail to achieve convergence, while values exceeding $1,000$ yield the most promising outcomes. This indicates that higher values of c are more effective in facilitating accurate and efficient learning in this environment. As illustrated in the figure above, and further corroborated by Figure (D.18), which compares the best-performing multiples of 10 values for c in a 15x15 grid world environment with obstacles and non-desirable states, larger values do not necessarily result in better solutions as observed with Q-learning. Furthermore, there is no significant difference in performance across this wide range of c values.

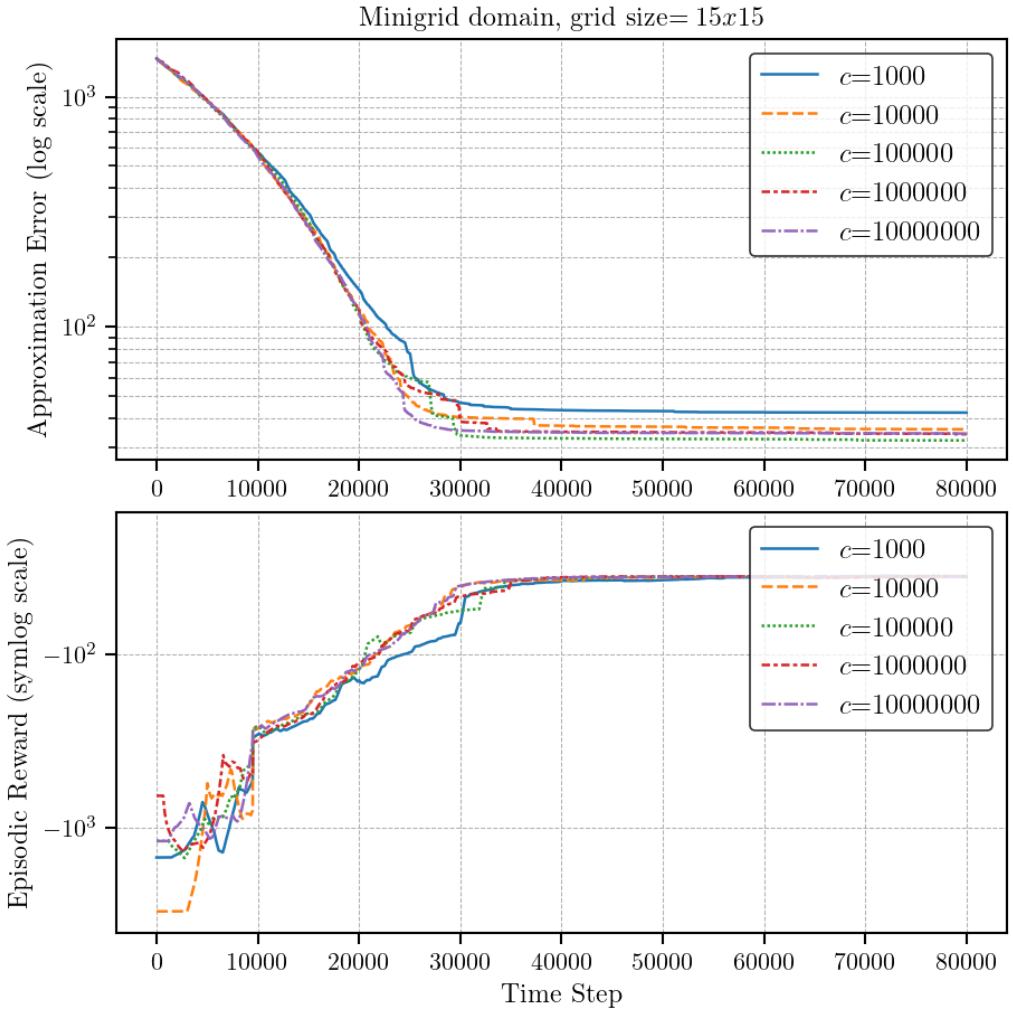

Figure D.18: Comparison of approximation error (top) and episodic reward (bottom) for different c values in a 15x15 non-empty grid world.

Despite the equivalent benchmarked performance between these values, this does not necessarily hold for more intricate settings. In the Hill-Cliff emulation domain, the values of c that appear to offer the best trade-off between convergence rate and approximation error are around $10,000$, as illustrated in Figure (D.19). Interestingly, this setting exhibits the fastest convergence among all the explored environments. This observation holds not only for Z-learning learning rate tuning but for the entire experimentation procedure. This phenomenon is mainly due to the nature of the problem, which has a very straightforward optimal solution that does not require extensive exploration. Consequently, the agent quickly tends to choose the optimal path rather than exploring. However, this often leads to a less accurate approximation of the suboptimal and safer paths.

