Compositionality for Hierarchical Reinforcement Learning

Guillermo Infante Molina

TESI DOCTORAL UPF / Year 2024

THESIS SUPERVISOR
Anders Jonsson, Vicenç Gómez
Department de Tecnologies de la Informació i les Comunicacions

Write here your dedication

Acknowledgements

Preface

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Thesis structure

This thesis follows the structure below in the upcoming chapers:

- Chapter 2 includes all the necessary technical background and notation to understand the succeeding chapters 2.
- Chapter 3 introduces a method for hierarchical Linearly-solvable markov decision processes. This work was published under the name of *"Globally optimal hierarchical reinforcement learning for linearly-solvable markov decision processes"* in the Proceedings of the 36th AAAI Conference on Artificial Intelligence in 2022.
- Chapter 3 follows up the previous work and extends the method for the average-reward setting. This work was recently accepted for publication as *"Hierarchical Average-Reward Linearly-solvable Markov Decision Processes"* in the 27th European Conference on Artificial Intelligence.
- In Chapter 4, a framework to exploit compositionality for solving complex task in a more general reinforcement learning setting is presented. Part of this work was published as a paper and presented at the 37th of the International Conference on Automated Planning and Scheduling. The paper is *"Planning with a Learned Policy Basis to Optimally Solve Complex Tasks"* and is the product of a four months research stay at the University of Amsterdam, under the supervision of Herke van Hoof.
- Lastly, in Chapter 5 the final remarks and conclusions of the thesis are discussed as well as some possible lines of future work.

1.2 Contributions

Chapter 2

Background

In this chapter we introduce and clarify the technical concepts to necessary understand the subsequent chapters. We also introduce the following notation that is used throughout the dissertation:

- Given a finite set $\mathcal{X}$, let $\Delta(\mathcal{X}) = \{p \in \mathbb{R}^{\mathcal{X}} : \sum_x p(x) = 1, p(x) \geq 0 \ (\forall x)\}$ we denote the probability simplex on $\mathcal{X}$.
- Given a probability distribution $p \in \Delta(\mathcal{X})$, we let $\mathcal{B}(p) = \{x \in \mathcal{X} : p(x) > 0\} \subseteq \mathcal{X}$ denote the support of p.
- Given a set $\mathcal{X}$, let$|\mathcal{X}|$ we denote the cardinality of set $\mathcal{X}$.
- We use capital, non-caligraphic letters to express random variables while their lowercase counterpart represents a realization of such a random variable. E.g: S_t is a random variable that denotes a state at timestep t and s_0 is a realization of such random variable.

2.1 Markov decision processes

Stochastic sequential decision problems are usually modeled as Markov decision processes (MDPs). Without loss of generality, we restrict our attention to countable MDPs which are formally defined as tuples $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \mathbb{P}_0 \rangle$, where:

- $\mathcal{S}$ is a countable set of states.
- $\mathcal{A}$ is a finite set of primitive actions.

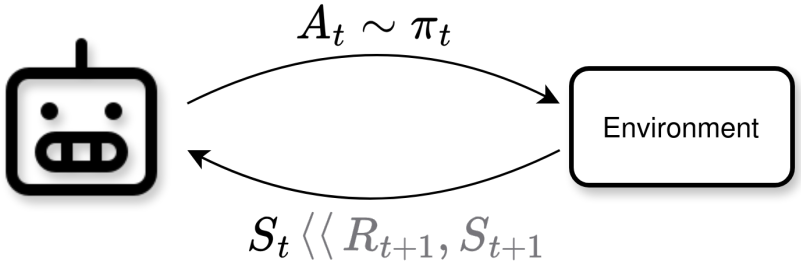


Figure 2.1: Interaction loop in a Markov decision process. The agent observes current state S_t and chooses $A_t \sim (\cdot|S_t)$. The agent receives a feedback (reward) from the environment and a new state which from the point of view of the agent becomes the new current state.

- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [R_{\text{MIN}}, R_{\text{MAX}}]$ is a reward function. Though the reward can also be defined as $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to [R_{\text{MIN}}, R_{\text{MAX}}]$ or $\mathcal{R} : \mathcal{S} \to [R_{\text{MIN}}, R_{\text{MAX}}]$, we stick to its most general definition that associates rewards to triplets $(s, a, s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}$, and modify it as necessary.
- $\mathbb{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is a transition function that encodes the probability distribution $\mathbb{P}(\cdot|s, a)$ over next states for every $(s, a) \in \mathcal{S} \times \mathcal{A}$.
- $\mathbb{P}_0 : \Delta(\mathcal{S})$ is the initial state distribution. We assume that $\mathbb{P}_0(s) > 0 \; \forall s \in \mathcal{S}$.

The most general form of the interaction between the learning agent and the environment is depicted in Figure 2.1. Initially, the environment is in a start state that is sampled from the initial state distribution $S_0 \sim \mathbb{P}_0$. At any timestep t, the agent observes a state S_t and chooses an action $A_t \sim \pi(S_t)$ according to a decision rule π. Such a decision rule is called policy, and it is a function $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ that maps states to distributions over actions. Once the agent executes action A_t in the environment, it receives a reward $R_{t+1} = \mathcal{R}(S_t, A_t, S_{t+1})$ and a new state $S_{t+1} \sim \mathbb{P}(\cdot|\mathcal{S}_t, A_t)$. When access to a simulator is available it is a common practice to restart the interaction every some timesteps according to the initial state distribution $\mathcal{P}_0$.

The quadruple $(s_t, a_t, r_{t+1}, s_{t+1})$ defines a transition while the sequence of transitions up to a certain timestep,

$$h_t = (s_0, s_0, r_1, s_1, \ldots, r_t, s_t),$$

is called history. We let $\mathcal{H}$ denote the set of all possible histories. Note that the interaction of an agent with the environment can be described at any time by means of elements $h \in \mathcal{H}$.

These processes are so-called Markov because the Markov property holds both in the reward and transition probability functions. Formally,

$$\mathbb{P}(s_{t+1}|h_t) = \mathbb{P}(s_{t+1}|s_t, a_t),$$
$$\mathcal{R}(s_t, a_t, s_{t+1}|h_t) = \mathcal{R}(s_t, a_t, s_{t+1}).$$

This indicates that at timestep t the reward and next state depends solely on the current state (or state and action), and not the full history. This has an important implication since the current state s_t conveys all the necessary information for the agent to learn how to behave optimally.

There are many ways to design policies, but our previous definition implies that we consider policies that:

- are Markovian, in the sense that the choice of the action depends on the current state $s_t \in \mathcal{S}$ and not the whole trajectory $h_t \in \mathcal{H}$,
- are stationary, because they remain the same over time. To be more precise, the choice of the action given a state does not depend on the timestep t in which the state is observed,
- are stochastic as they represent a distribution over actions. More concretely a distribution $\pi(\cdot|s)$ conditioned on the state $s \in \mathcal{S}$.

Unless otherwise specified, the policies are assumed to satisfy the characteristics just mentioned. We will also make use of the notion of deterministic policies. These are special cases of stochastic policies in which all the probability mass is put over one single action.

The aim of the learning agent is to come up with policies that optimize some numerical objective. We turn our attention to two optimality criteria: the discounted and the average-reward frameworks.

In the following sections we describe how we can solve each case when $\mathbb{P}$ and $\mathcal{R}$ are fully disclosed to the agent, via dynamic-programming techniques. This setting, in which the reward and transition functions are known, is also referred to as planning in MDPs.

2.1.1 The discounted case

In the discounted setting, an optimal policy is such that it maximizes the expected discounted return. The discounted return is given by the sum of discounted rewards,

$$G_t = \sum_{i=t}^{\infty} \gamma^{i-t} \mathcal{R}(S_i, A_i, S_{i+1}). \tag{2.1}$$

Here, $0 < \gamma < 1$ is the discount factor. There are several reasons to use the discount factor, such as giving more credit to rewards closer in time. Notwithstanding, its main purpose is to make the return in Equation 2.1 have a finite value.

Given a policy π, we define its value function, $v^\pi : \mathcal{S} \to \mathbb{R}$, as the expected discounted return of being at state $s \in \mathcal{S}$ and following policy π

$$v^\pi(s) = \mathbb{E}_\pi \left[\sum_{i=t}^{\infty} \gamma^{i-t} R_i \,\middle|\, S_t = s \right] \; \forall s \in \mathcal{S}, \tag{2.2}$$

where R_i is a shorthand for $\mathcal{R}(S_i, A_i, S_{i+1})$. We also define the action-value function, $q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, of a state-action pair as the expected discounted return of being at state $s \in \mathcal{S}$, choosing action $a \in \mathcal{A}$ and following policy π thereafter,

$$q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{i=t}^{\infty} \gamma^{i-t} R_i \,\middle|\, S_t = s, A_t = a \right] \; \forall (s, a) \in \mathcal{S} \times \mathcal{A}. \tag{2.3}$$

We use $\mathcal{R}(s, a) = \mathbb{P}(s'|s, a)\mathcal{R}(s, a, s')$ to denote the mean reward. The value function for state $s \in \mathcal{S}$ is known to satisfy the following Bellman equations:

$$\begin{aligned}
v^\pi(s) &= \mathbb{E}_\pi \left[\sum_{i=t}^{\infty} \gamma^{i-t} R_i \,\middle|\, S_t = s \right], \\
&= \mathbb{E}_\pi \left[\mathcal{R}(s, A_t, S_{t+1}) + \sum_{i=t+1}^{\infty} \gamma^{i-t+1} R_i \,\middle|\, S_t = s \right], \\
&= \sum_a \pi(a|s) \sum_{s'} \mathbb{P}(s'|s, a) \left(\mathcal{R}(a, a, s') + \mathbb{E}_\pi \left[\sum_{i=t}^{\infty} \gamma^{i-t+1} R_i \,\middle|\, S_t = s' \right] \right), \\
&= \pi(a|s)\mathbb{P}(s'|s, a) \left(\mathcal{R}(a, a, s') + \gamma \mathbb{E}_\pi \left[\sum_{i=t}^{\infty} \gamma^{i-t} R_i \,\middle|\, S_t = s' \right] \right) \\
v^\pi(s) &= \sum_a \pi(a|s) \left(\mathcal{R}(s, a) + \gamma \sum_{s'} \mathbb{P}(s'|s, a) v^\pi(s') \right).
\end{aligned} \tag{2.4}$$

And similarly, for the action-value function for state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$:

$$q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{i=t}^{\infty} \gamma^{i-t} R_i \,\middle|\, S_t = s, A_t = a \right],$$

$$= \mathbb{E}_\pi \left[\mathcal{R}(s, a, S_{t+1}) + \sum_{i=t+1}^{\infty} \gamma^{i-t+1} R_i \,\middle|\, S_t = s, A_t = a \right],$$

$$= \mathbb{P}(s'|s, a) \left(\mathcal{R}(s, a, s') + \mathbb{E}_\pi \left[\sum_{i=t+1}^{\infty} \gamma^{i-t+1} R_i \,\middle|\, S_t = s' \right] \right),$$

$$= \mathcal{R}(s, a) + \gamma \mathbb{P}(s'|s, a) \mathbb{E}_\pi \left[\sum_{i=t}^{\infty} \gamma^{i-t} R_i \,\middle|\, S_t = s' \right],$$

$$q^\pi(s, a) = \mathcal{R}(s, a) + \gamma \mathbb{P}(s'|s, a) v^\pi(s') \; \forall (s, a). \tag{2.5}$$

Combining Equations (2.4) and (2.5) yields

$$v^\pi(s) = \sum_a \pi(a|s) q^\pi(s, a). \tag{2.6}$$

The goal of the agent is to compute a policy π^* that maximizes the expected discounted return. We denote the optimal value and action-value functions attained by an optimal policy as v^* and q^*, respectively. Note that the optimal value function and action-value function, considering that the optimal policy is deterministic, are related by

$$v^*(s) = \max_a q^*(s, a). \tag{2.7}$$

The question is how to obtain such optimal value functions, so we can derive the optimal policy.

First, we describe how we can obtain the value function associated with a policy π. This procedure is usually known as *policy evaluation*. For such, we declare the following Bellman operator $T^\pi : \mathbb{R}^{\mathcal{S}} \to \mathbb{R}^{\mathcal{S}}$ as

$$(T^\pi v^\pi)(s) = \sum_a \pi(a|s) \left(\mathcal{R}(s, a) + \gamma \sum_{s'} \mathbb{P}(s'|s, a) v^\pi(s') \right) \; \forall s \in \mathcal{S}. \tag{2.8}$$

By looking at v^π as a vector of the appropriate size, Equation 2.8 can be rewritten in vector form as

$$T^\pi v^\pi = v^\pi. \tag{2.9}$$

Therefore, computing the true value of v^π requires finding the fixed-point solution of the previous system of linear equations.

We further introduce the Bellman optimality operator $T^* : \mathbb{R}^{\mathcal{S}} \to \mathbb{R}^{\mathcal{S}}$ defined as

$$(T^* v^*)(s) = \max_a \mathcal{R}(s, a) + \gamma \sum_{s'} \mathbb{P}(s'|s, a) v^*(s') \; \forall s \in \mathcal{S}, \tag{2.10}$$

and this allows us to rewrite 2.10 in vector form:

$$T^* v^* = v^*. \tag{2.11}$$

The previous expression indicates that the optimal value function v^* is the fixed-point solution of the previous system of equations, which in this case is not linear as it requires a $\max$ operation.

We derive the first dynamic-programming algorithm called value iteration [Bellman, 1958] from Equation 2.11. Here, we keep an estimate v_k of the optimal value function which updated iteratively as

$$v_{k+1} \leftarrow T^* v_k, \tag{2.12}$$

for some arbitrary initialization v_0 until $|v_{k+1} - v_k| < \epsilon$, where ϵ is the user-specified precision. Value iteration can be shown to converge thanks to the contraction mapping theorem.

Add proof in the appendix??

The idea of the operators can be also applied to the action-value function. Thus, we define the Bellman operator $T^\pi : \mathbb{R}^{\mathcal{S}\times\mathcal{A}} \to \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$ and the Bellman optimality operator $T^* : \mathbb{R}^{\mathcal{S}\times\mathcal{A}} \to \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$ over action-value functions in the following way:

$$(T^\pi q^\pi)(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s'} \mathbb{P}(s'|s,a) v^\pi(s') \;\forall (s,a) \in \mathcal{S}\times\mathcal{A}. \tag{2.13}$$

$$(T^* q^*)(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s'} \mathbb{P}(s'|s,a) v^*(s') \;\forall (s,a) \in \mathcal{S}\times\mathcal{A}. \tag{2.14}$$

Value iteration can also be implemented using action-value function, in that case we keep an estimate q_k and update it iteratively as

$$q_{k+1} \leftarrow T^* q_k. \tag{2.15}$$

We also introduce the greedy operator $\mathcal{G} : \mathbb{R}^{\mathcal{S}} \to \mathcal{A}^{\mathcal{S}}$ over action-value functions which is defined as

$$(\mathcal{G})(s) = \arg\max_{a\in\mathcal{A}} q^\pi(s,a) \tag{2.16}$$

and returns a deterministic, greedy policy with respect to the action-value function which takes the form of

$$\pi(s|s) = \begin{cases} 1 & \text{if } a = \mathcal{G} q^\pi(s,\cdot) \\ 0 & \text{otherwise.} \end{cases}$$

The second dynamic-programming algorithm is *policy iteration* [Howard, 1960] which consists of interleaved steps of policy evaluation and policy improvement. This algorithm works as follows:

(1) Fix some initial policy π_0.

(2) At each iteration solve $T^{\pi_k} q^{\pi_k} = q^{\pi_k}$ (policy evaluation).

(3) Then derive new policy $\pi_{k+1} \leftarrow \mathcal{G} q^{\pi_k}$ (policy improvement).

(4) If $\pi_k(s) \neq \pi_{k+1}(s)$ for some $s \in \mathcal{S}$, repeat (2) and (3).

Policy iteration converges to an optimal policy π^*.

The episodic case. For certain problems discount factor can be deprecated, and we can set $\gamma = 1$ when using a suitable reward function. Consider the extended MDP definition $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathbb{P}, \mathbb{P}_0 \rangle$ where $\mathcal{T}$ is a set of absorbing, terminal states. This implies that $\mathbb{P}(s|s,a) = 1 \ \forall a \in \mathcal{A}$ for any $s \in \mathcal{T}$. In this type of problems the interaction halts when the agent reaches some $s \in \mathcal{T}$, and gets restarted to an initial state according to $\mathbb{P}_0$. This piece of the interaction is called an episode, therefore the name. This setting usually appears under the names of total-reward, first-exit case or goal-oriented MDP. The return is now expressed as

$$G_t = \sum_{i=0}^{T} \mathcal{R}(S_i, A_i, S_{i+1}),$$

where T is a random variable that represents the timestep in which agent reaches some terminal state. The value and action-value functions satisfy the Bellman recursions exactly as in Equations (2.4) and (2.5), respectively. The value function for terminal states is set beforehand. Shortest path problems where there is a goal, terminal state can be modeled with this setting by letting $v^*(g) = 0$ where $g \in \mathcal{T}$ is the goal, terminal state, and $\mathcal{R}(s,a,s') = -1$ for any $(s,a,s') \in \mathcal{S} \times \mathcal{A} \times (\mathcal{S} \cup \mathcal{T})$.

2.1.2 The average-reward case

A better way to model continuing tasks is the average-reward setting. Here, the agent seeks a policy that maximizes

$$\rho^{\pi}(s) = \lim_{T \to \infty} \frac{1}{T} \mathbb{E}_{\pi} \left[\sum_{i=t}^{T} \mathcal{R}(S_i, A_i, S_{i+1}) \right] \ \forall s \in \mathcal{S} \tag{2.17}$$

which is known as the average-reward per step or gain.

This setting is arguably more complex than the discounted one. Commonly, the following assumptions are made to facilitate the design and analysis of algorithms.

Assumption 1. *The MDP $\mathcal{M}$ is communicating [Puterman, 1994]: for each pair of states $s, s' \in \mathcal{S}$, there exists a stationary policy π that has non-zero probability of reaching s' from s.*

Assumption 2. *The MDP $\mathcal{M}$ is unichain [Puterman, 1994]: the transition probability distribution induced by all stationary policies admit a single recurrent class.*

If no restrictions are applied over the MDP, it is possible that the agent is unable to visit all states preventing it of learning an optimal policy over those states. On the contrary, with the previous assumptions we can guarantee that the agent explores all states.

We illustrate the communicating and unichain concepts using the next example.