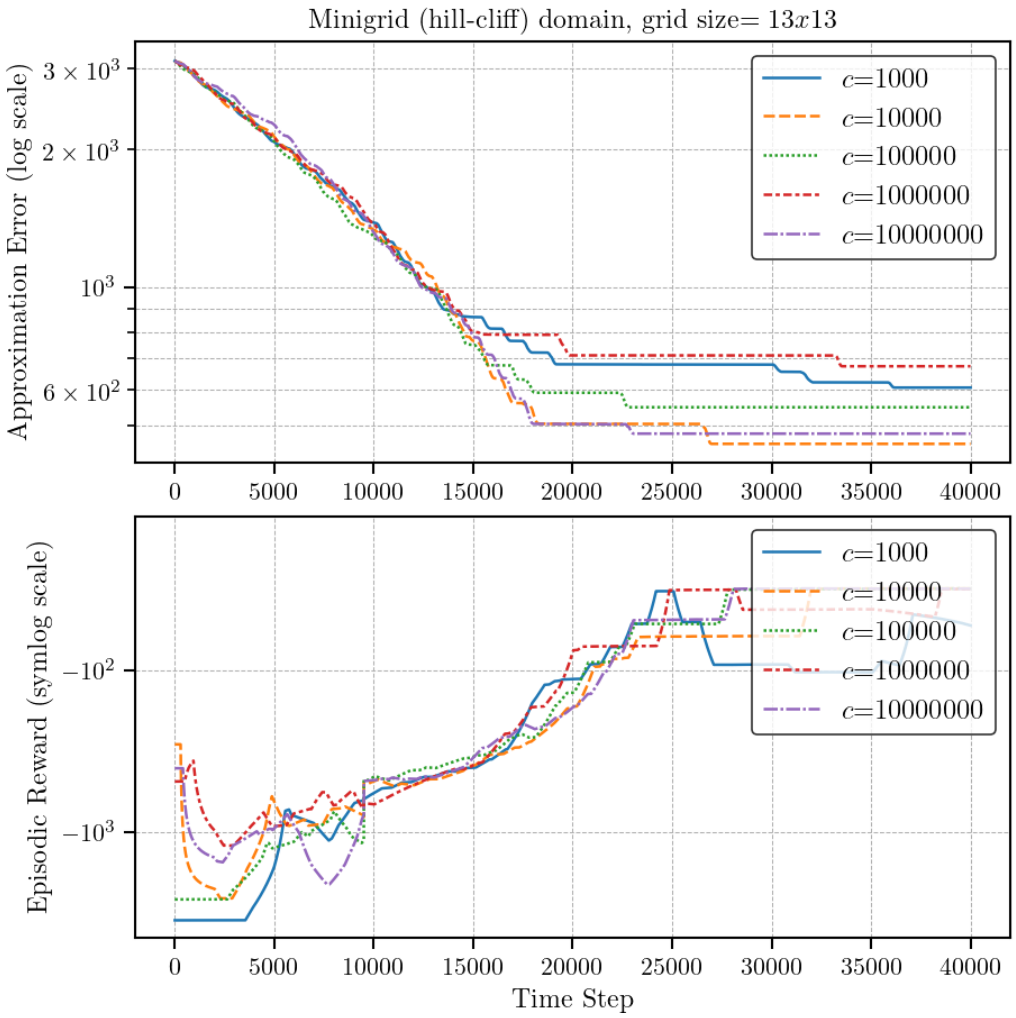

Figure D.19: Comparison of approximation error (top) and episodic reward (bottom) for different c values in a 13x13 hill-clif emulation grid world.

As we have observed, there is no single universal optimal solution for all settings, as the optimal choice of c is highly influenced by factors such as obstacles, non-desirable terminal states, and the complexity of the optimal path in the environment. However, $c = 10,000$ has consistently performed well across the explored settings, with $c = 100,000$ showing closer results in some cases but diverging in others. Therefore, the following figures showcase the comparison of values around $c = 10,000$ to identify the optimal choice for the learning rate.

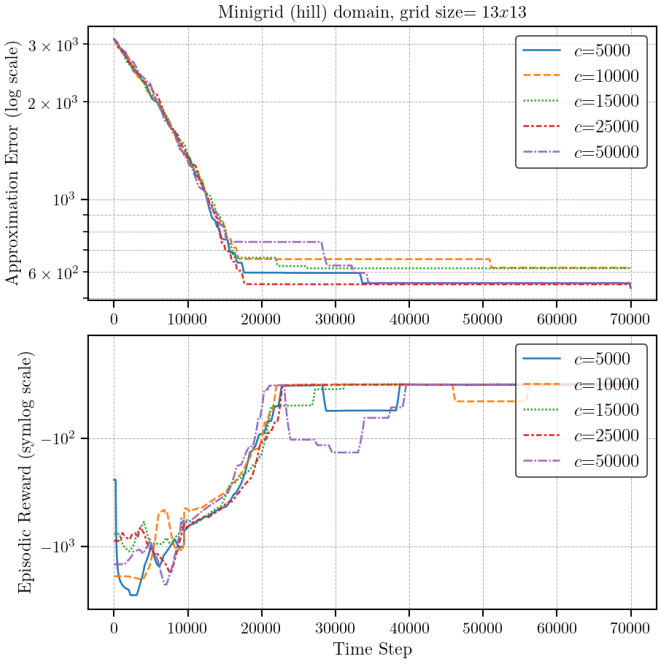

Figure D.20: Comparison of approximation error (top) and episodic reward (bottom) for different c values in a 13x13 hill-cliff emulation grid world.

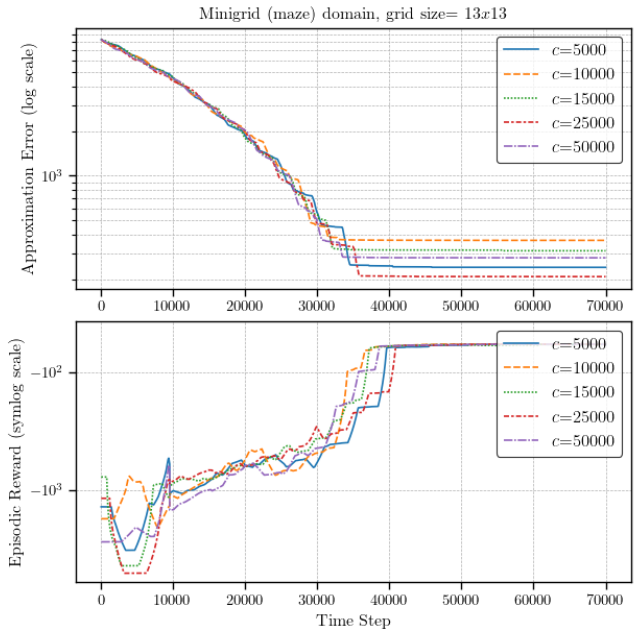

Figure D.21: Comparison of approximation error (top) and episodic reward (bottom) for different c values in a 13x13 maze-like grid world.

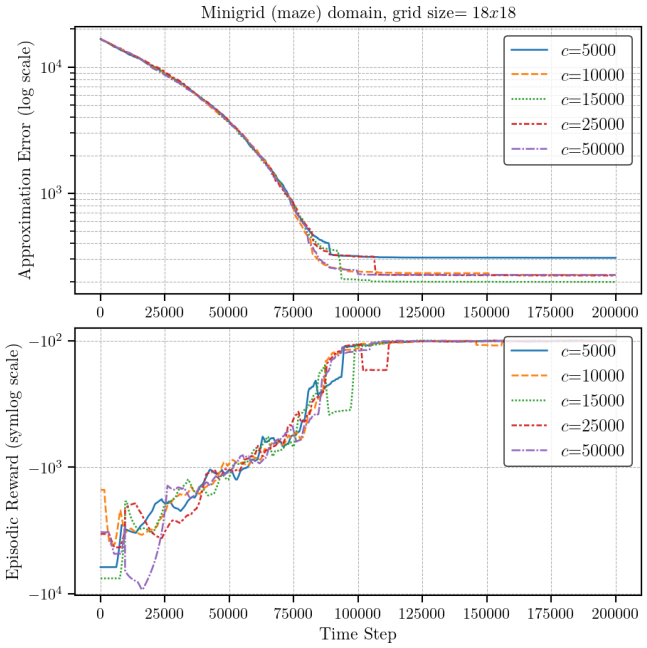

Figure D.22: Comparison of approximation error (top) and episodic reward (bottom) for different c values in a 18x18 maze-like grid world.

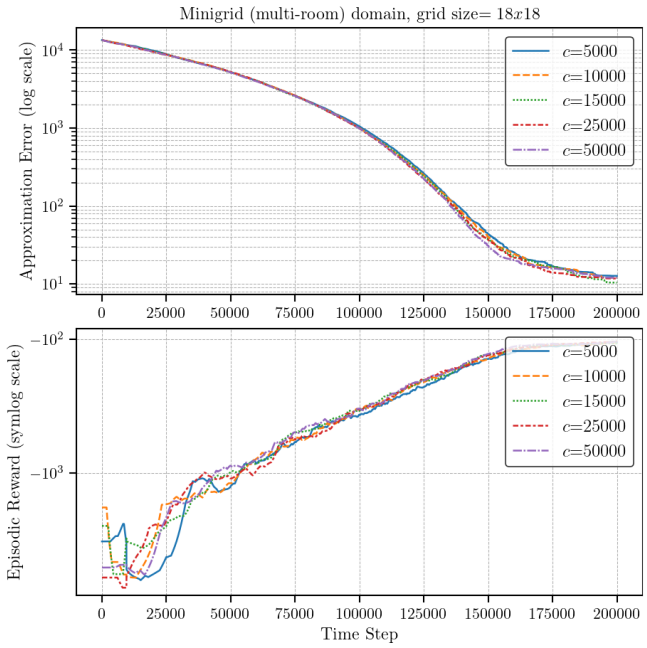

Figure D.23: Comparison of approximation error (top) and episodic reward (bottom) for different c values in a 18x18 multi-room grid world.

As illustrated in the previous figures, the range between $5,000$ and $50,000$ yields very similar solutions across all settings. Upon closer examination of the approximation error, values around $15,000$ and $25,000$ emerge as the most consistent across various settings. However, unlike the exploration parameters, there is no strong preference for any specific value within this range.

D.2.2 Temperature Parameter

The temperature parameter λ is crucial in modulating the deviation of optimal solutions from the inherent random dynamics of the problem. Through extensive experimentation, as depicted in the subsequent figures, it was determined that the optimal value for λ is consistently 1. Values smaller than 0.5 led to inaccuracies due to decimal underflow in the exponential calculations, while higher values produced suboptimal results in comparison to the standard $\lambda = 1$. Although values close to 1 yielded better episodic rewards similar to those achieved with $\lambda = 1$, they still resulted in significantly poorer value function approximations.

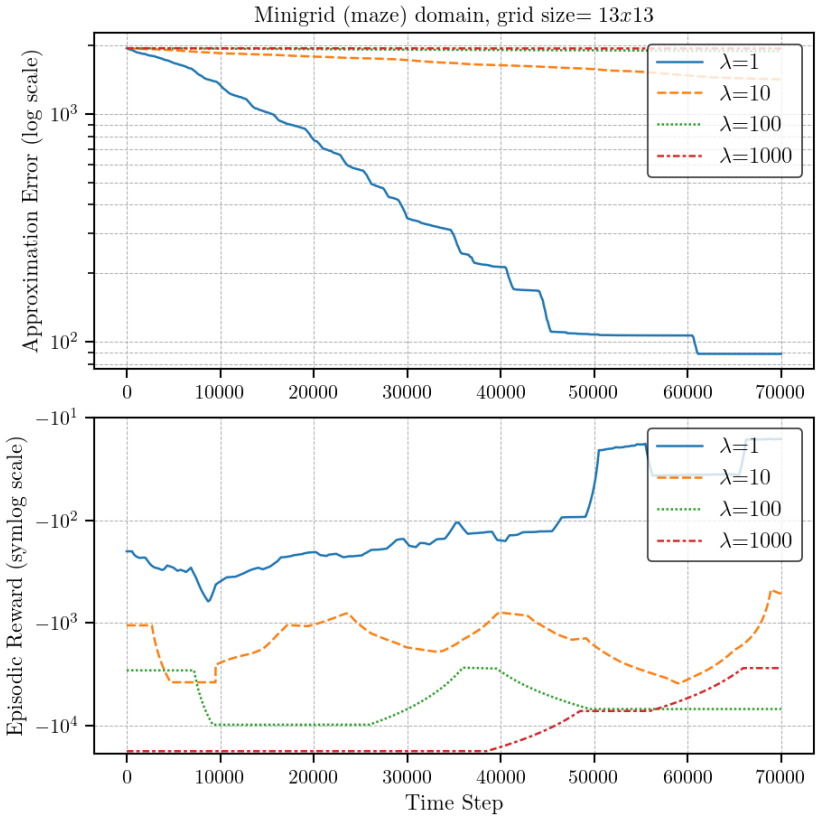

Figure D.24: Comparison of approximation error (top) and episodic reward (bottom) for different λ values in a 13x13 maze-like grid world.