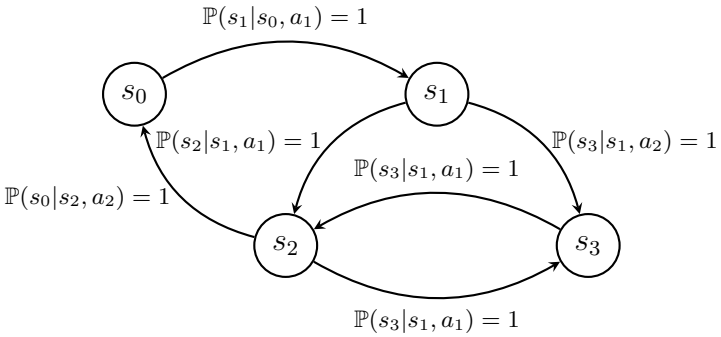


Figure 2.2: A communicating and unichain MDP. $\mathcal{S} \equiv \{s_0, s_1, s_2, s_3\}$ and there are up to 2 actions available in each state. $P_0(s)$ is 1 if $s = s_0$ and 0 otherwise.

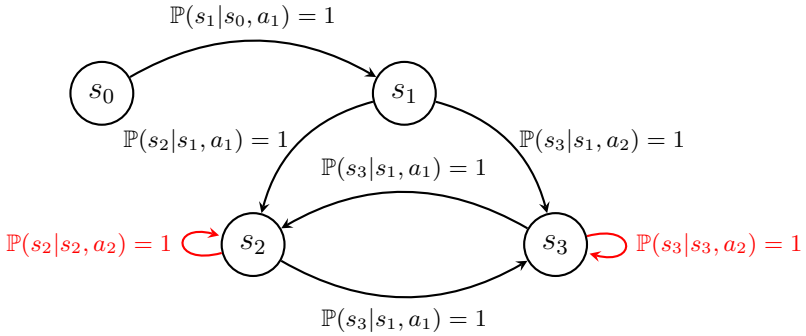


Figure 2.3: Modified version of Figure 2.2 where transition between s_2 and s_0 is removed and self-loop actions are possible at s_2 and s_3 (colored in red).

Example 1. Figure 2.2 shows a communicating and unichain MDP. The MDP is communicating because any stationary policy that assigns non-zero probability to $\pi(a_2|s_2)$

allows the agent to visit any state from any other state in the MDP. A recurrent class is a set of states such that all the states in it communicate exclusively with each other and with no state outside this set. Unichain MDPs are those which for every stationary policy, the policy-induced transition function, contains a single recurrent class, and a (possibly empty) set of transient states (these are states that the agent stops visiting at some point). Now, think about the policy that assigns $\pi(a_1|s_2) = 1$. Such policy contains a single recurrent class that includes states $\{s_2, s_3\}$ (that are visited indefinitely), while $\{s_0, s_1\}$, which are visited just once if we assume that the agent starts at s_0, constitute the set of transient states. On the contrary, Figure 2.3 shows a modified version of the previous in which the resulting MDP is not communicating and not unichain. Clearly, the probability of reaching s_0 from s_2 is zero. Additionally, the policy that chooses $\pi(a_2|s_2) = 1$ and $\pi(a_2|s_3) = 1$ creates two recurrent classes; one formed by s_2 and the other by s_3. In this scenario we say the MDP is multichain.

The aforeintroduced assumptions have some direct consequences that simplify the analysis and algorithms: the value functions are well-defined and that the gain of any policy does not depend on the state, thus, $\rho^\pi(s) = \rho^\pi(s') = \rho^\pi$. We use the latter term to denote the gain of the policy π.

In the absence of a discount factor, the infinite sum of future rewards might now be unbounded, and so, the value functions. To tackle this, the value and action-value functions are now average-regularized by subtracting the gain. Therefore, the value and action-value functions satisfy the following Bellman equations:

$$v^\pi(s) = \sum_a \pi(a|s)\left(\mathcal{R}(s,a) - \rho^\pi + \gamma \sum_{s'} \mathbb{P}(s'|s,a)v^\pi(s')\right) \; \forall s \in \mathcal{S}. \tag{2.18}$$

$$q^\pi(s,a) = \mathcal{R}(s,a) - \rho^\pi + \gamma \sum_{s'} \mathbb{P}(s'|s,a)v^\pi(s') \; \forall (s,a) \in \mathcal{S} \times \mathcal{A}. \tag{2.19}$$

And the following Bellman optimality equations:

$$v^*(s) = \sum_a \pi(a|s)\left(\mathcal{R}(s,a) - \rho^* + \gamma \sum_{s'} \mathbb{P}(s'|s,a)v^\pi(s')\right) \; \forall s \in \mathcal{S}. \tag{2.20}$$

$$q^*(s,a) = \mathcal{R}(s,a) - \rho^* + \gamma \sum_{s'} \mathbb{P}(s'|s,a)v^*(s') \; \forall (s,a) \in \mathcal{S} \times \mathcal{A}. \tag{2.21}$$

The Bellman equations and the Bellman optimality equations are related by the expressions given in (2.6) and (2.7). The goal of the agent is to find a policy π^* for that attains the optimal gain ρ^*.

We redefine the Bellman operators for the value function:

$$(T^\pi v^\pi)(s) = \sum_a \pi(a|s)\left(\mathcal{R}(s,a) + \sum_{s'} \mathbb{P}(s'|s,a)v^\pi(s')\right) \ \forall s \in \mathcal{S}. \tag{2.22}$$

$$(T^* v^*)(s) = \max_a \sum_{s'} \mathcal{R}(s,a) + \sum_{s'} \mathbb{P}(s'|s,a)v^*(s') \ \forall s \in \mathcal{S}. \tag{2.23}$$

We can use *value iteration* as-is (Equation 2.12) to compute the optimal value function with the newly defined operators. Unfortunately, the values can grow very large. *Relative value iteration* [White, 1963] can be used instead. The idea is to have some reference state s^* whose value is subtracted in every iteration. This modified version of value iteration can be expressed in vector form as

$$v_{k+1} \leftarrow T^* v_k - T^* v_k(s^*). \tag{2.24}$$

Even though neither of the versions of value iteration computes the optimal gain, this can be computed as

$$\rho^* = v_{k+1}(s) - v_k(s)$$

when $k \to \infty$.

Alternatively, the policy iteration scheme previously described can be used to compute the optimal policy in the average-reward case.

2.2 Reinforcement learning

Reinforcement learning (RL) proposes a learning paradigm when the agent is alien to the reward function $\mathcal{R}$ and the transition probability distribution $\mathbb{P}$ of the MDP. In this context, the learning happens through direct interaction with the environment. There is a vast collection of RL algorithms, but besides the intricacies of each method, most of them are built upon the same simple ideas. For the purpose of this dissertation we will limit ourselves to value-based (also known as critic-only in the literature) approaches. These algorithms maintain some estimates of the value function that are updated online using some update rule.

We review the update rules used by different algorithms in the discounted and average-reward settings in the ensuing sections.

2.2.1 The discounted setting

Sutton [1988] introduces the idea of temporal difference (TD) learning that uses bootstrapping (using predicted values as targets) during learning. The agent interacts with

the environment following policy π that is kept fixed. The interaction produces samples $(s_t, a_t, r_{t+1}, s_{t+1})$ that are used to learn the value function of the given policy in an online manner. The agent maintains an estimate $\widehat{v}$ of v^π. The update rule of the so-called TD(0) algorithm is

$$\delta_t = r_{t+1} + \gamma \widehat{v}(s_{t+1}) - \widehat{v}(s_t) \tag{2.25}$$

$$\widehat{v}(s_t) \leftarrow \widehat{v}(s_t) + \alpha_t \delta_t, \tag{2.26}$$

where the term δ_t is called the TD error and α_t is the learning rate. This algorithm is said to be on-policy because the policy that is target of the learning is the one used to interact with the environment.

We can also learn the optimal policy by directly estimating the optimal action-value function using the temporal difference technique. This is the idea of the probably most canonical algorithm in RL, called Q-learning [Watkins and Dayan, 1992]. In this case, the agent keeps an estimate $\widehat{q}$ of the optimal action-value function q^*, and its values are updated using the following scheme:

$$\delta_t = r_{t+1} + \gamma \max_a \widehat{q}(s_{t+1}, a) - \widehat{q}(s_t, a_t) \tag{2.27}$$

$$\widehat{q}(s_t, a_t) \leftarrow \widehat{q}(s_t, a_t) + \alpha_t \delta_t \tag{2.28}$$

During learning the agent follows an ϵ-greedy policy which with probability $1 - \epsilon$ selects the greedy action $(\mathcal{G})(s)$, otherwise selects a random exploratory action. Nonetheless, the target policy is a greedy, deterministic policy over the estimate of the optimal action-value function. Hence, Q-learning is an off-policy method: the policy that is target of the learning and the one used to act in the environment are different.

2.2.2 The average-reward setting

The TD learning scheme can also be applied in the average-reward setting. Mahadevan [1996] gives the average-reward counterpart of the Q-learning algorithm. Now, in addition to maintaining the action-value estimate, we also need to update an estimate $\widehat{\rho}$ of the optimal gain ρ^*. The update rules are as follows

$$\delta_t = r_{t+1} - \widehat{\rho} + \max_a \widehat{q}(s_{t+1}, a) - \widehat{q}(s_t, a_t) \tag{2.29}$$

$$\widehat{q}(s_t, a_t) \leftarrow \widehat{q}(s_t, a_t) + \alpha_t \delta_t \tag{2.30}$$

$$\widehat{\rho} \leftarrow \widehat{\rho} + \beta_t \big(r_t - \widehat{\rho} + \max_a \widehat{q}(s_{t+1}, a) - \max_b \widehat{q}(s_t, b) \big). \tag{2.31}$$

α_t and β_t are different learning rates that can be updated independently. The estimate of the gain $\widehat{\rho}$ is only updated when a non-exploratory action is executed.

2.3 Successor features

Successor features (SF) [Barreto et al., 2017] is a widely used RL representation framework that assumes the reward function is linearly expressible with respect to a feature vector,

$$\mathcal{R}^{\mathbf{w}}(s, a, s') = \mathbf{w}^\intercal \boldsymbol{\phi}(s, a, s'). \tag{2.32}$$

Here, $\boldsymbol{\phi} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}^d$ maps transitions to feature vectors and $\mathbf{w} \in \mathbb{R}^d$ is a weight vector. Every weight vector $\mathbf{w}$ induces a different reward function and, therefore, a task.

Following the definition of the action-value function in Equation (2.3), and considering the reward structure introduced in (2.32), the action-value function of a state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$ under policy π can be rewritten as

$$\begin{aligned} q^\pi_{\mathbf{w}}(s, a) &= \mathbb{E}_\pi \left[\sum_{i=t}^{\infty} \gamma^{i-t} \mathbf{w}^\intercal \boldsymbol{\phi}_i \;\middle|\; S_t = s, A_t = a \right] \\ &= \mathbf{w}^\intercal \mathbb{E}_\pi \left[\sum_{i=t}^{\infty} \gamma^{i-t} \boldsymbol{\phi}_i \;\middle|\; S_t = s, A_t = a \right] \\ &= \mathbf{w}^\intercal \boldsymbol{\psi}^\pi(s, a) \end{aligned} \tag{2.33}$$

where $\boldsymbol{\phi}_i = \boldsymbol{\phi}(S_i, A_i, S_{i+1})$, and the term

$$\boldsymbol{\psi}^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{i=t}^{\infty} \gamma^{i-t} \boldsymbol{\phi}_i \;\middle|\; S_t = s, A_t = a \right] \tag{2.34}$$

constitutes the SF vector of state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$ under a policy π, and it represents the expected discounted sum of future feature vectors when following policy π. The SF vector satisfies a type of Bellman equation

$$\begin{aligned} \boldsymbol{\psi}^\pi(s, a) &= \mathbb{E}_\pi \left[\sum_{i=t}^{\infty} \gamma^{i-t} \boldsymbol{\phi}_i \;\middle|\; S_t = s, A_t = a \right] \\ &= \mathbb{E}_\pi \left[\boldsymbol{\phi}(s, a, S_{t+1}) + \sum_{i=t+1}^{\infty} \gamma^{i-t+1} \boldsymbol{\phi}_i \;\middle|\; S_t = s, A_t = a \right] \\ &= \mathbb{P}(s'|s, a) \left(\boldsymbol{\phi}(s, a, s') + \gamma \mathbb{E}_\pi \left[\sum_{i=t}^{\infty} \gamma^{i-t} \boldsymbol{\phi}_i \;\middle|\; S_t = s' \right] \right) \end{aligned}$$

$$= \boldsymbol{\phi}(s,a) + \gamma \mathbb{P}(s'|s,a)\mathbb{E}_\pi \left[\sum_{i=t}^{\infty} \gamma^{i-t} \boldsymbol{\phi}_i \;\middle|\; S_t = s' \right]$$

$$\boldsymbol{\psi}^\pi(s,a) = \boldsymbol{\phi}(s,a) + \gamma \sum_{a'} \pi(a'|s)\mathbb{P}(s'|s,a')\boldsymbol{\phi}(s',a'),$$

where we use $\boldsymbol{\phi}^\pi(s,a) = \sum_{s'} \mathbb{P}(s'|a,s)\boldsymbol{\phi}(s,a,s')$. Therefore, it can be learned with any off-the-shelf RL algorithm such as Q-learning.

SF is a generalization of the successor representation (SR) framework [Dayan, 1993] where the feature map $\boldsymbol{\phi}$ is the one-hot encoding of the state space. In this case, SR vector of each state represents a discounted distribution over future next states under policy π.

The action value function for a state-action pair (s,a) under policy π can be efficiently represented using the SF vector. Due to the linearity of the reward function, the weight vector can be decoupled from the Bellman recursion. The SF representation leads to *generalized policy evaluation* (GPE) over multiple tasks [Barreto et al., 2020], and similarly, to *generalized policy improvement* (GPI) to obtain new better policies [Barreto et al., 2017].

2.3.1 Combining policies in SF

A family of MDPs is defined as the set of MDPs that share all the components, except the reward function. This set is formally defined as

$$\mathcal{M}^\phi \equiv \{\langle \mathcal{S}, \mathcal{E}, \mathcal{A}, \mathcal{R}_\mathbf{w}, \mathbb{P}_0, \mathbb{P}\rangle | \mathcal{R}_\mathbf{w} = \mathbf{w}^\intercal \boldsymbol{\phi}, \forall \mathbf{w} \in \mathbb{R}^d\}.$$

Transfer learning on families of MDPs is possible thanks to GPI. Given a set of policies Π, learned on the same family $\mathcal{M}^\phi$, for which their respective SF representations have been computed, and a new task $\mathbf{w}' \in \mathbb{R}^d$, a GPI policy π_{GPI} for any $s \in \mathcal{S}$ is derived as

$$\pi_{\text{GPI}}(s) \in \arg\max_{a \in \mathcal{A}} \max_{\pi \in \Pi} q^\pi_{\mathbf{w}'}(s,a). \tag{2.35}$$

However, there is no guarantee of optimality for $\mathbf{w}'$. A fundamental question to solve the so-called *optimal policy transfer learning problem* [Alegre et al., 2022] is which policies should be included in the set of policies Π so an optimal policy for any weight vector $\mathbf{w} \in \mathbb{R}^d$ can be obtained with GPI.

2.4 Linearly-solvable Markov decision processes

Linearly-solvable Markov decision processes (LMDPs) [Todorov, 2006, Kappen, 2005] are a restricted class of the more general MDPs where the Bellman optimality equations are linear. This makes the computation of optimal value functions more efficient. In continuous state space domains or contexts of optimal control as probabilistic inference, they frequently appear under the names of path-integral or Kullback-Leibler control [Kappen et al., 2012].

Even though this formulation is arguably restricted, the intuition of entropy-regularization [Neu et al., 2017], that lies at the core of LMDPs, is fundamental in RL as it is one of the building blocks of current state-of-the-art deep reinforcement learning algorithms such as trust region policy optimization [Schulman et al., 2015], soft actor-critic (SAC) [Haarnoja et al., 2018b] or Munchausen RL [Vieillard et al., 2020]. In what follows we introduce linearly-solvable Markov decision processes in the finite-horizon (first-exit) setting, its extension to the infinite-horizon (average-reward) setting and we briefly discuss how they can be used along with function approximation.

2.4.1 First-exit linearly-solvable Markov decision processes

We define a first-exit LMDP as a tuple $\mathcal{L} = \langle \mathcal{S}, \mathcal{T}, \mathbb{P}, \mathcal{R}, \mathcal{J} \rangle$, where:

- $\mathcal{S}$ is a set of non-terminal states.
- $\mathcal{T}$ is a set of terminal states.
- $\mathbb{P} : \mathcal{S} \to \Delta(\mathcal{S}^+)$ is an uncontrolled transition function, also known as passive dynamics or passive controls.
- $\mathcal{R} : \mathcal{S} \to \mathbb{R}$ is a reward function for non-terminal states.
- $\mathcal{J} : \mathcal{T} \to \mathbb{R}$ is a reward function for terminal states.

We let $\mathcal{S}^+ = \mathcal{S} \cup \mathcal{T}$ denote the full set of states and $B = \max_{s \in \mathcal{S}} |\mathcal{B}(\mathbb{P}(\cdot|s))|$ an upper bound on the support of $\mathbb{P}$.

Similarly to the standard RL learning loop, the agent interacts with the environment in a sequential manner. Nonetheless, there are no explicit actions, and now the learning agent follows a policy $\pi : \mathcal{S} \to \Delta(\mathcal{S}^+)$. Such a policy chooses, for each non-terminal state $s \in \mathcal{S}$, a probability distribution over next states in the support of $\mathbb{P}(\cdot|s)$, i.e. $\pi(\cdot|s) \in \Delta(\mathcal{B}(\mathbb{P}(\cdot|s))$. There is also no explicit mention of the initial state distribution, which is assumed to be uniform over the set of non-terminal states (this is $\mathbb{P}_0 = \text{unif}(\mathcal{S})$).

At each timestep t, the learning agent observes a state $s_t \in \mathcal{S}^+$. If s_t is non-terminal, the agent transitions to a new state $s_{t+1} \sim \pi(\cdot|s_t)$ and receives an immediate, regularized reward

$$\mathcal{R}_\eta(s_t, s_{t+1}, \pi) = \mathcal{R}(s_t) - \frac{1}{\eta} \log \frac{\pi(s_{t+1}|s_t)}{\mathbb{P}(s_{t+1}|s_t)},$$

where $\mathcal{R}(s_t)$ is the reward associated with state s_t, and η is a temperature parameter.

Hence the agent can set the probability $\pi(s_{t+1}|s_t)$ freely, but gets penalized by the entropy-regularization term for deviating from the passive controls $\mathbb{P}(s_{t+1}|s_t)$, and this penalty is modulated by the temperature parameter η. On the other hand, if s_t is terminal, the agent receives reward $\mathcal{J}(s_t)$ and then the current episode ends. The aim of the agent is to compute a policy π that maximizes the expected future *total reward*. For each non-terminal state $s \in \mathcal{S}$, the value function is defined as

$$v_\eta^\pi(s) = \mathbb{E}\left[\sum_{i=t}^{T-1} \mathcal{R}_\eta(S_i, S_{i+1}, \pi) + \mathcal{J}(S_T) \;\middle|\; S_t = s, \pi\right].$$

Here, T is a random variable representing the time at which the current episode ends, and S_t is a random variable representing the state at time t. The expectation is over the stochastic choice of next state $S_{t+1} \sim \pi(\cdot|S_t)$ at each time t, and the time T it takes for the episode to end. It is assumed that the reward of all non-terminal states is negative, i.e. $\mathcal{R}(s) < 0$ for each $s \in \mathcal{S}$. As a consequence, $\mathcal{R}(s, \pi) < 0$ holds for any policy π, and the value $v_\eta^\pi(s)$ has a well-defined upper bound. Note that the value function is computed with respect to a concrete value of the temperature parameter η.

We are interested in finding the optimal policy which implies computing the optimal value function $v_\eta^* : \mathcal{S} \to \mathbb{R}$, i.e. the maximum expected future total reward among all policies. The value function is extended to each terminal state $\tau \in \mathcal{T}$ by defining $v_\eta^*(\tau) \equiv \mathcal{J}(\tau)$. The value function v_η^* satisfies the Bellman optimality equations:

$$\begin{aligned} \eta v_\eta^*(s) &= \eta \max_\pi \left[\mathcal{R}(s, \pi) + \mathbb{E}_{s' \sim \pi(\cdot|s)} v_\eta^*(s')\right] \\ &= \eta \mathcal{R}(s) + \max_\pi \mathbb{E}_{s' \sim \pi(\cdot|s)} \left[\eta v_\eta^*(s') - \log \frac{\pi(s'|s)}{\mathbb{P}(s'|s)}\right] \quad \forall s. \end{aligned}$$

In expectation, the regularization term in reduces to the Kullback-Liebler divergence between π and $\mathbb{P}$,

$$\mathbb{E}_{s' \sim \pi(\cdot|s)} \log \frac{\pi(s'|s)}{\mathbb{P}(s'|s)} = \sum_{s'} \mathbb{P}(s'|s) \log \frac{\pi(s'|s)}{\mathbb{P}(s'|s)} = \mathrm{KL}\big(\pi(\cdot|s) \| \, \mathbb{P}(\cdot|s)\big).$$

The maximization in the Bellman equations can be resolved analytically [Todorov, 2006],

and the optimal value function can be rewritten as

$$\begin{aligned} v_\eta^*(s) &= \frac{1}{\eta} \log \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s) e^{\eta(\mathcal{R}(s) + v_\eta^*(s'))} \quad \forall s \in \mathcal{S}, \\ &= \mathcal{R}(s) + \frac{1}{\eta} \log \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s) e^{\eta v_\eta^*(s')} \quad \forall s \in \mathcal{S}, \\ \eta v_\eta^*(s) &= \eta \mathcal{R}(s) + \log \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s) e^{\eta v_\eta^*(s')} \quad \forall s \in \mathcal{S}. \end{aligned}$$

Now, we introduce the notation $z_\eta(s) = e^{\eta v_\eta^*(s)}$ for each $s \in \mathcal{S}^+$. We often abuse this notation by referring to $z_\eta(s)$ as the (optimal) value of s and using z instead of z_η when the context is clear. After exponentiating, the previous system yields the following Bellman optimality equations that are linear in z:

$$z(s) = e^{\eta \mathcal{R}(s)} \sum_{s'} \mathbb{P}(s'|s) z(s'). \tag{2.36}$$

Solving a first-exit LMDP

The Bellman equation can be expressed in matrix form by defining a $|\mathcal{S}| \times |\mathcal{S}|$ diagonal reward matrix $R = \text{diag}(e^{\eta \mathcal{R}(\cdot)})$ and a $|\mathcal{S}| \times |\mathcal{S}^+|$ stochastic transition matrix P whose entries (s, s') equal $\mathbb{P}(s'|s)$. We define a vector $\mathbf{z}$ that stores the values $z(s)$ for each non-terminal state $s \in \mathcal{S}$, and a vector $\mathbf{z}^+$ extended to all states in $\mathcal{S}^+$. The extended vector$\mathbf{z}^+$ contains the values of terminal states, which are known due to $\mathcal{J}$. Now we can write the Bellman equations in matrix form as:

$$\mathbf{z} = RP\mathbf{z}^+. \tag{2.37}$$

Given z, the optimal policy π is given by the following expression for each pair of states (s, s'):

$$\pi(s'|s) = \frac{\mathbb{P}(s'|s) e^{\eta v(s')}}{\sum_{s''} \mathbb{P}(s''|s) e^{\eta v(s'')}} = \frac{\mathbb{P}(s'|s) z(s')}{\sum_{s''} \mathbb{P}(s''|s) z(s'')}. \tag{2.38}$$

The solution for z corresponds to the largest eigenvector of RP. If the dynamics $\mathbb{P}$ and $\mathcal{R}$ are known, we can apply the power iteration method to Equation (2.37) [Todorov, 2006].

Alternatively, we can learn an estimate $\widehat{z}$ incrementally using stochastic updates based on state transitions sampled from the uncontrolled dynamics (s_t, r_t, s_{t+1}) with the following TD update rule

$$\widehat{z}(s_t) \leftarrow (1 - \alpha_t)\widehat{z}(s_t) + \alpha_t e^{\eta r_t} \widehat{z}(s_{t+1}),$$

where α_t is a learning rate. The previous update rule is called *Z-learning* [Todorov, 2006] and suffers from slow convergence in very large state spaces and when the optimal policy differs substantially from the uncontrolled dynamics $\mathbb{P}$. A better choice is importance sampling, which uses samples from the estimated policy $\widehat{\pi}$ derived from the estimated values $\widehat{z}$ and (2.38) and updates $\widehat{z}$ according to the following update

$$\widehat{z}(s_t) \leftarrow (1-\alpha_t)\widehat{z}(s_t) + \alpha_t e^{\eta r_t}\widehat{z}(s_{t+1})\frac{\mathbb{P}(s_{t+1}|s_t)}{\widehat{\pi}(s_{t+1}|s_t)}. \tag{2.39}$$

However, this requires local knowledge of $\mathbb{P}(\cdot|s_t)$ to correct for the different sampling distribution. Though this seems like a strong assumption, in practice $\mathbb{P}$ usually has a simple form, e.g. uniform distribution. Further, as shown in Jonsson and Gómez [2016], the corrected update rule in (2.39) can also be used to perform off-policy updates in case transitions are sampled using a policy different from $\widehat{\pi}$.

2.4.2 Compositionality

Todorov [2009b] introduces the concept of compositionality for LMDPs. Let us consider a set of LMDPs $\{\mathcal{L}_1, \ldots, \mathcal{L}_n\}$, where each LMDP $\mathcal{L}_i = \langle \mathcal{S}, \mathcal{T}, \mathbb{P}, \mathcal{R}, \mathcal{J}_i \rangle$ has the same components $\mathcal{S}, \mathcal{T}, \mathbb{P}, \mathcal{R}$ and only differ in the reward $\mathcal{J}_i(\tau)$ of each terminal state $\tau \in \mathcal{T}$, as well as its exponentiated value $z_i(\tau) = e^{\eta \mathcal{J}_i(\tau)}$.

Now consider a new LMDP $\mathcal{L} = \langle \mathcal{S}, \mathcal{T}, \mathbb{P}, \mathcal{R}, \mathcal{J} \rangle$ with the same components as the n LMDPs above, except for $\mathcal{J}$. Assume that there exist weights $w_1, \ldots, w_n$ such that the exponentiated value of each terminal state $\tau \in \mathcal{T}$ can be written as

$$e^{\eta \mathcal{J}(\tau)} = z(\tau) = w_1 z_1(\tau) + \ldots + w_n z_n(\tau) = \sum_{i=1}^{n} w_i z_i(\tau).$$

Since the Bellman optimality equation of each non-terminal state $s \in \mathcal{S}$ is linear in z, the optimal value of s satisfies the same equation:

$$z(s) = \sum_{i=1}^{n} w_i z_i(s).$$

Consequently, if the optimal values $z_1, \ldots, z_n$ of the n LMDPs are previously computed and the weights $w_1, \ldots, w_n$ are known, we immediately obtains the optimal values of the new LMDP $\mathcal{L}$ without further learning.

2.4.3 Average-reward linearly-solvable Markov decision processes

LMDPs can be extended to the average-reward setting. We say an average-reward Linearly-solvable Markov decision process (ALMDP) is a tuple $\mathcal{L} = \langle \mathcal{S}, \mathbb{P}, \mathcal{R} \rangle$, where $\mathcal{S}$ is a

set of states, $\mathbb{P} : \mathcal{S} \to \Delta(\mathcal{S})$ is the passive dynamics, and $\mathcal{R}$ is the reward function. ALMDPs represent continuing tasks and, unlike first-exit LMDPs, there are no terminal states. Consequently, there is no reward function for terminal states.

We restate Assumptions 1 and 2 for the case of LMDPs.

Assumption 3. *The ALMDP $\mathcal{L}$ is communicating [Puterman, 1994]: for each pair of states $s, s' \in \mathcal{S}$, there exists a policy π that has non-zero probability of reaching s' from s.*

Assumption 4. *The ALMDP $\mathcal{L}$ is unichain [Puterman, 1994]: the transition probability distribution induced by all stationary policies admit a single recurrent class.*

In the average-reward setting, the value function is defined as the expected average reward when following a policy $\pi : \mathcal{S} \to \Delta(\mathcal{S})$ starting from a state $s \in \mathcal{S}$. This is expressed as

$$v_\eta^\pi(s) = \lim_{T\to\infty} \mathbb{E}\left[\frac{1}{T}\sum_{i=t}^{T} \mathcal{R}_\eta(S_i, S_{i+1}, \pi) \;\middle|\; S_t = s, \pi\right], \tag{2.40}$$

where $\mathcal{R}_\eta(s_t, s_{t+1}, \pi)$ is defined as for first-exit LMDPs. Again, the goal is to obtain the optimal value function v_η^*. Under Assumption 4, the Bellman optimality equations can be written as

$$v_\eta^*(s) = \frac{1}{\eta}\log \sum_{s'\in\mathcal{S}} \mathbb{P}(s'|s) e^{\eta(\mathcal{R}(s)-\rho+v_\eta^*(s'))} \quad \forall s \in \mathcal{S}, \tag{2.41}$$

where ρ is the optimal one-step average reward (i.e. gain), which is state-independent for unichain ALMDPs [Todorov, 2006]. Exponentiating yields

$$z(s) = e^{\eta(\mathcal{R}(s)-\rho)} \sum_{s'\in\mathcal{S}} \mathbb{P}(s'|s) z(s') \quad \forall s \in \mathcal{S}. \tag{2.42}$$

For the optimal value function z, the optimal policy is given by the same expression as in (2.38).

Solving an ALMDP

We let $\Gamma = e^{\eta\rho}$ denote the exponentiated gain. Similar to the first-exit case, we can express Equation (2.42) in matrix form as

$$\Gamma \mathbf{z} = RP\mathbf{z}, \tag{2.43}$$

where the matrices $P \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ and $R \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ are appropriately defined as in (2.37). The exponentiated gain Γ can be shown to correspond to the largest eigenvalue of

RP [Todorov, 2009a]. An ALMDP can be solved using *relative value iteration* by selecting a reference state $s^* \in \mathcal{S}$, initializing $\widehat{\mathbf{z}}_0 = \mathbf{1}$ and iteratively applying

$$\widehat{\mathbf{z}}_{k+\frac{1}{2}} \leftarrow RP\widehat{\mathbf{z}}_k, \qquad \widehat{\mathbf{z}}_{k+1} \leftarrow \widehat{\mathbf{z}}_{k+\frac{1}{2}} / \widehat{z}_{k+\frac{1}{2}}(s^*).$$

The reference state s^* satisfies $z(s^*) = 1$, which makes the optimal value z unique (else any constant shift preserves optimality). After convergence, the exponentiated gain equals $\Gamma = \widehat{z}_{k+\frac{1}{2}}(s^*)$. Under Assumption 3, relative value iteration converges to the unique optimal value z [Todorov, 2009a].

Analogously to first-exit LMDPs, when $\mathbb{P}$ and $\mathcal{R}$ are not known, the agent can learn estimates $\widehat{z}$ and $\widehat{\Gamma}$ of the optimal value function and the exponentiated gain in an online manner, using samples (s_t, r_t, s_{t+1}) generated when following the estimated policy $\widehat{\pi}$. The update rules for the so-called *differential Z-learning* algorithm are given by

Move it to the chapter?

$$\widehat{z}(s_t) \leftarrow \widehat{z}(s_t) + \alpha_t \left(\frac{e^{\eta r_t}}{\widehat{\Gamma}_t} \frac{\mathbb{P}(s_{t+1}|s_t)}{\widehat{\pi}_t(s_{t+1}|s_t)} - \widehat{z}(s_t) \right), \tag{2.44}$$

$$\widehat{\Gamma}_{t+1} \leftarrow \widehat{\Gamma}_t + \beta_t \left(\frac{e^{\eta r_t}}{\widehat{z}_t(s_t)} \frac{\mathbb{P}(s_{t+1}|s_t)}{\widehat{\pi}_t(s_{t+1}|s_t)} - \widehat{\Gamma}_t \right). \tag{2.45}$$

The learning rates α_t and β_t can be chosen independently.

Unlike the first-exit case, the compositionality property does not hold in the average-reward case.

2.4.4 Function approximation in LMDPs

So far, we have introduced methods for solving (A)LMDPs in the tabular case, where the full state space can be enumerated and stored in memory. However, in cases where the state space is considerably large, tabular methods are unfeasible, and the value function is approximated.

In this line, Todorov [2010] prescribes two families of methods for adapting LMDPs in the average-reward setting to function approximation. The first one lies in the family of policy gradients while the second ones can be considered critic-only approaches. Here a valid parameterization of the policy $\pi(s'|s, \mathbf{w})$ is considered.

The Bellman equation (2.41) can be rewritten as follows:

$$\rho + v^*(s) = \mathcal{R}(s) + \log \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s) e^{v^*(s')} \quad \forall s \in \mathcal{S}, \tag{2.46}$$

where for the sake of simplicity we assume that $\eta = 1$.

The first flavor of methods builds on the policy gradient theorem for ALMDPs [Todorov, 2010, cf. Theorem 1] that states that the gradient of the gain in ALMDPs is

$$\nabla_{\mathbf{w}}\rho = \sum_{s} \mu(s, \mathbf{w}) \sum_{s'} \nabla_{\mathbf{w}}\pi(s'|s, \mathbf{w}) \Big(\log \frac{\pi(s'|s, \mathbf{w})}{\mathbb{P}(s'|s)} + \widetilde{v}(s', \mathbf{r}) \Big), \tag{2.47}$$

where $\mu(x, \mathbf{w})$ is the stationary distribution induced by π, and $\widetilde{v}(s, \mathbf{r})$ is an approximation to the (optimal) value function with parameterization $\mathbf{r}$.

For the second type of methods one could use *approximate* value iteration or *approximate* policy iteration to iteratively improve the weight vector $\mathbf{w}$, even though this results in a biased estimator. An even more direct approach is to use Gauss-Newton method to fit the weight vector $\mathbf{w}$ by directly optimizing (2.46).

Despite the correctness of the theoretical results, which guarantee that function approximation can be used along with LMDPs, they are of little practical use. Even in the linear function approximation (LFA) case, the policy gradient approach is intricate and unusual. To find the parameterization $\widetilde{v}(s, \mathbf{r})$, a previous step of finding a projection $\widetilde{v}(s, \mathbf{s})$ of v^* using a different set of policy-dependent features is required. The process is not as direct as in traditional MDPs and, additionally, the representation of the policy still depends on the size of the whole state space. This makes the approach intractable for real-world problems as it does not scale with large state spaces.

2.5 Hierarchical reinforcement learning

Hierarchical reinforcement learning (HRL) embodies the idea of divide-and-conquer paradigm in sequential decision problems. This algorithmic paradigm hypothesizes that breaking down a problem into smaller (sub)problems should facilitate its solution, by solving the small problems in isolation and then combining them to give an overall solution. But the philosophical motivations go beyond this as human thinking process happens at different timescales while at the same time we can use the same piece of knowledge to solve different, but similar tasks. Think, for example, about cooking a recipe. This human activity involves different steps than can be solved in isolation such as shopping the ingredients, cutting and chopping vegetables, using the stove or placing the final outcome in plates. A specific sequence of these steps with certain ingredients leads to a unique result, though many of these steps can transfer from recipe to recipe.

We review how the modelling of these (sub)problems are commonly addressed in the context of RL and the issues that they usually entail regarding the optimality of the final solution.

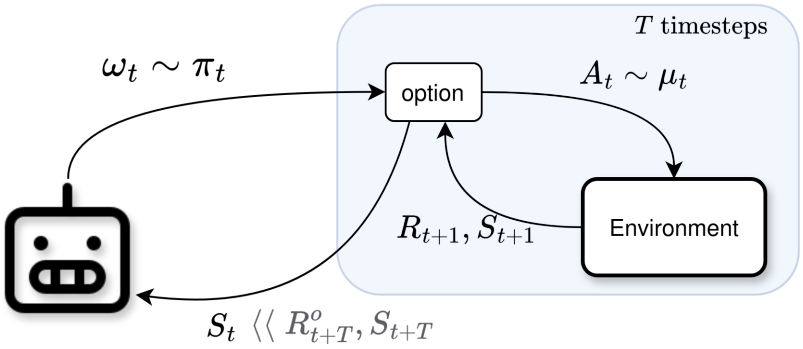


Figure 2.4: The interaction loop in a semi MDP where the underlying MDP is extended with options.

2.5.1 The options framework

A common way to model (sub)problems[1] in RL is by using the options framework. A Markovian option represents a temporally-extended action and is a tuple $\omega = \langle \mathcal{I}, \mu, f \rangle$ where:

- $\mathcal{I} \subseteq \mathcal{S}$ is the initiation set, i.e. all the states in which the option ω can be called.
- $\mu : \mathcal{S} \to \mathcal{A}$ is the option's policy.
- $f : \mathcal{S} \to \Delta(\{0, 1\})$ is a termination function.

The combination of an MDP $\mathcal{S}$ with a fixed set of options Ω results in a semi-MDP $\widetilde{\mathcal{M}} = \langle \mathcal{S}, \Omega, \mathcal{R}, \mathbb{P}, \mathbb{P}_0 \rangle$, and now, unlike the regular MDP formulation, the agent learns a policy over the space of options $\pi : \mathcal{S} \to \Delta(\Omega)$. The interaction is now slightly modified (see Figure 2.4). At timestep s_t, the agent selects an option according to policy π conditioned on that option being available at that state(i.e. $s_t \in \mathcal{I}_\omega$). Such an option that takes control for period of time and that is run until termination. After T timesteps, where T is a random variable representing duration of the option, the control is returned to the agent which then can choose a new option. Note that primitive actions can be considered a special case of options that terminate after one timestep.