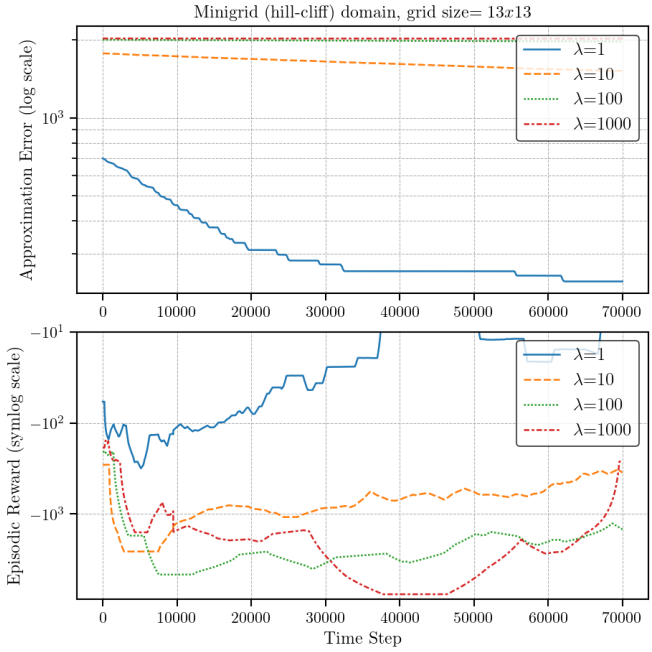

Figure D.25: Comparison of approximation error (top) and episodic reward (bottom) for different λ values in a 13x13 hill-cliff emulation grid world.

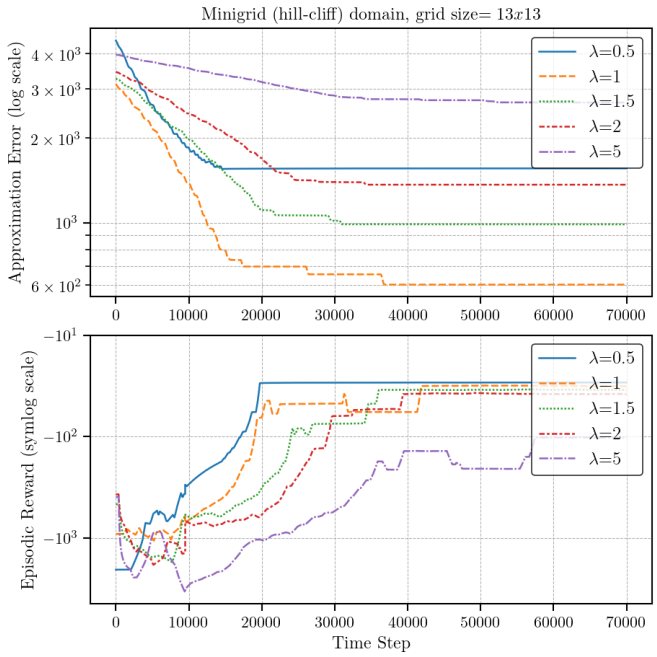

Figure D.26: Comparison of approximation error (top) and episodic reward (bottom) for different λ values in a 13x13 hill-cliff emulation grid world.

Appendix E

EMBEDDINGS EFFICIENCY

E.1 Stochastic MDP Embedding into LMDP

One of the main objectives of this research is to develop efficient and scalable methods for MDPs, with a particular emphasis on LMDPs. Given the computational challenges associated with solving these processes, especially in large-scale environments, a significant focus has been placed on improving both the precision and efficiency of embedding techniques. By optimizing the embedding process, this work aims to facilitate faster and more reliable solutions.

Table (E.1) presents the execution times for embedding an MDP with stochastic dynamics into an LMDP across various simple empty grid settings, measured on a standard academic computer. The vectorized implementation developed in this work demonstrates a significant improvement in efficiency. For environments with approximately 100,000 states, the vectorized approach is about 15% faster compared to the loop-based implementation. This efficiency gain is even more pronounced in smaller settings, where the vectorized approach achieves up to a 90% reduction in execution time.

Despite the relatively small differences in execution times between the vectorized and loop-based techniques in absolute terms, the observed efficiency improvements could be highly beneficial for significantly larger settings. While the current work was limited by computational resources and could not explore these larger environments, the methods developed here are designed to be scalable and applicable in future research. By providing a more efficient approach to embeddings, this work lays the groundwork for tackling much larger and more complex MDPs and LMDPs.

Grid Size	Number of States	Loop Time (s)	Vectorized Time (s)
2	4	0.00042	0.00007
3	9	0.00120	0.00021
5	25	0.00311	0.00053
10	100	0.01111	0.00153
15	225	0.02439	0.00315
20	400	0.04360	0.00596
30	900	0.11308	0.02454
40	1600	0.20145	0.06201
50	2500	0.34319	0.13674
60	3600	0.55813	0.27363
70	4900	0.84053	0.49956
80	6400	1.34381	0.94295
90	8100	1.78063	1.38341
100	10000	2.36575	1.95391

Table E.1: Embedding execution times for different grid sizes using vectorized and loop approaches.

E.2 LMDP Embedding into MDP

Table (E.2) presents the execution times for embedding an LMDP into an MDP across various grid sizes. The vectorized version for the embedding from LMDP to MDP is noticeably faster than the original loop-based version. This is especially pronounced in medium-to-large settings consisting of thousands of states. However, as the state space approaches tens of thousands, the timings become more balanced, possibly due to the overhead of handling such a large state space not being sufficiently offset by the vectorized operations.