We have only described Markovian options. There are cases in which we might want to include a timeout so that the options terminate surely after a certain number of steps. In that case, the options' policies should not only consider the current state, but the history

[1] I use the nomenclature (sub)problem with the prefix sub in parentheses to highlight that, even though these problems are part of a bigger one, they are well-defined problems that can be solved in isolation.

since the option was initialized. We do not go in detail as they these type of options are out of the scope of this dissertation.

In order to write the Bellman equations for the semi MDP formulation, we need to make use of the reward and transition multi-time model [Sutton et al., 1999] induced by the option. These are defined as

$$\mathcal{R}(s,\omega) = \mathbb{E}_{\mu}\left[\sum_{i=t}^{t+T} R_i | S_t = s\right], \tag{2.48}$$

$$\mathbb{P}(s'|s,\omega) = \sum_{k=0} \gamma^k \Pr(s_k = s'|s,\omega), \tag{2.49}$$

where $\Pr(s_k = s'|s,\omega)$ represents the probability that option ω terminates in state $s' \in \mathcal{S}$ in exactly k timesteps according to the underlying MDP transition function. Using this extended model, we can write the counterparts of Bellman equations for the value function

$$v^{\pi}(s) = \sum_{\omega\in\Omega} \pi(\omega|s)\left(\mathcal{R}(s,\omega) + \sum_{s'} \mathbb{P}(s'|s,\omega) v^{\pi}(s')\right), \tag{2.50}$$

and the Bellman equations for the action-value function

$$q^{\pi}(s,\omega) = \mathcal{R}(s,\omega) + \sum_{s'} \mathbb{P}(s'|s,\omega) v^{\pi}(s'). \tag{2.51}$$

By defining the proper Bellman operators, with little modification we can use the dynamic-programming algorithms discussed in section 2.1.1.

Similarly, we can learn the optimal policy by letting the agent interact directly with the environment. In this case, the update rule for the Q-learning counterpart with options is

$$\delta_t = r_t^o + \gamma^k \max_{\omega} q(s_{t+k},\omega) - q(s_t,\omega_t), \tag{2.52}$$

$$\widehat{q}(s_t,\omega_t) \leftarrow \widehat{q}(s_t,\omega_t) + \alpha_t \delta_t. \tag{2.53}$$

Here, $r_t^o = \sum_{i=t}^{t+k} \gamma^{i-t} R_i$ is the cumulative discounted reward achieved by the option, and k is the number of timesteps the option takes to terminate.

Intra-task learning

Up to this point, the options have been assumed to be learned before computing the optimal value functions. We can also learn the options' models on the fly while interacting online with the environment. In addition, any sample $(s_t, a_t, r_{t+1}, s_{t+1})$ obtained by executing one option can be used to update the policies of other options consistent with that transition.

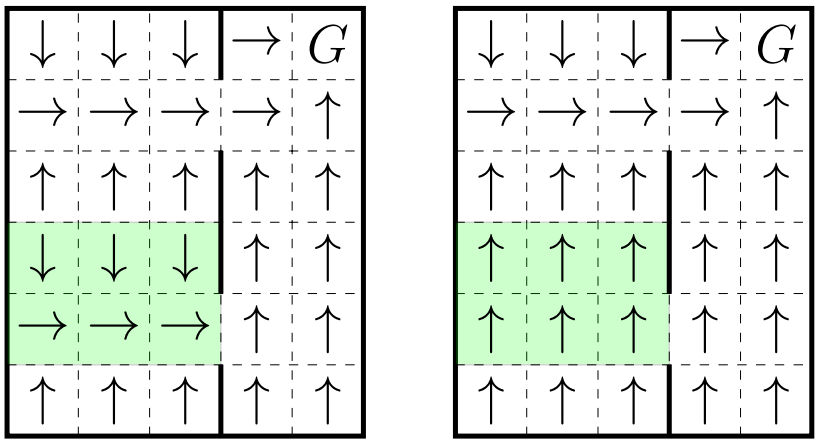


Figure 2.5: Example of a recursively optimal policy (left) and a hierarchically optimal one (right) in a . The area in green highlights the states in which policies disagree.

2.5.2 Optimality of HRL algorithms

Most hierarchical algorithms [Dietterich, 2000, Parr and Russell, 1997] are known to suffer from suboptimality when providing an overall solution by combining the solution for the (sub)problems. Dietterich [2000] distinguishes two types of optimality for hierarchical methods in reinforcement learning, namely *recursive optimality* and *hierarchical optimality*. Recursive optimality refers to hierarchical algorithms that attain locally optimal solution for each subtask according to the high-level. Though the solution to each independent (sub)problem is indeed optimal, the overall solution is not. Hierarchical optimality, on the other hand, is a stronger concept that implies optimality with regard a constrained space of policies, given by the hierarchy structure.

Example 2. We recover the example in [Dietterich, 2000]. Figure the gridworld environment depicted in Figure 2.5. There are two rooms (left and right) separated by a wall with two doors. The primitive actions $\mathcal{A} \equiv \{\texttt{left}, \texttt{up}, \texttt{right}, \texttt{down}\}$, though the agent has only access to two options. The first is `EXIT ROOM` when the agent is placed somewhere in the left room. The agent can execute a second option `GO TO G` when it finds itself in the room on the right.

2.6 Non-Markovian task specification

- Difficulty about expressing some tasks in Markovian terms
- Finite State automata and Reward Machines

- Reward Machines and their relationship with logics
- Comment on algorithms given in the

Chapter 3

Globally Optimal Hierarchical Reinforcement Learning for Linearly-solvable Markov Decision Processes

3.1 Introduction

A major challenge in reinforcement learning is to design agents that are able to learn efficiently and to adapt their existing knowledge to solve new tasks. As it has been discussed, one way to reduce the complexity of learning is hierarchical reinforcement learning [Sutton et al., 1999, Dietterich, 2000, Barto and Mahadevan, 2003]. In this chapter, we propose a novel approach to hierarchical reinforcement learning in LMDPs that takes advantage of the compositionality of LMDPs. This approach assumes that the state space is partitioned into subsets, and the subtasks consist in moving between these partitions. The subtasks are parameterized on the current value estimates of boundary states.

In section 2.4.2, compostionality was shown to be one of the computational advantages of LMDPs, which allows for zero-shot learning of new skills by linearly combining previously learned base skills which only differ in their cost or reward at boundary states [Todorov, 2009a, da Silva et al., 2009]. In this work, instead of solving the subtasks each time the value estimates change, the compositionality property of LMDPs is exploited to express the solution to an arbitrary subtask as a linear combination of a set of base LMDPs. The result is a form of value function decomposition which allows expressing an estimate of the optimal value of an arbitrary state as a combination of

multiple value functions with smaller domains.

In this chapter, we present two novel algorithms. The first is a two-step eigenvector when both the passive dynamics and the reward functions are known. When these are unknown to the learning agent, an online algorithm can be used to simultaneously learn the value function of the subtasks and the value for the boundary states. We accompany the theoretical results with an empirical evaluation of the learning agent on two classic control problems.

3.2 Contributions

- To define a novel scheme based on compositionality for solving subtasks, defining local rewards that constitute a convenient basis for composite rewards.
- The subtask decomposition is at the level of the value function, not of the actual policy. Hence the proposed approach does not suffer from non-stationarity in the online setting, unlike approaches that select among subtasks whose associated policies are being learned.
- Even though the subtasks have local reward functions, under mild assumptions the proposed approach converges to the globally optimal value function.
- The proposed learning algorithm is analyzed empirically, and we show that it is more sample efficient compared to a flat learner and similar hierarchical approaches when the set of boundary states is smaller than the entire state space.

3.3 Related Work

Several authors have recently exploited concurrent compositionality of tasks in the context of transfer learning. van Niekerk et al. [2019] use the linear compositionality of LMDPs to solve new tasks that can be expressed as combinations of a series of existing base tasks. They show that, while disjunctions of base tasks (OR-compositionality) can be performed exactly, the AND composition (when the goals of base tasks partially overlap) can only be performed approximately.

[Haarnoja et al., 2018a] exploit a similar idea to transfer knowledge from existing tasks to new tasks by averaging their reward functions. [Hunt et al., 2019] further extended this by introducing the so-called compositional optimism, and apply divergence correction in case compositionality does not transfer well.

More recently, [Nangue Tasse et al., 2020] derive a formal characterization of union and intersection of tasks in terms of Boolean algebra. They show that learning (extended) value functions that account for all achievable goals, exact zero-shot transfer learning using both AND- and OR- compositionality is possible, achieving an exponential increase in skills compared to the previous works.

All the aforementioned results are derived for general MDPs with deterministic dynamics and, possibly, entropy regularization. This setting is no more general than the class of LMDPs.

Several authors have proposed hierarchical versions of LMDPs. Jonsson and Gómez [2016] extend MAXQ [Dietterich, 2000] to LMDPs by defining subtasks that represent high-level decisions. The top-level policy chooses multi-step transitions, which introduces non-stationarity in the high-level decision process if subtasks are learned concurrently, and also prevents global optimality. The authors discuss the idea of compositionality, but do not explore the concept further. Saxe et al. [2017] propose a hierarchical multi-task architecture that does exploit compositionality. Their multitask LMDP maintains a parallel distributed representation of tasks, reducing the complexity through stacking. However, the approach requires to augment the state space with many additional boundary (subtask) states. Further, the stacking introduces additional costs (cf. their Equation 10), and does not provide global optimality.

The options keyboard [Barreto et al., 2019] combines a successor feature representation with generalized policy improvement to obtain subtask policies from a set of base subtasks without learning, similar to subtask compositionality used in the proposed method. However, unlike in this chapter, their composition weights have to be set manually, and although the composed policy is guaranteed to be better than the individual base policies, it is not guaranteed to be optimal.

The aim of this work is to propose an efficient approach to hierarchical RL that does not sacrifice global optimality. To do so, we integrate both concurrent task composition, as done in the above approaches, together with hierarchical composition, where skills are chained in a temporal sequence, under the framework of LMDPs.

The proposed method is similar to that of Wen et al. [2020] since a hierarchical decomposition based on a partition of the state space is defined, and exploit the equivalence of subtasks to reduce the learning effort. Unlike previous work, however, our approach is not restricted to single initial states, does not suffer from non-stationarity in the online setting, proposes a more general definition of equivalence that captures more structure, and guarantees convergence to the optimal value function for stochastic dynamics.

The concept of equivalent subtasks is strongly related to factored (L)MDPs, which capture conditional independence among a set of state variables [Boutilier et al., 1995, Koller and Parr, 2000]. Equivalence arises whenever a subset of state variables are condition-

ally independent of another subset. Several authors have shown how to automatically discover the structure of factored MDPs from experience [Strehl et al., 2007, Kolobov et al., 2012], which in turn could be used to define equivalence classes of subtasks.

3.4 Hierarchical LMDPs

In this section we describe our novel approach to hierarchical LMDPs. We first describe the particular form of hierarchical decomposition that we consider, and then present algorithms for solving a decomposed LMDP.

3.4.1 Hierarchical Decomposition

Our hierarchical decomposition is similar to that of [Wen et al., 2020]. Formally, given an LMDP $\mathcal{L} = \langle \mathcal{S}, \mathcal{T}, \mathbb{P}, \mathcal{R}, \mathcal{J} \rangle$, the set of non-terminal states $\mathcal{S}$ is partitioned into L subsets $\{\mathcal{S}_i\}_{i=1}^{L}$. For each such subset $\mathcal{S}_i$, there exists an induced subtask $\mathcal{L}_i = \langle \mathcal{S}_i, \mathcal{T}_i, \mathbb{P}_i, \mathcal{R}_i, \mathcal{J}_i \rangle$, i.e. an LMDP whose components are defined as follows:

- The set of non-terminal states is $\mathcal{S}_i$.
- The set of terminal states $\mathcal{T}_i = \{\tau \in \mathcal{S}^+ \setminus \mathcal{S}_i : \exists s \in \mathcal{S}_i \text{ s.t. } \tau \in \mathcal{B}(\mathbb{P}(\cdot|s))\}$ includes all states in $\mathcal{S}^+ \setminus \mathcal{S}_i$ (terminal or non-terminal) that are reachable in one step from a state in $\mathcal{S}_i$.
- $\mathbb{P}_i : \mathcal{S}_i \to \Delta(\mathcal{S}_i^+)$ and $\mathcal{R}_i : \mathcal{S}_i \to \mathbb{R}$ are the restrictions of $\mathbb{P}$ and $\mathcal{R}$ to $\mathcal{S}_i$, where $\mathcal{S}_i^+ = \mathcal{S}_i \cup \mathcal{T}_i$ denotes the full set of subtask states.
- The reward of a terminal state $\tau \in \mathcal{T}_i$ equals $\mathcal{J}_i(\tau) = \mathcal{J}(\tau)$ if $\tau \in \mathcal{T}$, and $\mathcal{J}_i(\tau) = \widehat{v}(\tau)$ otherwise, where $\widehat{v}(\tau)$ is the estimated value in $\mathcal{L}$ of the non-terminal state in $\tau \in \mathcal{S} \setminus \mathcal{S}_i$.

Intuitively, if the reward $\mathcal{J}_i(\tau)$ of each terminal state $\tau \in \mathcal{T}_i$ equals its optimal value $v(\tau)$ for the original LMDP $\mathcal{L}$, then solving the subtask $\mathcal{L}_i$ yields the optimal values of the states in $\mathcal{S}_i$. In practice, however, we use an estimate $\widehat{v}(\tau)$ of the optimal value.

In this case, the subtask $\mathcal{L}_i$ is *parameterized* on the value estimate $\widehat{v}$ of terminal states in $\mathcal{T}_i$, and each time the value estimate changes, $\mathcal{L}_i$ can be solved to obtain a new value estimate $\widehat{v}(s)$ for each state $s \in \mathcal{S}_i$.

Let $\mathcal{E} = \cup_{i=1}^{L} \mathcal{T}_i$ be a set of *exit states*, i.e. the union of the terminal states of each subtask in $\{\mathcal{L}_1, \ldots, \mathcal{L}_L\}$. For convenience, let $\mathcal{E}_i = \mathcal{E} \cap \mathcal{S}_i$ denote the set of (non-terminal) exit states in the subtask $\mathcal{L}_i$. Also, consider the notation $K = \max_{i=1}^{L} |\mathcal{S}_i|$, $N = \max_{i=1}^{L} |\mathcal{T}_i|$ and $E = |\mathcal{E}|$.

The notion of subtasks is just like in [Wen et al., 2020].

Definition 1. *Two subtasks $\mathcal{L}_i$ and $\mathcal{L}_j$ are equivalent if there exists a bijection $f : \mathcal{S}_i \to \mathcal{S}_j$ such that the transition probabilities and rewards of non-terminal states are equivalent through f.*

Unlike [Wen et al., 2020], this definition does *not* require the sets of terminal states $\mathcal{T}_i$ and $\mathcal{T}_j$ to be equivalent. Instead, for each class of equivalent subtasks, the approach is to define a single subtask whose set of terminal states is the *union* of the sets of terminal states of subtasks in the class.

Formally, a set of equivalence classes $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_C\}$ is defined, $C \leq L$, i.e. a partition of the set of subtasks $\{\mathcal{L}_1, \ldots, \mathcal{L}_L\}$ such that all subtasks in a given partition are equivalent. Only a single subtask $\mathcal{L}_j = \langle \mathcal{S}_j, \mathcal{T}_j, \mathbb{P}_j, \mathcal{R}_j, \mathcal{J}_j \rangle$ needs to be represented per equivalence class $\mathcal{C}_j \in \mathcal{C}$. The components $\mathcal{S}_j, \mathbb{P}_j, \mathcal{R}_j$ are shared by all subtasks in the equivalence class, while the set of terminal states is $\mathcal{T}_j = \bigcup_{\mathcal{L}_i \in \mathcal{C}_j} \mathcal{T}_i$, where the union is taken w.r.t. the bijection f relating all equivalent subtasks. As before, the reward $\mathcal{J}_j$ of terminal states is parameterized on a given value estimate $\widehat{v}$. We assume that each non-terminal state $s \in \mathcal{S}$ can be easily mapped to its subtask $\mathcal{L}_i$ and equivalence class $\mathcal{C}_j$.

Example 1. Figure 3.1a) shows an example 4-room LMDP with a single terminal state marked F, separate from the room but reachable in one step from the highlighted location. The rooms are only connected via a single doorway; hence the states are partitioned by room, the subtask corresponding to each room has two terminal states in other rooms, plus the terminal state F for the top right room. The 9 exit states in $\mathcal{E}$ are highlighted and correspond to states next to doorways, plus F. Figure 3.1b) shows a single subtask that is equivalent to all four room subtasks, since dynamics is shared inside rooms and the set of terminal states is the union of those of the subtasks. Hence the number of equivalent subtasks is $C = 1$, the number of non-terminal and terminal states of subtasks is $K = 25$ and $N = 5$, respectively, and the number of exit states is $E = 9$.

3.4.2 Subtask Compositionality

During learning, the value estimate $\widehat{v}$ changes frequently, and it is inefficient to solve all subtasks after each change. Instead, we use compositionality to obtain solutions to the subtasks without learning. The idea is to introduce several base LMDPs for each subtask $\mathcal{L}_j$ such that *any* reward function $\mathcal{J}_j$ can be expressed as a combination of the reward functions of the base LMDPs.