These results demonstrate the efficiency of the vectorized implementation, making it highly suitable for any experimentation involving various settings and for multiple comparisons and embedding constructions simultaneously, due to the reduced computation times. Additionally, these embedding techniques are designed to work for any type of LMDP construction across diverse environments, as outlined in the previous class definitions. This scalability and adaptability make the approach well-suited for benchmarking, testing, and comparing methods over a wide range of environment sizes. Moreover, The approximation errors observed between the loop-based and vectorized methods are remarkably close, differing by an infinitesimal margin. Specifically, the approximation errors in both methods are exactly equivalent within a scale of 10^{-24} in Mean Squared Error.

Grid Size	Number of States	Loop Time (s)	Vectorized Time (s)
2	4	0.00844	0.00590
3	9	0.01250	0.00845
5	25	0.02302	0.00899
10	100	0.08089	0.01512
15	225	0.18019	0.02333
20	400	0.33202	0.05006
30	900	0.83924	0.21857
40	1600	1.92297	0.83924
50	2500	3.58723	2.03189
60	3600	7.30093	4.79076
70	4900	14.47642	9.59661
80	6400	20.80158	16.69498
90	8100	33.92493	29.42800
100	10000	54.42098	51.65870

Table E.2: Embedding execution times for different grid sizes using vectorized and loop approaches.

Furthermore, it is important to note that these embeddings rely on other methods, such as power iteration, which have also been optimized. Without these optimizations, the times reported in Tables E.1 and E.2 would be significantly longer.

Appendix F

EMBEDDINGS PRECISION

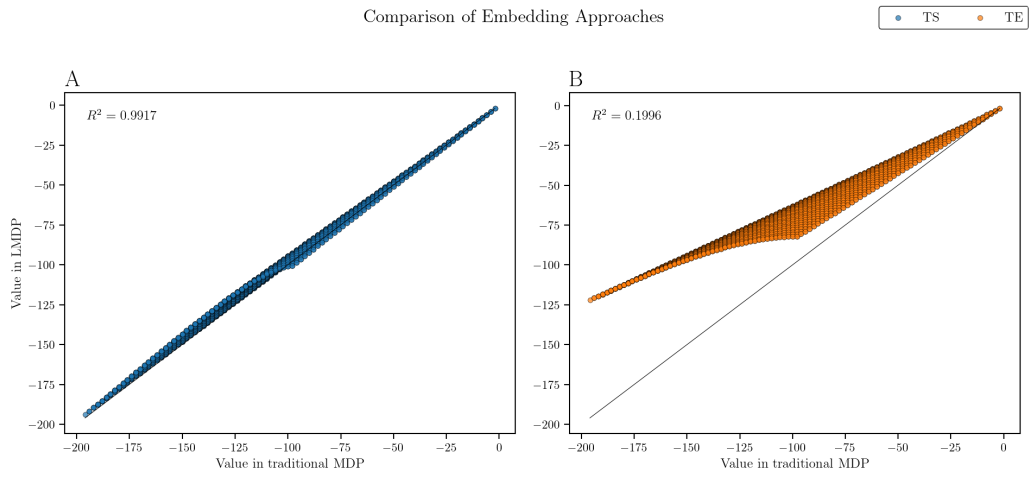

Figure F.1: Embedding Approximation by Ternary Search and Todorov's Method

Figure (F.1) showcases the embedding approximations from Todorov's original approach and the implementation used in this work. The environment used was a simple grid with four possible actions leading to the neighboring states and no walls. The terminal states, as in any embedding, receive the same value function as their rewards, $v(s) = \max_{a \in \mathcal{A}} \tilde{\mathcal{R}}(s, a)$ for all $s \in \mathcal{T}$, which is why they are not considered in these plots. Each dot corresponds to a state, with a total of 2,500.

This setting, despite having fewer states than the one illustrated in Figure (3.2), is larger. The reason for this is that we are now considering a simple grid setting with only one possible state per cell, whereas in the prior example there were four states per cell, one for each agent orientation. Thus, despite having fewer states, the value function range is wider, with higher absolute values. This is why the

embedding precision is considerably worse than before in the case of Todorov's Embedding (TE), with an R^2 of 0.1996 and an MSE of 1330.32. However, having a larger setting does not degrade the performance of our method, which still achieves an R^2 of 0.9917 and an MSE of 13.76, exactly 1% of the approximation error obtained with the baseline approach.

F.1 Ternary Search in Multidimensional Transition Dynamics Settings

In settings such as Minigrid, where uniform dynamics exist, the Ternary Search (TS) method clearly outperforms the Stochastic Policy Averaging (SPA) method. This improvement is attributed to the consistency in state transitions and the fixed number of actions, which allow TS to effectively refine the scaling factor and achieve a more accurate value function approximation. However, in cases where column-rank deficiency occurs, such as when there are more symbolic actions than possible next states in some states, an exact embedding cannot be constructed. This issue was highlighted by Todorov [Todorov, 2006] and is observed in the context of stochastic embeddings, where the precision slightly decreases in multidimensional dynamic settings.