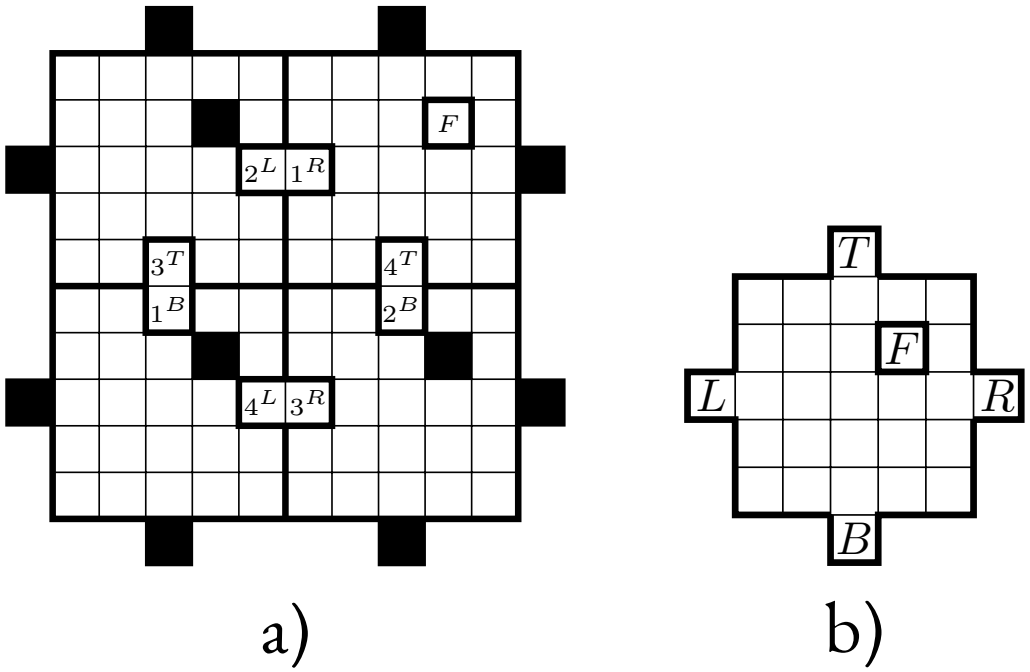


Figure 3.1: a) A 4-room LMDP, with a terminal state F and 8 other exit states; b) a single subtask with 5 terminal states F, L, R, T, B that is equivalent to all 4 room subtasks. Rooms are numbered 1 through 4, left-to-right, then top-to-bottom, and exit state 1^B refers to the exit B of room 1, etc. Black squares represent exit states for which their value function is $z(s) = 0$. The states must be present so that the passive dynamics at the interior states for all the equivalent

Given a subtask $\mathcal{L}_j = \langle \mathcal{S}_j, \mathcal{T}_j, \mathbb{P}_j, \mathcal{R}_j, \mathcal{J}_j \rangle$ as defined above, assume that the set $\mathcal{T}_j$ contains n states, i.e. $\mathcal{T}_j = \{\tau_1, \ldots, \tau_n\}$. Then, n base LMDPs $\mathcal{L}_j^1, \ldots, \mathcal{L}_j^n$ are defined, where each base LMDP is given by $\mathcal{L}_j^k = \langle \mathcal{S}_j, \mathcal{T}_j, \mathbb{P}_j, \mathcal{R}_j, \mathcal{J}_j^k \rangle$. Hence the base LMDPs only differ in the reward of terminal states. Concretely, the exponentiated reward is defined as $z_j^k(\tau) = 1$ if $\tau = \tau_k$, and $z_j^k(\tau) = 0$ otherwise. This corresponds to an actual reward of $\mathcal{J}_j^k(\tau) = 0$ for $\tau = \tau_k$, and $\mathcal{J}_j^k(\tau) = -\infty$ otherwise.

Even though the exit reward $\mathcal{J}_j^k(\tau)$ equals negative infinity for terminal states different from τ_k, this does not cause computational issues in the exponentiated space, since the value $z_j^k(\tau) = 0$ is well-defined in (2.37) and (2.38). Moreover, there are two good reasons for defining the rewards in this way. The first is that the rewards form a convenient basis that allows expressing *any* value estimate on the terminal states in $\mathcal{T}_j$ as a linear combination of $z_j^1, \ldots, z_j^n$. The second is that a value estimate $\widehat{z}(\tau) = 0$ can be used to *turn off* terminal state τ, since the definition of the optimal policy in (2.38) assigns probability 0 to any transition that leads to a state τ with $\widehat{z}(\tau) = 0$. This is the reason

that the sets of terminal states do not need to be equal for equivalent subtasks.

Now assume that the base LMDPs are solved to obtain the optimal value functions $z_j^1, \ldots, z_j^n$. Also assume a given value estimate $\widehat{v}$ for the terminal states in $\mathcal{T}_j$, i.e. $\mathcal{J}_j(\tau) = \widehat{v}(\tau)$ for each $\tau \in \mathcal{T}_j$. Then the exponentiated reward $\widehat{z}(\tau) = e^{\widehat{v}(\tau)/\lambda}$ of each terminal state can be written as

$$\widehat{z}(\tau) = \sum_{k=1}^{n} w_k z_j^k(\tau) = \sum_{k=1}^{n} \widehat{z}(\tau_k) z_j^k(\tau), \tag{3.1}$$

where each weight is simply given by $w_k = \widehat{z}(\tau_k)$. This is because for a given terminal state $\tau_\ell \in \mathcal{T}_j$, the value $z_j^k(\tau_\ell)$ equals 0 for $k \neq \ell$, so the weighted sum simplifies to $w_\ell z_j^\ell(\tau_\ell) = w_\ell \cdot 1 = \widehat{z}(\tau_\ell)$.

Due to compositionality, the estimated value of each non-terminal state $s \in \mathcal{S}_i$ is

$$\widehat{z}(s) = \sum_{k=1}^{n} \widehat{z}(\tau_k) z_j^k(s) \quad \forall s \in \mathcal{S}_i, \forall \mathcal{L}_i \in \mathcal{C}_j. \tag{3.2}$$

Here, the terminal states $\tau_1, \ldots, \tau_n$ are by definition exit states in $\mathcal{E}$. Assuming that we have access to a value estimate $\widehat{z}_\mathcal{E} : \mathcal{E} \to \mathbb{R}$ on exit states, as well as the value functions $z_j^1, \ldots, z_j^n$ of all base LMDPs, (3.2) can be used to express the value estimate of each other state without learning. Hence (3.2) is a form of value function decomposition, that allows expressing the values of arbitrary states in $\mathcal{S}$ in terms of value functions with smaller domains. Concretely, there are $O(CN)$ base LMDPs, each with $O(M)$ values, so in total $O(CMN + E)$ values are needed for the decomposition.

Example 1. In the 4-room example, there are five base LMDPs with value functions z^F, z^L, z^R, z^T and z^B, respectively. Given an initial value estimate $\widehat{z}_\mathcal{E}$ for each exit state in $\mathcal{E}$, a value estimate of any state in the top left room is given by

$$\widehat{z}(s) = \widehat{z}_\mathcal{E}(1^B) z^B(s) + \widehat{z}_\mathcal{E}(1^R) z^R(s),$$

where $\widehat{z}_\mathcal{E}(F) = \widehat{z}_\mathcal{E}(L) = \widehat{z}_\mathcal{E}(T) = 0$ can be used to indicate that the terminal states F, L and T are not present in the top left room. $CMN = 125$ values are needed to store the value functions of the 5 base LMDPs, and $E = 9$ values to store the value estimates of all exit states. Although this is more than the 100 states of the original LMDP, if the number of rooms increases to $X \times Y$, the term CMN is a constant as long as all rooms have equivalent dynamics, and the number of exit states is $E = (2X - 1)(2Y - 1)$, which is much smaller than the $25XY$ total states. For 10×10 rooms, the value function decomposition requires 486 values to represent the values of $2{,}500$ states.

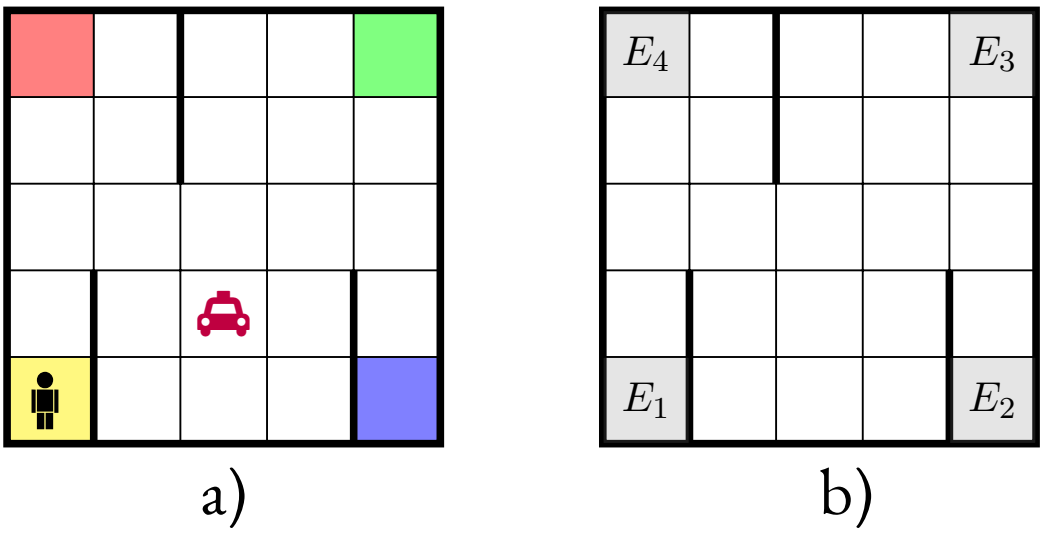


Figure 3.2: a) A 5x5 grid of the Taxi LMDP, where there are 4 colored pickup/drop-off locations (which are one step away from the corresponding gridcell); b) the only existing subtask with exit states E_1, E_2, E_3, and E_4 which are one step away of the grid corners.

The 4-room example is limited in the sense that changing the configuration and size of the rooms may break the assumption of equivalence, which in turn makes the hierarchical approach less powerful. However, the notion of equivalence is naturally associated with factored (L)MDPs, in which the state is factored into a set of variables $\mathcal{V} = \{v_1, \ldots, v_m\}$, i.e. $\mathcal{S} = \mathcal{D}(v_1) \times \cdots \times \mathcal{D}(v_m)$, where $D(v_i)$ is the domain of variable v_i, $1 \leq i \leq m$. Concretely, if there is a subset of variables $\mathcal{U} \subset \mathcal{V}$ such that the transitions among $\mathcal{U}$ are independent of the variables in $\mathcal{V} \setminus \mathcal{U}$, then it is natural to partition the states based on their assignment to the variables in $\mathcal{V} \setminus \mathcal{U}$. Consequently, there is a single equivalent subtask whose set of states is $\times_{v \in \mathcal{U}} D(v)$, i.e. all partial states on the variables in $\mathcal{U}$.

Example 2. The Taxi domain [Dietterich, 2000] (see Figure 3.2) is described by three variables: the location of the taxi (v_1), and the location and destination of the passenger (v_2 and v_3). Since the location of the taxi is independent of the other two, it is natural to partition the states according to the location and destination of the passenger. Each partition consists of the possible locations of the taxi, defining a unique equivalent subtask whose terminal states are the locations at which the taxi can pick up or drop off passengers. Since there are 16 valid combinations of passenger location and destination, there are 16 such equivalent subtasks. [Dietterich, 2000] calls this condition *max node irrelevance*, where "max node" refers to a given subtask.

3.4.3 Eigenvector Approach

If the dynamics $\mathbb{P}$ and the state costs $\mathcal{R}, \mathcal{J}$ are known, we can use the power method to solve the original LMDP $\mathcal{L}$ by composing individual solutions of the subtask LMDPs $\mathcal{L}_i$. In this case, the base LMDPs of all equivalence classes are solved by defining Bellman equations in Equation (2.37). To compute the values of the original LMDP $\mathcal{L}$ for the exit states in $\mathcal{E}$, the compositionality relation in (3.2) gives rise to an additional system of linear equations, one for each non-terminal exit state. This additional system of equations can be reformulated in matrix form defined for the exit states $\mathbf{z}_\mathcal{E}$:

$$\mathbf{z}_\mathcal{E} = G\mathbf{z}_\mathcal{E}. \tag{3.3}$$

Here, the matrix G contains the values of the base LMDPs according to (3.2). The power method can be used on this system of linear equations to obtain the values of all exit states in $\mathcal{E}$.

Example 1. In the 4-room example, the row in G corresponding to $\widehat{z}_\mathcal{E}(2^L)$ contains the element $z^B(2^L)$ in the column for $\widehat{z}_\mathcal{E}(1^B)$, and the element $z^R(2^L)$ in the column for $\widehat{z}_\mathcal{E}(1^R)$, while all other elements equal 0. While the flat approach requires one run of the power method on a large matrix, the hierachical approach needs five runs of the power method on significantly reduced matrices (these runs can be parallelized), and one additional run on a 8×9 (where the value of G is known by definition) matrix, corresponding to (3.3).

The values of states in $\mathcal{S} \setminus \mathcal{E}$ are not explicitly represented since they are given by (3.2). Since now the value $z(s)$ of each state $s \in \mathcal{S}$ can be obtained, the optimal policy is directly defined in terms of the values z and the expression in (2.38). Hence unlike most approaches to hierarchical reinforcement learning, the policy does not select among subtasks, but instead depends directly on the decomposed value estimates.

3.4.4 Online and Intra-task Learning

In the online learning case, we maintain estimates $\widehat{z}_j^1, \ldots, \widehat{z}_j^n$ of the value functions of the base LMDPs associated with each equivalent subtask $\mathcal{L}_j$. These estimates can be updated using the Z-learning rule (2.39) after each transition. But to make learning more efficient, a single transition (s_t, r_t, s_{t+1}) with $s_t \in \mathcal{S}_j$ can be used to update the values of *all* base LMDPs associated with $\mathcal{L}_j$ simultaneously. This is known in the literature as intra-task learning [Kaelbling, 1993, Jonsson and Gómez, 2016].

Given the estimates $\widehat{z}_j^1, \ldots, \widehat{z}_j^n$, then the same system of linear equations in (3.2) could be formulated and solved to obtain the value estimates of exit states. However, it is

impractical to solve this system of equations every time $\widehat{z}_j^1, \ldots, \widehat{z}_j^n$ are updated. Instead, we explicitly maintain estimates $\widehat{z}_{\mathcal{E}}$ of the values of exit states in the set $\mathcal{E}$, and we update them incrementally. For that, (3.2) is turned into an update rule:

$$\widehat{z}_{\mathcal{E}}(s) \leftarrow (1-\alpha_\ell)\widehat{z}_{\mathcal{E}}(s) + \alpha_\ell \sum_{k=1}^{n} \widehat{z}_j^k(s)\widehat{z}_{\mathcal{E}}(\tau_k). \tag{3.4}$$

The question is when to update the value of an exit state. We propose three different alternatives:

V_1: To update the value of an exit state $s \in \mathcal{E}_i$ each time the agent takes a transition from s.

V_2: When the agent reaches a terminal state of the subtask $\mathcal{L}_i$, update the values of all exit states in $\mathcal{E}_i$.

V_3: When the agent reaches a terminal state of the subtask $\mathcal{L}_i$, update the values of all exit states in $\mathcal{E}_i$ and all exit states of subtasks in the equivalence class $\mathcal{C}_j$ of $\mathcal{L}_i$.

Again, the estimated policy π is defined directly by the value estimates $\widehat{z}$ and (2.38), and thus does not select among subtasks. We provide the pseudo-code of this in Algorithm 1.

Algorithm 1 Online and Intra-Task Learning Algorithm

1: **Input:** An LMDP $\mathcal{L} = \langle \mathcal{S}, \mathcal{T}, \mathbb{P}, \mathcal{R}, \mathcal{J} \rangle$ and a partition $\{\mathcal{S}_i\}_{i=1}^{L}$ of $\mathcal{S}$
A set $\{\mathcal{C}_1, \ldots, \mathcal{C}_C\}$ of equivalent subtasks and related base LMDPs $\mathcal{L}_j^k = \langle \mathcal{S}_j, \mathcal{T}_j, \mathbb{P}_j, \mathcal{R}_j, \mathcal{J}_j^k \rangle$
2: **Initialization:**
$\widehat{z}_{\mathcal{E}}(s) := 1 \;\; (\forall s \in \mathcal{E})$ ▷ high-level Z function approximation
$\widehat{z}_j^k(s) := 1$ ▷ base LMDPs $1 \ldots |\mathcal{T}_j|$ for each equivalent subtask $\mathcal{L}_j$
3: **while** termination condition is not met **do**
4: observe transition $s_t, r_t, s_{t+1} \sim \widehat{\pi}(\cdot|s_t)$, where $s_t \in \mathcal{S}_i$ and $\mathcal{L}_i \in \mathcal{C}_j$
5: update lower-level estimations $\widehat{z}_j^k(s_t)$ using (2.39)
6: **if** s_t is an exit or s_{t+1} is terminal for current subtask $\mathcal{L}_j$ **then** $\{s_t \in \mathcal{E}$ or $s_{t+1} \in \mathcal{T}_j\}$
7: apply (3.4) to update $\widehat{z}_{\mathcal{E}}$ using variant V_1, V_2 or V_3
8: **return** $\widehat{z}_{\mathcal{E}}$ and value functions for the subtasks

3.4.5 Analysis

Let $\mathcal{L} = \langle \mathcal{S}, \mathcal{T}, \mathbb{P}, \mathcal{R}, \mathcal{J} \rangle$ be an LMDP, and let $\mathcal{L}_i = \langle \mathcal{S}_i, \mathcal{T}_i, \mathbb{P}_i, \mathcal{R}_i, \mathcal{J}_i \rangle$ be a subtask associated with the partition $\mathcal{S}_i \subseteq \mathcal{S}$. Let z denote the optimal value of $\mathcal{L}$, and let z_i denote the optimal value of $\mathcal{L}_i$.

Lemma 5. *If the reward of each terminal state $\tau \in \mathcal{T}_i$ equals its optimal value in $\mathcal{L}$, i.e. $z_i(\tau) = z(\tau)$, the optimal value of each non-terminal state $s \in \mathcal{S}_i$ equals its optimal value in $\mathcal{L}$, i.e. $z_i(s) = z(s)$.*

Proof. Since $\mathbb{P}_i$ and $\mathcal{R}_i$ are the restriction of $\mathbb{P}$ and $\mathcal{R}$ onto $\mathcal{S}_i$, then for each $s \in \mathcal{S}_i$

$$\begin{aligned} z_i(s) &= e^{\mathcal{R}_i(s)/\lambda} \sum_{s'} \mathbb{P}_i(s'|s) z_i(s') \\ &= e^{\mathcal{R}(s)/\lambda} \sum_{s'} \mathbb{P}(s'|s) z_i(s'), \end{aligned}$$

which is the same Bellman equation as for $z(s)$. Since $z_i(\tau) = z(\tau)$ for each terminal state $\tau \in \mathcal{T}_i$, $z_i(s) = z(s)$ is immediately obtained for each non-terminal state $s \in \mathcal{S}_i$. □

As a consequence of Lemma 5, assigning the optimal value $z(\tau)$ to each exit state $\tau \in \mathcal{E}$ yields a solution to (3.3), which is thus guaranteed to have a solution with eigenvalue 1. Lemma 5 also guarantees that (3.2) can be used to compute the optimal value of any arbitrary state given optimal values of the base LMDPs and the exit states. The only necessary conditions needed for convergence to the optimal value function is that (i) $\{\mathcal{S}_i\}_{i=1}^{L}$ is a proper partition of the state space; and (ii) the set of terminal states $\mathcal{T}_i$ of each subtask $\mathcal{L}_i$ includes all states reachable in one step from $\mathcal{S}_i$.

Lemma 6. *The solution to* (3.3) *is unique.*

Proof. By contradiction. Assume that there exists a solution $\mathbf{z}'_{\mathcal{E}}$ which is different from the optimal values $\mathbf{z}_{\mathcal{E}}$. Then $\mathbf{z}$ and $\mathbf{z}'$ can be extended to all states in $\mathcal{S}$ by applying (3.2). Due to the same argument as in the proof of Lemma 5, the solution $\mathbf{z}'$ satisfies the Bellman optimality equation of all states in $\mathcal{S}$. Hence $\mathbf{z}'$ is an optimal value function for the original LMDP $\mathcal{L}$, which contradicts that $\mathbf{z}'$ is different from $\mathbf{z}$ since the Bellman optimality equations have a unique solution. □

Lemma 7. *For each subtask $\mathcal{L}_i$ and state $s \in \mathcal{S}_i^+$, it holds that $z_i^1(s) + \cdots + z_i^n(s) \leq 1$.*

Proof. By induction. The base case is given by terminal states $t_\ell \in \mathcal{T}_i$, in which case $z_i^1(t_\ell) + \cdots + z_i^n(t_\ell) = z_i^\ell(t_\ell) = 1$. For $s \in \mathcal{S}_i$, the Bellman equation for each base LMDP yields

$$\sum_{k=1}^{n} z_i^k(s) = e^{\mathcal{R}_i(s)/\lambda} \sum_{s'} \mathbb{P}(s'|s) \sum_{k=1}^{n} z_i^k(s').$$

Since $\mathcal{R}_i(s) = \mathcal{R}(s) < 0$ holds by assumption, and since $z_i^1(s') + \cdots + z_i^n(s') \leq 1$ holds for each s' by hypothesis of induction, it follows that $z_i^1(s) + \cdots + z_i^n(s) \leq 1$. □

As a consequence, just like the matrix RP in (2.37), the matrix G in (3.3) has spectral radius at most 1, and hence the power method is guaranteed to converge to the unique solution with the largest eigenvalue 1, corresponding to the optimal values of the exit states.

The convergence rate of the power method is exponential in $\gamma < 1$, the eigenvalue of RP or G with second largest value and independent of the state space. The average running time scales linearly with the number of non-zero elements in RP or G [Todorov, 2006], which is drastically reduced compared to the non-hierarchical approach. More precisely, given an upper bound B on the support of $\mathbb{P}$ and a sparse representation, the matrix multiplication in (2.37) has complexity $\mathcal{O}(BS)$. In comparison, the matrix multiplication of the $\mathcal{O}(CN)$ base LMDPs has complexity $\mathcal{O}(BK)$, while the matrix multiplication in (3.3) has complexity $\mathcal{O}(NE)$. Hence the hierarchical approach is competitive whenever $\mathcal{O}(CNBK + NE)$ is smaller than $\mathcal{O}(BS)$. In a 10×10-room example, $CNBK + NE = 500 + 1{,}805 = 2{,}305$, while $BS = 10{,}000$.

3.5 Experiments

We evaluate the learning algorithm in the two classic control problems already introduced.[1]. The objective of this evaluation is to analyze empirically the different update alternatives (V_1, V_2, and V_3), and to compare against a flat approach which exploits the benefits of LMDPs without the hierarchy (Z-IS), and the hierarchical approach Q-learning with options (Q_o) [Sutton et al., 1999] that was introduced in section 2.5.1. The main goal is to empirically demonstrate that the approach here presented is more sample efficient than the other algorithms. Each algorithm is run with four different random seeds to analyze the (normalized) average MAE (mean absolute error) against the optimal value function (computed separately) and its standard deviation over the number of samples. Since the value functions are different for Q-learning and LMDP methods,

[1]Code available at `https://github.com/guillermoim/HRL_LMDP`

the self-normalized MAE (Figures 3.3 and 3.4, top row) for different configurations and domains is reported instead. Further, for a fair comparison between approaches, only the exit set is used for calculating the MAE.

In all experiments, the learning rates for each abstraction level is $\alpha_\ell(t) = c_\ell/(c_\ell + n)$ where n represents the episode each sample t belongs to. The constant c_ℓ is empirically optimized for each domain. For LMDPs, a temperature $\eta = 1$ is used, which provides good results. Q_o solves an equivalent MDP with *deterministic* actions, which should actually give it an advantage. For fairness, Q_o obtains the same per-step negative reward, exploits the same equivalence classes, learns the same subtasks (i.e. reach a terminal state), and has knowledge of which options are available in each state.

Mention how the learning rates are updated in q-learning for options.

In addition, the MAE for the subtasks learning (Figures 3.3 and 3.4, bottom row) is also reported. The intention for this is two-fold: first, to show that the value functions for the subtasks also converge to their optimal value, and second, that the faster learning of the subtasks boosts learning at the higher level.

3.5.1 N-room domain.

Here, we report the results for different environment configurations in which we change the number of the rooms and the room sizes (Figure 3.3) is presented. In all configurations the proposed hierarchical approach outperforms Z-IS and Q_o. Concretely, Q_o suffers from non-stationarity: initial option executions will incur more negative reward than later executions, which causes high-level Q-learning updates to be *incorrect*, and it takes the learner significant time to recover from this.

Figure 3.3 (left) shows results for 3×3 rooms of size 5×5 and Figure 3.3 (center) shows results for 5×5 rooms of size 3×3. Both scenarios have 225 interior states. The difference between variants V_1, V_2 and V_3 is more pronounced in the second case, when the number of subtasks increases (more rooms) and the partition for each subtask is smaller (smaller rooms). Figure 3.3 (right) shows how the method scales with the number of rooms of size 5×5. Again, variant V_3 has the best performance, in this case by a larger margin than before.

In all cases, the convergence of the subtasks coincides almost exactly with the convergence of the higher level. This suggests that, once the agent has learned the value function for the subtasks, learning the value in the reduced number of exit states is an easy process due to compositionality. Surprisingly, the variant V_3 seems to converge before the subtasks do. This might be an effect of the choice of the metric used at the higher level, since only the error at the exit states is reported. It might be the case that, since exit states are pairwise contiguous, the algorithm does not need the value function for the whole subtask to be accurate, only at the required exit states. This phenomenon should disappear if the

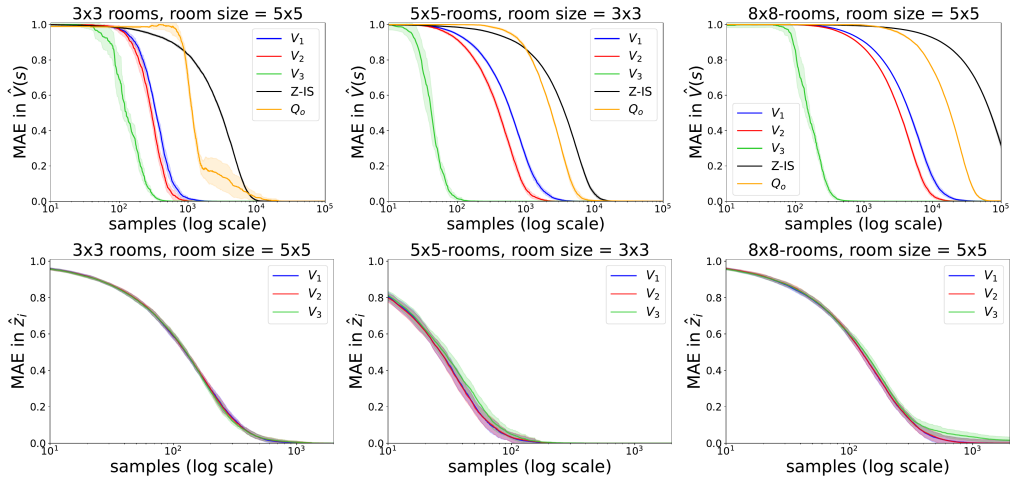


Figure 3.3: Results for 3×3 rooms of size 5×5 (left); 5×5 rooms of size 3×3 (center); 8×8 rooms of size 5×5 (right).

MAE is taken over the whole state space instead.

3.5.2 Taxi Domain.

To allow comparison between all the methods, we adapt the Taxi domain (see Figure 3.2) as follows: when the taxi is at the correct pickup location, it can transition to a state with the passenger in the taxi. In a wrong pickup location, it can instead transition to a terminal state with large negative reward (simulating an unsuccessful pick-up). When the passenger is in the taxi, it can be dropped off at any pickup location, successfully completing the task whenever dropped at the correct destination.

Figure 3.4 shows results in two instances of size 5×5 (408 states) and 10×10 (1608 states). Again, the proposed hierarchical approach outperforms Z-IS and Q_o. In this case, the difference between V_1, V_2 and V_3 is less pronounced, even when the grid size increases, even though V_3 shows slightly better convergence. One possible explanation is the small number of exit states in this problem.

Again, the subtasks of the higher level converge in the same order of magnitude as the subtasks.

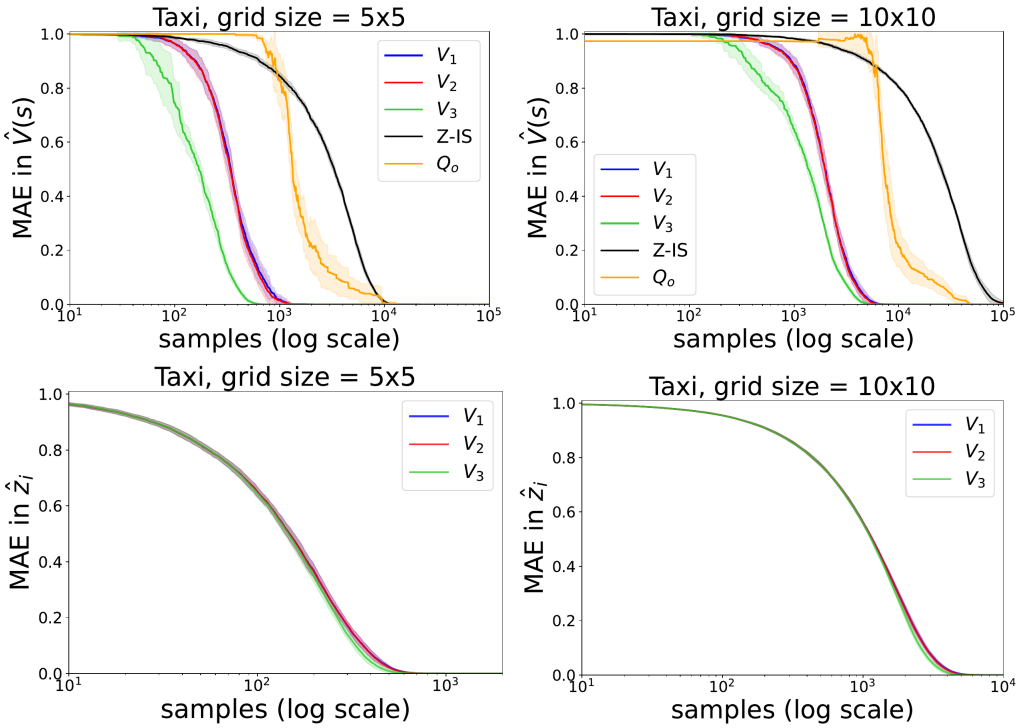


Figure 3.4: Results for 5×5 (left) and 10×10 (right) grids of Taxi domain.

3.6 Discussion and Conclusion

In summary, this chapter introduces a novel approach to hierarchical reinforcement learning that focuses on the class of linearly-solvable Markov decision processes. Using subtask compositionality, the value function can be effectively decomposed, and new hierarchical algorithms can be derived so that they converge to the optimal value function. To the best of our knowledge, this approach is the first to exploit both the concurrent compositionality enabled by LMDPs together with hierarchies and intra-task learning to obtain globally optimal policies efficiently.

The proposed hierarchical decomposition leads to a new form of zero-shot learning that allows to incorporate subtasks that belong to an existing equivalent class without additional learning effort. For example, adding new rooms in our example. This is in contrast with existing methods that only exploit linear compositionality of tasks.

Our approach is limited to OR compositionality of subtasks, but there is no fundamental limitation that prevents arbitrary compositions. The benefits of hierarchies can be combined for example, with the extended value functions proposed in [Nangue Tasse et al., 2020].

Chapter 4

Hierarchical Average-reward Linearly-solvable Markov Decision Pocesses

4.1 Introduction

Most previous work on hierarchical RL considers either the finite-horizon setting or the infinite-horizon setting with discounted rewards. The average-reward setting is better suited for cyclical tasks characterized by continuous experience. In the few works on hierarchical RL in the average-reward setting, either the low-level tasks are assumed to be solved beforehand [Fruit and Lazaric, 2017, Fruit et al., 2017, Wan et al., 2021b] or they have important restrictions that severely reduce their applicability, e.g. a single initial state [Ghavamzadeh and Mahadevan, 2007]. It is therefore an open question how to develop algorithms for hierarchical RL in the average-reward setting in order to learn the low-level and high-level tasks simultaneously.

In this chapter we describe a novel framework for hierarchical RL in the average-reward setting that simultaneously solves low-level and high-level tasks. Concretely, considering the class of Linearly-solvable Markov Decision Processes (LMDPs) [Todorov, 2006]. The method here developed is an extension of the episodic case, described in the previous chapter, to the average-reward setting. Even though the compositionality property for LMDPs or methods of similar flavor have been proposed in RL Hunt et al. [2019], van Niekerk et al. [2019], Nangue Tasse et al. [2020] and in combination with hierarchical RL in the finite-horizon setting Jonsson and Gómez [2016], Saxe et al. [2017], Infante et al. [2022], adapting this idea to the average-reward setting requires careful analysis and poses a new challenge.

Similarly to the first-exit case and unlike most frameworks for hierarchical RL, this approach does not decompose the policy, only the value function. Hence the agent never chooses a subtask to solve, and instead uses the subtasks to compose the value function of the high-level task. This avoids introducing non-stationarity at the higher level when updating the low-level policies.

4.2 Contributions

In this chapter we make the following contributions:

- To develop a hierarchical framework that learns the low-level and the high-level tasks simultaneously in the average-reward setting, without imposing additional restrictions on the low-level tasks.
- To propose two novel algorithms for solving hierarchical RL in the average reward setting: the first one is based on the eigenvector approach used for solving LMDPs. The second is an online variant in which an agent learns simultaneously the low-level and high-level tasks.
- To provide two main theoretical contributions FOR LMDPs: convergence proofs for both differential soft TD-learning for (non-hierarchical) LMDPs and also for the eigenvector approach in the hierarchical case.

This work is the first that extends the combination of compositionality and hierarchical RL to the average-reward setting.

4.3 Related work

Most research on hierarchical RL formulates problems as a Semi-Markov Decision Process (SMDP) with options [Sutton et al., 1999] or the MAXQ decomposition [Dietterich, 2000].

Fruit and Lazaric [2017] and Fruit et al. [2017] propose algorithms for solving SMDPs with options in the average-reward setting, proving that the regret of their algorithms is polynomial in the size of the SMDP components, which may be smaller than the components of the underlying Markov Decision Process (MDP). Wan et al. [2021b] present a version of differential Q-learning for SMDPs with options in the average-reward setting, proving that differential Q-learning converges to the optimal policy. However, the above works assumes that the option policies are given prior to learning. Ghavamzadeh and Mahadevan [2007] propose a framework for hierarchical average-reward RL based on

the MAXQ decomposition, in which low-level tasks are also modeled as average-reward decision processes. However, since the distribution over initial states can change as the high-level policy changes, the authors restrict low-level tasks to have a single initial state.

In the previous chapter combine the compositionality of LMDPs with the equivalence of low-level tasks to develop a framework for hierarchical RL in the finite-horizon setting.

In contrast, our hierarchical framework based on LMDPs can represent the globally optimal policy.

4.4 Alternative method for solving an ALMDP

An alternative method for solving an ALMDP $\mathcal{L}$ is to transform it to a first-exit LMDP. Given a reference state s^* and an original ALMDP $\mathcal{L} = \langle \mathcal{S}, \mathbb{P}, \mathcal{R} \rangle$, define a first-exit LMDP $\mathcal{L}' = \langle \mathcal{S} \setminus \{s^*\}, \{s^*\}, \mathbb{P}', \mathcal{R}', \mathcal{J}' \rangle$, where $\mathbb{P}'(s'|s) = \mathbb{P}(s'|s)$ for all state pairs $(s, s') \in (\mathcal{S} \setminus \{s^*\}) \times \mathcal{S}$, and $\mathcal{J}(s^*) = 0$ (implying $z(s^*) = 1$). By inspection of (2.36) and (2.42), we observe that the Bellman optimality equation of $\mathcal{L}'$ is identical to that of $\mathcal{L}$ if $\mathcal{R}'(s) = \mathcal{R}(s) - \rho$. Even though the agent has no prior knowledge of the exponentiated gain $\Gamma = e^{\eta\rho}$, we perform binary search to find Γ. For a given estimate $\widehat{\Gamma}$ of Γ, after solving $\mathcal{L}'$, $\widehat{\Gamma} z(s^*)$ is compared to $e^{\eta \mathcal{R}(s^*)} \sum_s \mathbb{P}(s|s^*) z(s)$. If $\widehat{\Gamma} z(s^*)$ is greater, then $\widehat{\Gamma}$ is too large, else it is too small.

Alternatively, when $\mathbb{P}$ and $\mathcal{R}$ are not known, an estimate $\widehat{v}$ of the optimal value v and an estimate $\widehat{\rho}$ of the optimal gain ρ are kept using *differential soft TD-learning*, similar to differential Q-learning [Wan et al., 2021a]. We collect (s_t, r_t, s_{t+1}) generated by the estimated policy $\widehat{\pi}$ derived from $\widehat{v}$ as in (2.38). The update rules for $\widehat{v}$ and $\widehat{\rho}$ from (2.41) are as follows

$$\widehat{v}_{t+1}(s_t) \leftarrow \widehat{v}_t(s_t) + \alpha_t \delta_t, \tag{4.1}$$

$$\widehat{\rho}_{t+1} \leftarrow \widehat{\rho}_t + \lambda \alpha_t \delta_t. \tag{4.2}$$

Here, the TD error δ_t is given by

$$\begin{aligned} \delta_t &= r_t - \widehat{\rho}_t - \frac{1}{\eta} \log \frac{\widehat{\pi}_t(s_{t+1}|s_t)}{\mathbb{P}(s_{t+1}|s_t)} + \widehat{v}_t(s_{t+1}) - \widehat{v}_t(s_t) \\ &= r_t - \widehat{\rho}_t + \frac{1}{\eta} \log \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s_t) e^{\eta \widehat{v}_t(s')} - \widehat{v}_t(s_t). \end{aligned}$$

Note that both updates use the same TD error. At any time, we can retrieve the estimates of $\widehat{z}$ and $\widehat{\Gamma}$ by exponentiating $\widehat{v}$ and $\widehat{\rho}$, respectively.

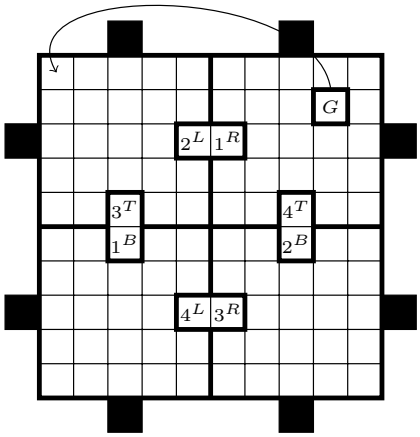


Figure 4.1: An example 4-room ALMDP that shows the adaptation of the previously introduced N-room domain to the average-reward setting.