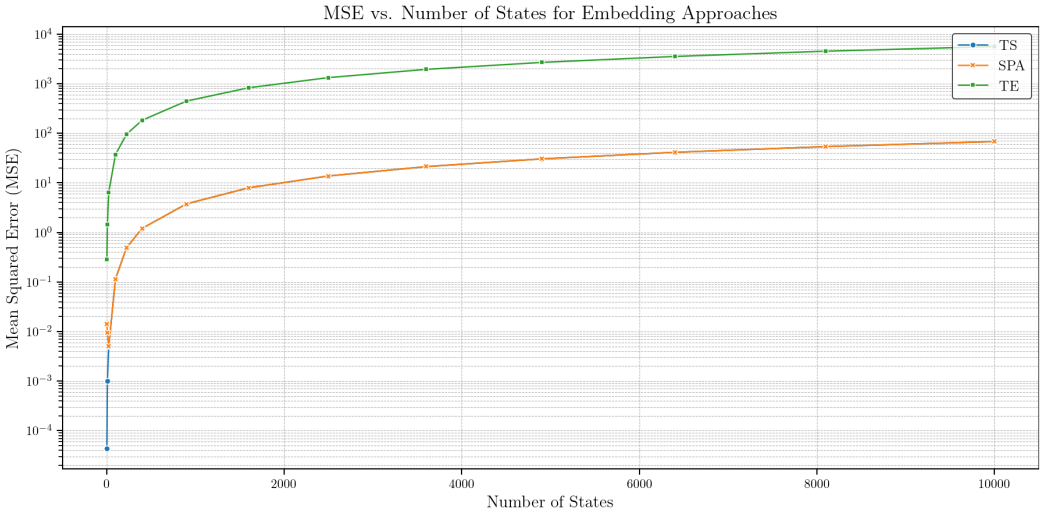

Figure F.2: Embedding MSE Comparison in Simple Grid

In deterministic environments, this column-rank deficiency results in TS converging to the same solution as SPA. Both methods find the optimal approximation near $K = 1$, which indicates that the most accurate approximation is inherently close to the SPA algorithm's initialization. Consequently, TS and SPA yield the same approximation Mean Squared Error (MSE) in these cases.

Despite this, both methods demonstrate significantly better performance compared to Todorov's original embedding approach for deterministic dynamics, as shown in Figure (F.2). This figure illustrates the embedding MSE comparison in a simple grid environment, where both TS and SPA achieve lower MSE values, highlighting their effectiveness in handling deterministic environments.Therefore, the TS method not only proves to be superior in uniform dynamic settings but also performs optimally in scenarios where an exact embedding cannot be constructed due to multidimensional transition dynamics. This comprehensive embedding approach is more adaptable and closer to the theoretical optimum than the baseline method, making it a robust solution for a wide range of Markov Decision Process (MDP) scenarios.

Appendix G

OTHER ALGORITHMS

Algorithm 8 Vectorized Value Iteration Algorithm

input: MDP with state space $\mathcal{S}$, terminal state space $\mathcal{T}$, non-terminal state space $\mathcal{S}^-$, action space $\mathcal{A}$, reward matrix $\mathcal{R}$, transition matrix $\mathcal{P}$, discount factor γ, and convergence threshold ϵ
output: Action-value function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, policy $\pi : \mathcal{S} \to \mathcal{A}$, and number of iterations N
initialize: $Q(s, a) \leftarrow 0$ for all $s \in \mathcal{S}^-$, for all $a \in \mathcal{A}$, $Q(s, a) \leftarrow \mathcal{R}(s, a)$ for all $s \in \mathcal{T}$, for all $a \in \mathcal{A}$, $N \leftarrow 0, V_{diff}$ with the state indices
$R \leftarrow \mathcal{R}(\mathcal{S}^-)$
$P \leftarrow \gamma\mathcal{P}$
$QT \leftarrow \mathcal{R}(\mathcal{T})$
while $\max(V_{diff}) - \min(V_{diff}) > \epsilon$ **do**
    $TQ \leftarrow R + P \max_a(Q)$
    $TQ \leftarrow (TQ, QT)$
    $V_{diff} \leftarrow \max_a(TQ) - \max_a(Q)$
    $Q \leftarrow TQ$
    $N \leftarrow N + 1$
end while
$\pi \leftarrow \arg_a \max(Q)$
return Q, π, N

Algorithm 9 Power Iteration Algorithm

input: LMDP with initial state distribution $\mathcal{P}$, state space $\mathcal{S}$, terminal state space $\mathcal{T}$, non-terminal state space $\mathcal{S}^-$, reward matrix $\mathcal{R}$, temperature parameter λ, and convergence threshold ϵ
output: Optimal Z function and number of iterations N
initialize: $Z(s) \leftarrow 1$ for all $s \in \mathcal{S}^-$, $Z(s) \leftarrow e^{\mathcal{R}(\mathcal{T})/\lambda}$ for all $s \in \mathcal{T}$, $N \leftarrow 0, V_{diff}$ with the state indices
$ZT \leftarrow e^{\mathcal{R}(\mathcal{T})/\lambda}$
$G \leftarrow e^{\mathcal{R}(\mathcal{S}^-)/\lambda}$ as a diagonal matrix
while $\max(V_{diff}) - \min(V_{diff}) > \epsilon$ **do**
    $TZ \leftarrow G\mathcal{P}Z$
    $TZ \leftarrow (TZ, ZT)$
    $TV \leftarrow \lambda \log TZ$
    $V \leftarrow \lambda \log Z$
    $V_{diff} \leftarrow (TZ) - (Z)$
    $Z \leftarrow TZ$
    $N \leftarrow N + 1$
end while
return Z, N

Algorithm 10 Deterministic MDP Embedding through iterative KL update

input: MDP $\mathcal{M}$ with $\mathcal{A}$, $\mathcal{S}$, $\mathcal{T}$, $\mathcal{S}^-$, $\tilde{\mathcal{R}}$ and deterministic $\tilde{\mathcal{P}}$, temperature parameter λ, optimal value function V^* from $\mathcal{M}$, convergence threshold ϵ, stopping criteria parameter N
output: LMDP $\mathcal{L}$ with $\hat{\mathcal{R}}$ and $\mathcal{P}$
$\hat{\mathcal{R}}(\cdot) \leftarrow \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \tilde{\mathcal{R}}(\cdot, a)$
$\mathcal{P}(\cdot \mid \cdot) \leftarrow \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \tilde{\mathcal{P}}(\cdot|\cdot, a)$
Obtain Z from power iteration for LMDP $\mathcal{L}$ with $\mathcal{P}$ and $\hat{\mathcal{R}}$
$V = \lambda \log Z$
$n = 0$
while $MSE(V^*, V) > \epsilon$ and $n < N$ **do**
    $\mathcal{P}_{\mathbf{u}^*} \leftarrow \mathcal{P}Z / \sum_{s' \in \mathcal{S}} \mathcal{P}(s' \mid \cdot) Z(s')$
    $\hat{\mathcal{R}} \leftarrow \hat{\mathcal{R}} + \lambda KL(\mathcal{P}_{\mathbf{u}^*} \| \mathcal{P})$
    $n \leftarrow n + 1$
end while
return $\mathcal{L}$ with $\hat{\mathcal{R}}$ and $\mathcal{P}$

Algorithm 11 Deterministic MDP Embedding through binary search

input: MDP $\mathcal{M}$ with $\mathcal{A}$, $\mathcal{S}$, $\mathcal{T}$, $\mathcal{S}^{-}$, $\tilde{\mathcal{R}}$ and deterministic $\tilde{\mathcal{P}}$, temperature parameter λ, small ϵ, optimal value function V^{*} from $\mathcal{M}$
output: LMDP $\mathcal{L}$ with $\mathcal{R}$
$\hat{\mathcal{R}}(\cdot) \leftarrow \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \tilde{\mathcal{R}}(\cdot, a)$
$\mathcal{P}(\cdot|\cdot) \leftarrow \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \tilde{\mathcal{P}}(\cdot|\cdot, a)$
Obtain Z from power iteration method for LMDP $\mathcal{L}$ with $\mathcal{P}$ and $\hat{\mathcal{R}}$
$\mathcal{P}_{\mathbf{u}^{*}} \leftarrow \mathcal{P}Z / \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|\cdot) Z(s')$
$\hat{\mathcal{R}} \leftarrow \hat{\mathcal{R}} + \lambda KL(\mathcal{P} \| \mathcal{P}_{\mathbf{u}^{*}})$
Set $K_{min} = 0$ and $K_{max} = 1$
$V^{*}_{MS} = \frac{1}{|\mathcal{S}|} \left(\sum_{s \in \mathcal{S}} V^{*} \right)^{2}$
while $K_{max} - K_{min} \geq \epsilon$ **do**
    $K = (K_{min} + K_{min}) / 2$
    $\hat{\mathcal{R}}' = K \cdot \hat{\mathcal{R}}$
    Obtain Z' from power iteration method for LMDP $\mathcal{L}$ with $\mathcal{P}$ and $\hat{\mathcal{R}}'$
    $V = \lambda \log Z'$
    $V_{MS} = \frac{1}{|\mathcal{S}|} \left(\sum_{s \in \mathcal{S}} V \right)^{2}$
    if $V^{*}_{MS} > V_{MS}$ **then**
        $K_{min} = K$
    else
        $K_{max} = K$
    end if
end while
$\mathcal{R} = K \cdot \hat{\mathcal{R}}$
return $\mathcal{L}$ with $\mathcal{P}$ and $\mathcal{R}$

Algorithm 12 Q-learning with ϵ-decay

input: discount rate $\gamma \in [0, 1]$, exploration factor $\epsilon \in (0, 1]$, ϵ-decay parameter $d_\epsilon \in (0, 1]$, $min_\epsilon \in [0, 1)$, $c \in \mathbb{R}$, restart randomness parameter r_{s_0}, MDP with $\mathcal{R}$, $\mathcal{P}$, $\mathcal{S}$, $\mathcal{A}$, $\mathcal{S}^-$, $\mathcal{T}$, s_0
output: $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$
initialize $Q(s, a) \leftarrow 0$ for all $s \in \mathcal{S}^-$, $a \in \mathcal{A}$, $Q(s, a) \leftarrow \mathcal{R}(s, a)$ for all $s \in \mathcal{T}$, $a \in \mathcal{A}$, $N = 0$, $\alpha = 1$
repeat
    if $r_{s_0} = 0$ **then**
        $s_t = s_0$
    else
        Sample a random state s_t from $\mathcal{S}^-$
    end if
    while $s_t \notin \mathcal{T}$ **do**
        Choose a from s_t using ϵ-greedy policy derived from Q
        Take action a, observe r, s_{t+1}
        $Q(s_t, a) \leftarrow Q(s_t, a) + \alpha \left[r + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a) \right]$
        $s_t \leftarrow s_{t+1}$
    end while
    $N \leftarrow N + 1$
    $\epsilon \leftarrow \max \left(min_\epsilon, \epsilon \cdot d_\epsilon \right)$
    $\alpha \leftarrow c / \left(c + N \right)$
until convergence
output: Q

Algorithm 13 Z-learning

input: temperature parameter $\lambda \in \mathbb{R}$, $c \in \mathbb{R}$, restart randomness parameter r_{s_0}, LMDP $\mathcal{L}$ with $\mathcal{R}$, $\mathcal{P}$, $\mathcal{S}$, $\mathcal{A}$, $\mathcal{S}^-$, $\mathcal{T}$, s_0
output: $Z : \mathcal{S} \to \mathbb{R}$
initialize $Z(s) \leftarrow 1$, for all $s \in \mathcal{S}^-$ and $Z(s) \leftarrow e^{\mathcal{R}(s)/\lambda}$ for all $s \in \mathcal{T}$, $\mathcal{P}_{\mathbf{u}^*} = \mathcal{P}$, $N = 0$
repeat
    if $r_{s_0} = 0$ **then**
        $s_t = s_0$
    else
        Sample a random state s_t from $\mathcal{S}^-$
    end if
    while $s_t \notin \mathcal{T}$ **do**
        Take reward r_t from current state s_t
        Sample a next state s_{t+1} according to $\mathcal{P}_{\mathbf{u}}$.
        $\mathcal{G}\left[z\right](s_t) = \sum_{s' \in \mathcal{S}} \mathcal{P}(s' \mid s_t) z(s')$
        $Z(s_t) \leftarrow Z(s_t) + \alpha \left[e^{r_t/\lambda} \mathcal{G}\left[z\right](s_t) - Z(s_t) \right]$
        Update $\mathcal{P}_{\mathbf{u}}$
        $s_t \leftarrow s_{t+1}$
    end while
    $N \leftarrow N + 1$
    $\alpha \leftarrow c / \left(c + N\right)$
until convergence
return Z