Theorem 8. *Under mild assumptions, differential soft TD-learning in* (4.1) *and* (4.2) *converges to the optimal values of v and ρ in $\mathcal{L}$.*

Proof sketch. The proof is adapted from the proof of convergence of differential Q-learning Abounadi et al. [2001], Wan et al. [2021a], which requires the ALMDP to be communicating (Assumption 3). Define a Bellman operator T as

$$T(v)(s) = \mathcal{R}(s) + \frac{1}{\eta} \log \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s) e^{\eta v(s')}.$$

To adapt the previous proof, it is sufficient to show that T is a non-expansion in the max norm, i.e. $\|T(x) - T(y)\|_\infty \leq \|x - y\|_\infty$ for each $x, y \in \mathbb{R}^{|\mathcal{S}|}$, and that T satisfies $T(x + c\mathbb{1}) = T(x) + c\mathbb{1}$ for each $x \in \mathbb{R}^{|\mathcal{S}|}$ and constant $c \in \mathbb{R}$, where $\mathbb{1}$ is $|\mathcal{S}|$-dimensional vector of all ones. For completeness, the full proof appears in Appendix A. □

4.5 Hierarchical Average-Reward LMDPs

In this section we present our approach for hierarchical average-reward LMDPs. The idea is to take advantage of the similarity of the value functions in the first-exit and average-reward settings, and use compositionality to compose the value functions of the subtask LMDPs without additional learning.

4.5.1 Hierarchical Decomposition

Consider an ALMDP $\langle \mathcal{S}, \mathbb{P}, \mathcal{R} \rangle$. Similarly to the previous chapter, we assume the state space $\mathcal{S}$ is partitioned into subsets $\{\mathcal{S}_i\}_{i=1}^{L}$, with each partition $\mathcal{S}_i$ inducing a first-exit LMDP $\mathcal{L}_i = \langle \mathcal{S}_i, \mathcal{T}_i, \mathbb{P}_i, \mathcal{R}_i, \mathcal{J}_i \rangle$. The components of each such subtask $\mathcal{L}_i$ are defined as follows:

- The set of states is $\mathcal{S}_i$.
- The set of terminal states $\mathcal{T}_i = \{\tau \in \mathcal{S} \setminus \mathcal{S}_i : \exists s \in \mathcal{S}_i, \mathbb{P}(\tau|s) > 0\}$ contains states not in $\mathcal{S}_i$ that are reachable in one step from any state inside the partition.
- The transition function $\mathbb{P}_i$ and reward function $\mathcal{R}_i$ are projections of $\mathbb{P}$ and $\mathcal{R} - \widehat{\rho}$ onto $\mathcal{S}_i$, where $\widehat{\rho}$ is a gain estimate.
- $\mathcal{J}_i$ is defined for each $\tau \in \mathcal{T}_i$ as $\mathcal{J}_i(\tau) = \widehat{v}(\tau)$, where $\widehat{v}$ is a current value estimate (hence $z_i(\tau) = e^{\eta \widehat{v}(\tau)} = \widehat{z}(\tau)$ is defined by a current exponentiated value estimate $\widehat{z}$).

The Bellman optimality equations of each subtask $\mathcal{L}_i$ are given by

$$z_i(s) = e^{\eta \mathcal{R}_i(s)} \sum_{s'} \mathbb{P}_i(s'|s) z_i(s') \quad \forall s \in \mathcal{S}_i. \tag{4.3}$$

By inspection of the Bellman optimality equations in (2.42) and (4.3), they are equal if $\mathcal{R}_i(s) = \mathcal{R}(s) - \rho$. Thus, if $z_i(\tau) = z(\tau)$ for each $\tau \in \mathcal{T}_i$ then the solution of the subtask $\mathcal{L}_i$ corresponds to the optimal solution for each $s \in \mathcal{S}_i$. However, in general neither ρ nor $z(\tau)$ are known prior to learning and, therefore, estimates $\widehat{\rho}$ and $\widehat{z}(\tau)$ must be used instead. Each subtask $\mathcal{L}_i$ can be seen as being *parameterized* on the value estimates $\widehat{z}(\tau)$ for each $\tau \in \mathcal{T}_j$ and the gain estimate $\widehat{\rho}$. Every time that $\widehat{z}(\tau)$, $\tau \in \mathcal{T}_i$, and $\widehat{\rho}$ change, we can obtain a new value estimate for each $s \in \mathcal{S}_i$ by solving the subtask for the new parameters.

4.5.2 Subtask Compositionality

In the same way as in the first-exit case, it is impractical to solve each subtask $\mathcal{L}_i$ every time the estimate $\widehat{z}(\tau)$ changes for $\tau \in \mathcal{T}_j$. In order to alleviate this, we leverage compositionality for LMDPs. Again, the key insight is to build a basis of value functions that can be combined to obtain the solution for the subtasks.

Consider a subtask $\mathcal{L}_i = \langle \mathcal{S}_i, \mathcal{T}_i, \mathbb{P}_i, \mathcal{R}_i, \mathcal{J}_i \rangle$ and let $n = |\mathcal{T}_i|$. Let $\{\mathcal{L}_i^1, \ldots, \mathcal{L}_i^n\}$ be n base LMDPs that are first-exit LMDPs and terminate in $\mathcal{T}_i$. These base LMDPs only differ from $\mathcal{L}_i$ in the reward of each terminal state $\tau^k \in \mathcal{T}_i$. For all $s \in \mathcal{S}_i$, the reward

for each $\mathcal{L}_i^k$ is by definition $\mathcal{R}_i(s) = \mathcal{R}(s) - \widehat{\rho}$ for all $s \in \mathcal{S}_i$, while at terminal states $\tau \in \mathcal{T}_i$ let the reward function is $z_i^k(\tau; \widehat{\rho}) = 1$ if $\tau = \tau^k$ and $z_i^k(\tau; \widehat{\rho}) = 0$ otherwise. Thus, the base LMDPs are parameterized by the gain estimate $\widehat{\rho}$. This is equivalent to setting the reward to $\mathcal{J}_i^k(\tau) = 0$ if $\tau = \tau_k$ and $\mathcal{J}_i^k(\tau) = -\infty$ otherwise. Intuitively, each base LMDP solves the subtask of reaching one specific terminal state $\tau_k \in \mathcal{T}_i$.

Assume that the solution $z_i^1(\cdot; \rho), \ldots, z_i^n(\cdot; \rho)$ for the base-LMDPs (for the optimal gain ρ) is available as well as the optimal value $z(\tau^k)$ of the original ALMDP for each terminal state $\tau^k \in \mathcal{T}_i$. Then by compositionality we could represent the value function of each terminal state can be represented as a weighted combination of the subtasks:

$$z(\tau) = \sum_{k=1}^{n} w_k z_i^k(\tau; \rho) = \sum_{k=1}^{n} z(\tau^k) z_i^k(\tau; \rho) \;\; \forall \tau \in \mathcal{T}_i. \tag{4.4}$$

Clearly, the RHS in the previous expression evaluates to $z(\tau)$ since $z(\tau^k) z_i^k(\tau; \rho) = z(\tau) \cdot 1$ when $\tau = \tau^k$, and $z(\tau^k) z_i^k(\tau; \rho) = z(\tau^k) \cdot 0$ otherwise.

Thanks to compositionality, it is also possible to represent the value function for each subtask state $s \in \mathcal{S}_i$ as

$$z(s) = \sum_{k=1}^{n} z(\tau^k) z_i^k(s; \rho) \;\; \forall s \in \mathcal{S}_i. \tag{4.5}$$

It is significant to remark that the base LMDPs depend on the gain ρ by the definition of the reward function. This parameter is not known prior to learning. The subtasks in practice are solved for the latest estimate $\widehat{\rho}$ and must be re-learned for every update of this parameter until convergence.

4.5.3 Efficiency of the value representation

Similar to previous work [Wen et al., 2020, Infante et al., 2022] the equivalence of subtasks is exploited to learn more efficiently. Let $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_C\}$, $C \leq L$, be a set of equivalence classes, i.e. a partition of the set of subtasks $\{\mathcal{L}_1, \ldots, \mathcal{L}_L\}$ such that all subtasks in a given partition are equivalent. As before, a set of exit states as $\mathcal{E} = \cup_{i=1}^{L} \mathcal{T}_i$ is also defined. Due to the decomposition, there is no need to keep an explicit value estimate $\widehat{z}(s)$ for every state $s \in \mathcal{S}$. Instead, it is sufficient to keep a value function for exit states $\widehat{z}_{\mathcal{E}} : \mathcal{E} \to \mathbb{R}$ and a value function for each base LMDP of each equivalence class. This is enough to represent the value for any state $s \in \mathcal{S}$ using the compositionality expression in (4.5).

Again, let $K = \max_{i=1}^{L} |\mathcal{S}_i|$, $N = \max_{i=1}^{L} |\mathcal{T}_i|$ and $E = |\mathcal{E}|$. Then only $O(KN)$ values are needed to represent the base LMDPs of a subtask, and the value function can

be represented with $O(CKN + E)$ values. The decomposition leads to an efficient representation of the value function whenever $CKN + E \ll |\mathcal{S}|$. This is achieved when there are few equivalence classes, the size of each subtask is small (in terms of the number of states) and there are relatively few exit states.

Example 1. Figure 4.1 shows an 4-room LMDP adapted to the average-reward seeing. When reaching the state marked G, separate from the room but reachable in one step from the highlighted location, the agent receives a reward of 0 and, in contrast to the first-exit case, the system transitions to a restart state (top left corner). In all other states the reward is -1. To behave optimally, thus, the agent should visit the state marked as G as often as possible. The other aspects of the decomposition, remain equal to the first-exit setting. In this case, the total number of values needed to store the optimal value function is $E + CKN = 9 + 125 = 134$, and the base LMDPs are faster to learn since they have smaller state space.

4.6 Algorithms

In this section we describe two novel algorithms for solving hierarchical ALMDPs. The first is a two-stage eigenvector approach that relies on first solving the subtasks. The second is an online algorithm in which an agent simultaneously learns the subtasks, the gain and the exit values from samples (s_t, r_t, s_{t+1}). Once again it is important to remark that the values for states $s \notin \mathcal{E}$ are not explicitly represented.

4.6.1 Eigenvector approach

In the episodic case, the base LMDPs are only solved once, and the solutions are then reused to compute the value function $z_{\mathcal{E}}$ on exit states (see section 3.4.3). However, in the case of ALMDPs, the reward functions of base LMDPs depend on the current gain estimate $\widehat{\rho}$, which is initially unknown.

The eigenvector approach for solving hierarchical ALMDPs appears in Algorithm 2. The intuition is that in each iteration, we first solve the subtasks for the latest estimate of the exponentiated gain $\widehat{\Gamma}$. For this, the base LMDPs are solved with (4.3) using with the current value of $\widehat{\rho}$. Then (4.5) is applied, restricted to $\mathcal{E}$ to obtain an estimate of the value for the exit states. This yields the system of linear equations

$$\mathbf{z}_{\mathcal{E}} = G_{\mathcal{E}} \mathbf{z}_{\mathcal{E}}. \tag{4.6}$$

Here, the matrix $G_{\mathcal{E}} \in \mathbb{R}^{|\mathcal{E}| \times |\mathcal{E}|}$ contains the optimal values of the base LMDPs and has elements defined as in (4.5). The previously introduced idea to transform the ALMDP

$\mathcal{L}$ to a first-exit LMDP $\mathcal{L}'$ parameterized on the estimated gain $\widehat{\rho}$ is used, and the optimal exponentiated gain Γ is found using binary search. There is a reference state $s^* \in \mathcal{S}$ (which is by definition an exit state) and on which binary search is performed to find Γ.

Theorem 9. *Algorithm 2 converges to the optimal value function z of $\mathcal{L}$ as $\epsilon \to 0$.*

First note that the optimal value function z of $\mathcal{L}$ exists and is unique due to Assumption 3. Due to the equivalence between $\mathcal{L}$ and the corresponding first-exit LMDP $\mathcal{L}'$, this implies that $\mathcal{L}'$ has a unique solution $z(\cdot;\rho)$ when the estimated gain $\widehat{\rho}$ equals ρ, and that this solution equals $z(\cdot;\rho) = z$, the optimal solution to $\mathcal{L}$.

Lemma 10. *Given a first-exit LMDP $\mathcal{L}'$ parameterized on $\widehat{\rho}$, the optimal value $z(s;\widehat{\rho})$ of each non-terminal state $s \in \mathcal{S}$ is strictly monotonically decreasing in $\widehat{\rho}$.*

Proof. Strict monotonicity requires that there exists $\varepsilon > 0$ such that $z(s;\widehat{\rho}-\varepsilon) > z(s;\widehat{\rho}) > z(s;\widehat{\rho}+\varepsilon)$ when $\varepsilon \to 0$. The first inequality is proven by induction; the second is analogous. The base case is given by the terminal states $\tau \in \mathcal{T}$, for which $z(\tau;\widehat{\rho}-\varepsilon) = z(\tau;\widehat{\rho})$. The inductive case is given by

$$\begin{aligned} z(s;\widehat{\rho}-\varepsilon) &= e^{\eta(\mathcal{R}(s)-(\widehat{\rho}-\varepsilon))} \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s) z(s';\widehat{\rho}-\varepsilon) \\ &\geq e^{\eta\varepsilon} e^{\eta(\mathcal{R}(s)-\rho))} \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s) z(s';\widehat{\rho}) \\ &= e^{\eta\varepsilon} z(s;\widehat{\rho}) > z(s;\widehat{\rho}). \end{aligned}$$

Algorithm 2 Eigenvector approach to solving a hierarchical ALMDP.

Input: $\mathcal{L}, \mathcal{S}_1, \ldots, \mathcal{S}_L, \mathcal{E}, \epsilon, \eta$
lo $\leftarrow 0$, hi $\leftarrow 1$
while hi $-$ lo $> \epsilon$ **do**
    $\widehat{\Gamma} \leftarrow$ (hi + lo) / 2
    Solve base LMDPs $\mathcal{L}_j^1, \ldots, \mathcal{L}_j^n$ for each equivalence class $\mathcal{C}_j$
    Form the matrix $G_{\mathcal{E}}$ from the optimal value functions
    Solve the system of equations $\widehat{\mathbf{z}}_{\mathcal{E}} = G_{\mathcal{E}} \widehat{\mathbf{z}}_{\mathcal{E}}$
    if $\widehat{\Gamma} \widehat{z}_{\mathcal{E}}(s^*) > e^{\eta \mathcal{R}(s^*)} \sum_{s \in \mathcal{S}} \mathbb{P}(s|s^*) \widehat{z}_{\mathcal{E}}(s)$ **then**
        hi $\leftarrow \widehat{\Gamma}$
    else
        lo $\leftarrow \widehat{\Gamma}$
return value functions of all base LMDPs, $\widehat{\mathbf{z}}_{\mathcal{E}}$

This concludes the proof. □

As a consequence of Lemma 10, $\mathcal{L}'$ has a unique solution $z(\cdot, \widehat{\rho})$ for each $\widehat{\rho} \geq \rho$, since the values $z(\cdot, \widehat{\rho})$ decrease as $\widehat{\rho}$ increases. In contrast, there may be values of $\widehat{\rho} > \rho$ for which power iteration does not converge.

Lemma 11. *Given a subtask $\mathcal{L}_i$, if the optimal value of each terminal state $\tau \in \mathcal{T}_i$ equals its optimal value in $\mathcal{L}$, i.e. $z_i(\tau) = z(\tau)$, and the optimal gain ρ in $\mathcal{L}$ is known, then the optimal value of each non-terminal state $s \in \mathcal{S}_i$ is unique and equals $z_i(s) = z(s)$.*

Proof. Since $\mathcal{R}_i$ and $\mathbb{P}_i$ are restrictions of $\mathcal{R} - \rho$ and $\mathbb{P}$, respectively, to $\mathcal{S}_i$, then

$$z_i(s) = e^{\eta \mathcal{R}_i(s)} \sum_{s'} \mathbb{P}_i(s'|s) z_i(s') = e^{\eta(\mathcal{R}(s) - \rho)} \sum_{s'} \mathbb{P}(s'|s) z_i(s'),$$

which is the same Bellman equation as for $z(s)$. Assuming that $z_i(\tau) = z(\tau)$ for each $\tau \in \mathcal{T}$, directly yields that $z_i(s) = z(s)$ for each non-terminal state $s \in \mathcal{S}_i$. □

Corollary 12. *If the optimal gain ρ in $\mathcal{L}$ is known, each base LMDP $\mathcal{L}_j^k$ has a unique solution $z_j^k(\cdot; \rho)$.*

Proof. From (4.5), the optimal values of subtask states satisfy

$$z(s) = \sum_{k=1}^{n} z(\tau^k) z_i^k(s; \rho) \;\; \forall s \in \mathcal{S}_i.$$

Due to Lemma 11, the optimal value $z(s)$ is unique, which is only possible if $z_i^k(s; \rho)$ is unique for each $\tau^k \in \mathcal{T}_i$. □

Combined with Lemma 10, Corollary 12 implies that each base LMDP $\mathcal{L}_j^k$ has a unique solution $z_j^k(\cdot; \widehat{\rho})$ whenever $\widehat{\rho} \geq \rho$.

Lemma 13. *For $\widehat{\rho} \geq \rho$, the equation $\widehat{\mathbf{z}}_{\mathcal{E}} = G_{\mathcal{E}} \widehat{\mathbf{z}}_{\mathcal{E}}$ has a unique solution that equals $z_{\mathcal{E}}(\tau) = z(\tau; \widehat{\rho})$ for each exit $\tau \in \mathcal{E}$, where $z(\cdot; \widehat{\rho})$ is the unique value of the first-exit LMDP $\mathcal{L}'$ for $\widehat{\rho}$.*

Proof. At convergence, due to (4.5) it has to hold for each non-terminal exit $\tau \in \mathcal{E}$ that

$$z_{\mathcal{E}}(\tau) = \sum_{k=1}^{n} z_{\mathcal{E}}(\tau^k) z_i^k(s; \widehat{\rho}) \;\; \forall s \in \mathcal{S}_i,$$

where each τ^k is also an exit and $z_i^k(s; \widehat{\rho})$ is well-defined and unique since $\widehat{\rho} \geq \rho$. This equation is satisfied when $z_{\mathcal{E}}(\tau) = z(\tau; \widehat{\rho})$ for each exit. Since $z(\cdot; \widehat{\rho})$ is unique, this is the only solution. □

Now we have all the ingredients to prove Theorem 9. When $\widehat{\rho} \geq \rho$ (or equivalently, $\widehat{\Gamma} \geq \Gamma$), each base LMDP has a unique solution, and $z_{\mathcal{E}}$ is also unique. Moreover, when $\widehat{\rho} > \rho$, the condition on line 8 is true, which causes binary search to discard all values greater than $\widehat{\rho}$. If the base LMDPs or $z_{\mathcal{E}}$ do not have a unique solution, it means that $\widehat{\rho}$ is too small, and hence all values less than $\widehat{\rho}$ can be discarded. Since the solution $z(\cdot; \widehat{\rho})$ is monotonically decreasing in $\widehat{\rho}$, binary search is guaranteed to find the optimal gain ρ within a factor of ϵ.