Appendix H

GRID WORLD DOMAINS

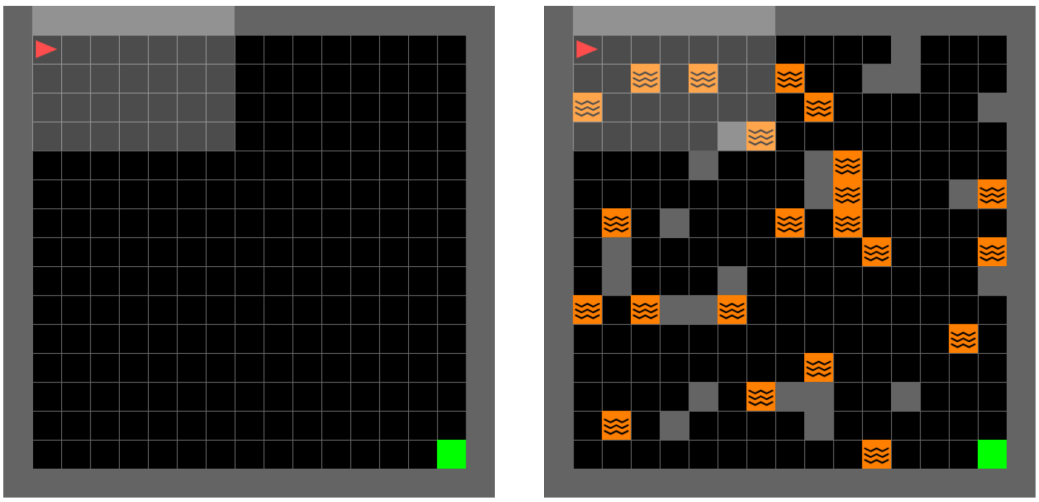

Figure H.1: 15x15 empty grid world domain (left) and 15x15 non-empty grid world domain (right).

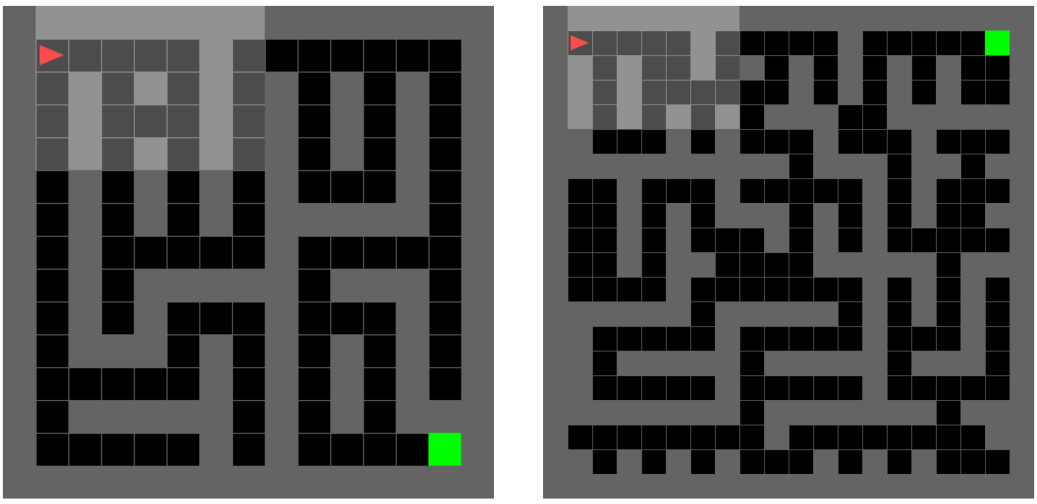

Figure H.2: 13x13 maze-like grid world domain (left) and 18x18 maze-like grid world domain (right).

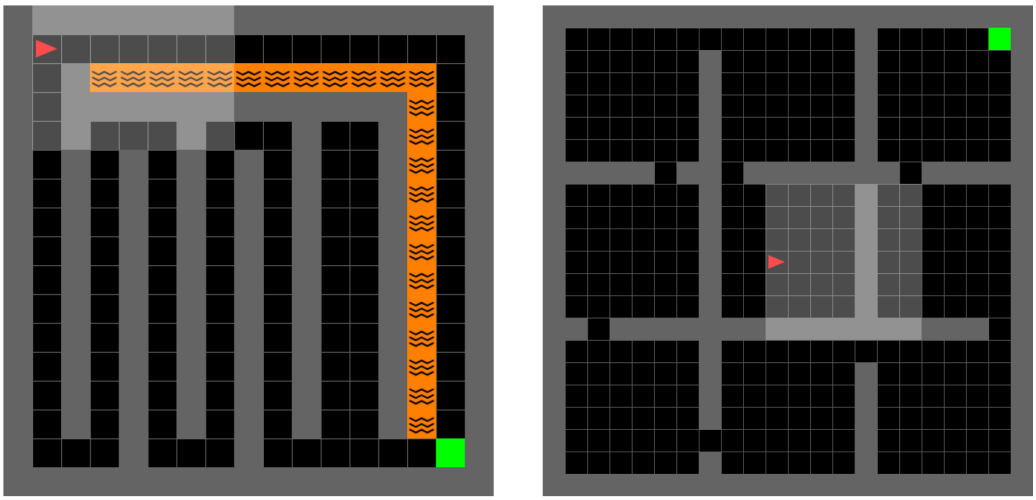

Figure H.3: 13x13 hill-cliff emulation grid world domain (left) and 18x18 multi-room grid world domain (right).