4.6.2 Online hierarchical algorithm

In the online case (see Algorithm 3), we keep an estimate of the exponentiated gain $\widehat{\Gamma} = e^{\eta\widehat{\rho}}$ which is updated every timestep. We also keep estimates of the value functions of the base LMDPs $\widehat{z}_i^1(\cdot; \widehat{\rho}), \ldots, \widehat{z}_i^n(\cdot; \widehat{\rho})$ for each equivalence class $\mathcal{C}_i$, and estimates of the value function on exit states $\widehat{z}_{\mathcal{E}}$. All the base LMDPs of the same equivalence class can be updated with the same sample using intra-task learning with the appropriate *importance sampling weights* [Jonsson and Gómez, 2016]. We only update the estimates of the exit states upon visitation. In that case, the compositionality expression in (4.5) is used to derive the following update:

$$\widehat{z}_{\mathcal{E}}(s) \leftarrow (1 - \alpha_\ell)\widehat{z}_{\mathcal{E}}(s) + \alpha_\ell \sum_{k=1}^{n} \widehat{z}_{\mathcal{E}}(\tau^k)\widehat{z}_i^k(s; \rho). \tag{4.7}$$

Here, α_ℓ is the learning rate. Each of the learned components (i.e., gain, base LMDPs and exit state value estimates) maintain independent learning rates.

Algorithm 3 Online hierarchical algorithm.

1: **Input:** $\mathcal{L}, \mathcal{S}_1, \ldots, \mathcal{S}_L, \mathcal{E}, s_0, \epsilon, \eta$
2: $t \leftarrow 0, \widehat{\Gamma} \leftarrow 1$
3: **while** *not terminate* **do**
4: Observe $(s_t, r_t, s_{t+1}) \sim \widehat{\pi}_t$
5: Update $\widehat{z}_j^1(\cdot; \widehat{\rho}), \ldots, \widehat{z}_j^n(\cdot; \widehat{\rho})$ using Equation (4.1)
6: Compute $\widehat{z}(s_t)$ and $\sum_{s'} \mathbb{P}(s'|s_t)\widehat{z}(s')$ using Equation (4.5)
7: Update $\widehat{\Gamma}$ using Equation (4.2)
8: **if** $s_t \in \mathcal{E}$ **then**
9: Update $\widehat{z}_{\mathcal{E}}(s_t)$ using Equation (4.7)
10: **return** value functions of all base LMDPs, $\widehat{z}_{\mathcal{E}}$

4.7 Experiments

We now present the experimental results for the two benchmark control problems introduced in the previous chapter and explain how we adapt them to the continuing case. [1]

We report some results for Algorithm 2. These are simple experiments meant to show that the hierarchical eigenvector method indeed converges to the optimal (exponentiated) gain up to an arbitrary precision, and, consequently, to the optimal value function.

Then, we compare Algorithm 3 against differential soft TD-learning in the flat representation of the ALMDP. Similarly to the first-exit case, the metric used for the experiments Mean Absolute Error (MAE) between the estimated value function $\widehat{z}$ and the true optimal value function z, taken exclusively at the exit states. For each algorithm, the results are averaged over five seeds. Along with the average, the standard deviation is also reported. Here, the learning rates decay every n samples, and are given by the expression $\alpha_\ell \leftarrow k\alpha_\ell$, where k is the decay factor. The hyperparameters n and k are optimized per instance.

Why no V_2 and V_3.

Note that, while the updates rules for differential Z-learning (Equations (2.44) and (2.45)) are in the exponential domain, the here derived soft TD-learning updates (Equations (4.1) and (4.2)) are in the logarithmic domain. This has some advantages such as using a single TD-error for both the value function and the gain update. Similarly, for our hierarchical approach we can keep an estimate of the value function $\widehat{v}_{\mathcal{E}}$ in the logarithmic domain. In such a case, the update rule (4.7) can be expressed as

$$\widehat{v}_{\mathcal{E}}(s) \leftarrow (1-\alpha_\ell)\widehat{v}_{\mathcal{E}}(s) + \alpha_\ell \frac{1}{\eta} \log\Big(\sum_{k=1}^{n} e^{\eta \widehat{v}_{\mathcal{E}}(\tau^k)} \widehat{z}_i^k(s;\rho)\Big).$$

Simultaneously, the gain estimate is updated with (4.2). Recall that, at any moment, these estimates can be expressed in the exponential domain with $\widehat{z}_{\mathcal{E}}(s) = e^{\eta \widehat{v}_{\mathcal{E}}(s)}$ and $\widehat{\Gamma} = e^{\eta\widehat{\rho}}$. Our experimental results show that this version of the hierarchical algorithm is more stable.

Besides the learning error for the higher level, the learning error for the subtasks and the gain are also disclosed for the logarithmic and exponential domains. These results should be able to partly show better stability in the logarithmic domain. In contrast to the first-exit case, now there is a three-way dependency during learning between the (optimal) gain, value function of the subtasks and values of the exit states which makes hierarchical learning in the average-reward setting more challenging.

[1] Code available at `https://github.com/guillermoim/halmdps`.

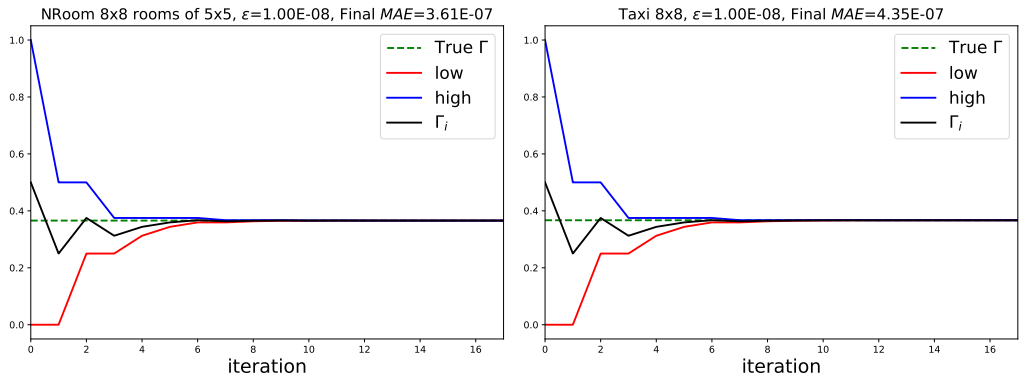


Figure 4.2: Error in $\widehat{\Gamma}$ per iteration for Algorithm 2.

4.7.1 N-room domain

As in the episodic case, there are some 'goal' states with high reward (i.e. 0). When the agent enters a goal state, the next action causes it to receive the given reward and transition to a *restart* location. The number of rooms as well as the size of the rooms are varied to obtain different instances of the problem.

Figure 4.2 (left) shows the results for a single run of Algorithm 2 in the largest instance of this domain. We report the absolute error of the estimate $\widehat{\Gamma}$ at each iteration. We observe that the binary search procedure finds the optimal value up to an arbitrary precision which can be then used to compute the optimal value function.

The results for learning are in Figure 4.3. The difference in the error scale in the figures is due to the initialization of the base LMDPs. We observe that both the hierarchical approaches (logarithmic and exponential domains) outperfom the flat learner by several orders of magnitude (note the logarithmic scale). The average reward setting poses an extra challenge since the 'sparsity' of the reward can make the estimates of the gain oscillate. This ultimately has an impact on the estimates of the base LMDPs and the value estimates of the exit states, and it is likely the reason why in Figure 4.4 the error increases before decreasing down to zero. Nonetheless, the logarithmic version of the algorithm converges faster. This is likely to be caused by a faster convergence of $\widehat{\Gamma}$. The bottom row of Figure 4.3 shows that the magnitude of this error behaves more stably in the logarithmic version, which might cause of the error in the subtasks (middle row of Figure 4.3) to be better upperbounded. Consequently, this might translate in a better convergence in the high-level (top row of Figure 4.3).

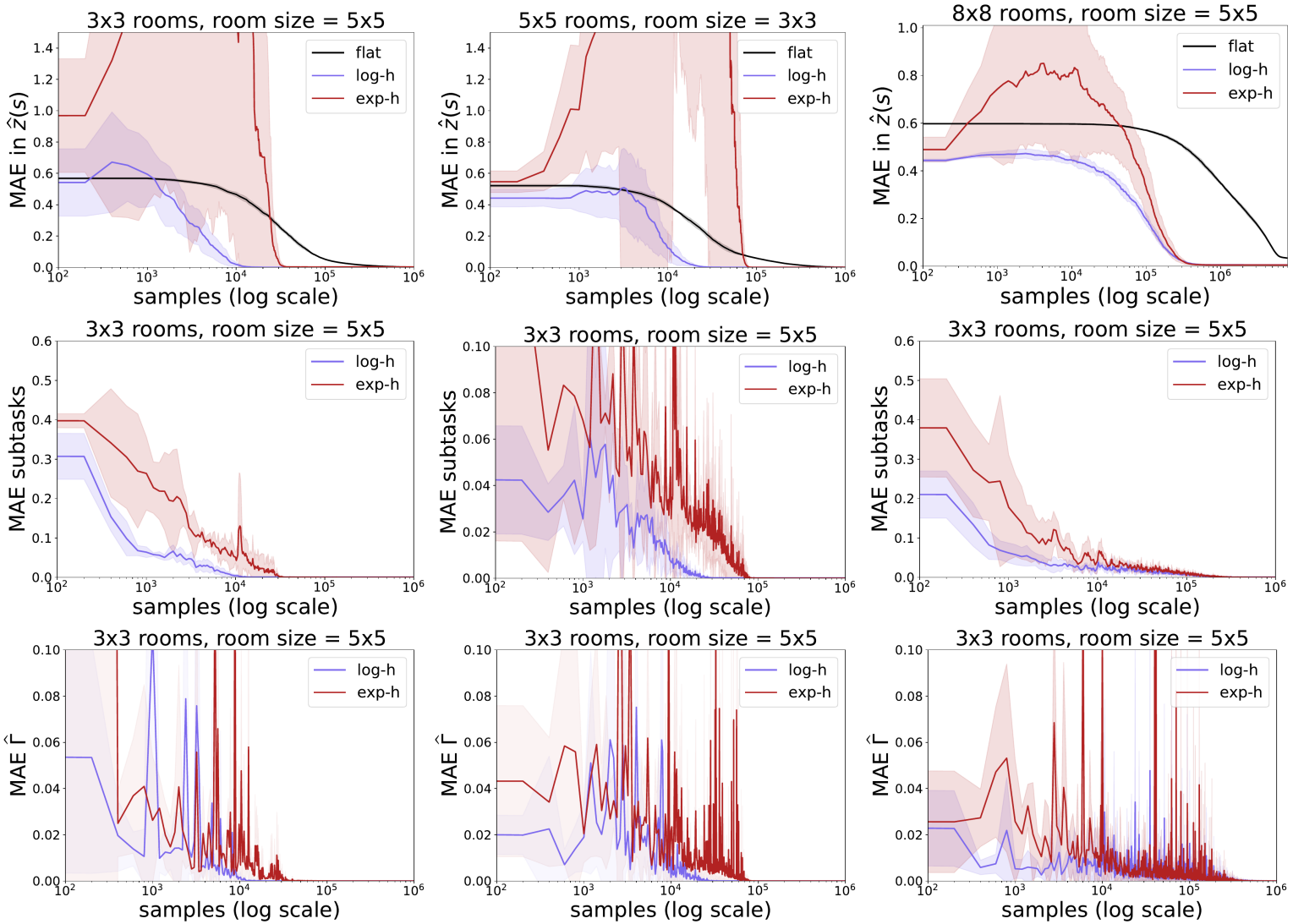


Figure 4.3: Results in N-room when varying the number of rooms and the size of the rooms.

4.7.2 Taxi domain

Adapting this environment to the continuing case is more natural. In this variant of the original domain [Dietterich, 2000], once the passenger has been dropped off, the system transitions to a state in which the driver is in the last drop-off location, but a new passenger appears randomly at another location. Hence the driver immediately receives a new assignment after having successfully completed the previous one.

The results of the eigenvector algortihm in Figure 4.2 (right) shows the same pattern as in the N-room domain. In this domain, the results for the learning algirthm Figure 4.4 also provide the same conclusions as in the N-room domain: the hierarhical approaches need less number of samples to converge, and the logarithmic version seems to be faster and more stable.

4.8 Conclusion

In this chapter we present a novel framework for hierarchical average-reward reinforcement learning which makes it possible to learn the low-level and high-level tasks simultaneously. We propose an eigenvector approach and an online algorithm for solving problems in our hierarchical framework, and show that the former converges to the optimal value function. We provide experimental results of the hierarchical updates both in the logarithmic and exponential domains that show that the logarithmic domain behaves more stably. The experimental results also show that either of the hierarchical versions outperforms a flat learner by several orders of magnitude. As a by-product of our analysis, we also provide a convergence theorem in the non-hierarchical case for average-reward LMDPs, which to the extent of our knowledge, was not previously done. In the future we would like to prove convergence also for the proposed online algorithm.

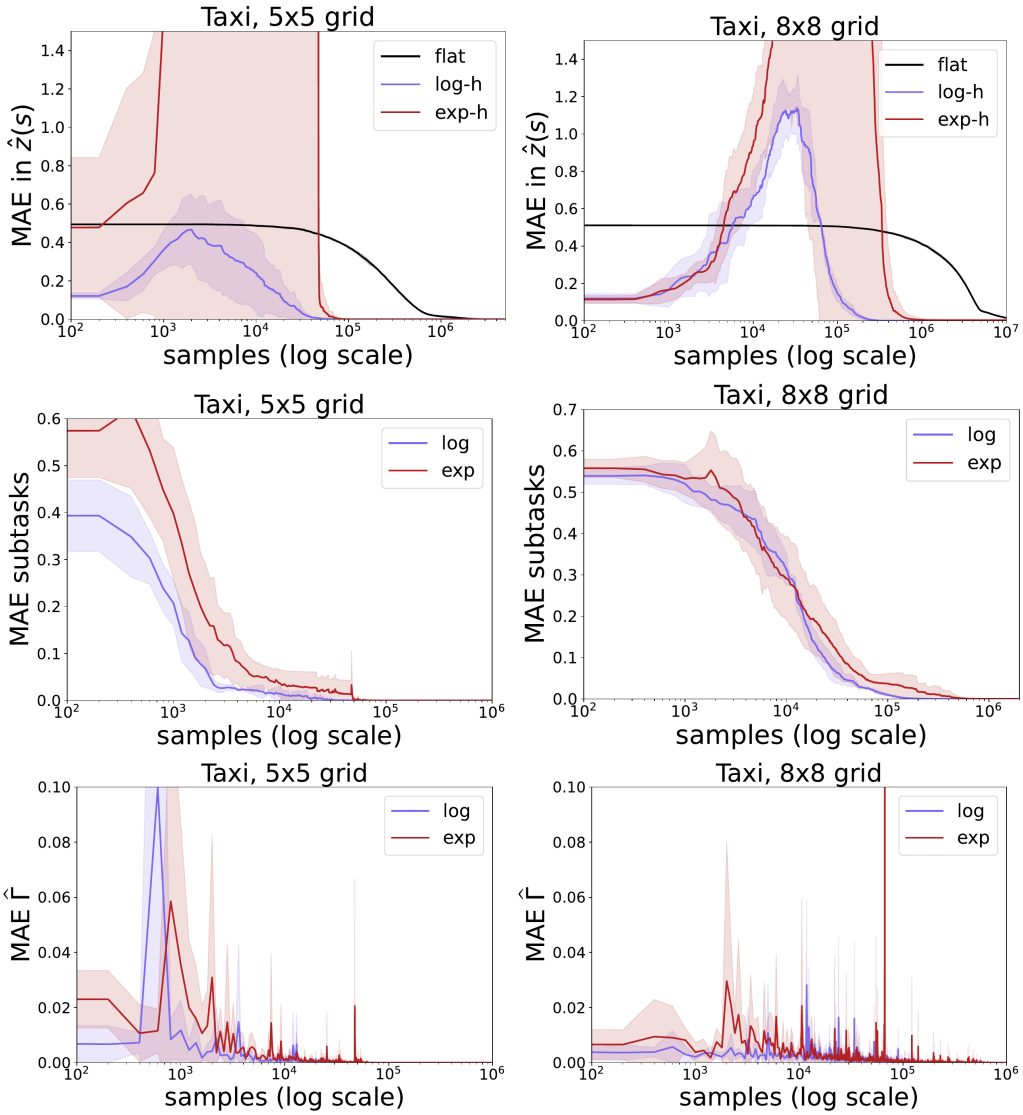


Figure 4.4: Results for 5×5 (top) and 8×8 (bottom) grids of the Taxi domain.

Chapter 5

Compositionality with Successor Features via Policy Basis

5.1 Introduction

Autonomous agents that interact with an environment usually face tasks that comprise complex, entangled behaviors over long horizons. Conventional reinforcement learning (RL) methods have successfully address this. Notwithstanding, there are some limitations that are inherent to classical RL methods, even when they are endowed with function approximation techniques. One of such challenges is that stadard RL algorithms are data greedy and training tasks from scratch is usually inefficient. In this line, there is no universal consensus about how to combine previously learned behaviors so the combined policy generalizes predictably to unseen tasks. As it was introduced in section 2.3, the successor features framework is a way to allow generalization in the context of RL. Nonetheless, the question of identifying which policies should be in the so-called basis was up to this point unclear, leaving the problem of optimal policy transfer open. Another challenge is that traditional RL methods rely on Markovian reward functions. Section 2.6 describes cases where coming up with these reward specification might be difficult or even not possible at all. In such scenarios, there has been a growing interest in alternative methods for non-Markovian task specification in recent years. The work of Camacho et al. [2019] and Icarte et al. [2022] purveyed techniques that successfully use formal languages for such specifications.

The method introduced in this chapter aims to build optimal solutions to non-Markovian reward functions exploiting the generalization capabilities of successor features. In this context, prior techniques for tackling similar problems have been proposed. As it was explained in section 2.6, a fundamental assumption in this setting is there ex-

ist a set of propositional symbols that permits the definition of high-level tasks using logic [Toro Icarte et al., 2019, Vaezipoor et al., 2021] or finite state automata (FSA) [Icarte et al., 2022]. This field is tightly connected to hierarchical RL. This connection arises naturally when satisying subtaks is seen as solving a problem in isolation, which can be then combined for a global solution in the high-level. However, combining optimal solutions for subtasks may potentially result in a suboptimal overall policy, and many of these approaches produce *myopic policies* [Vaezipoor et al., 2021] (whcih are equivalent to recursively optimal policies in the context of hierarchical RL).

More broadly, generalization has been introduced by conditioning the policy or the value function on the specification of the whole task [Schaul et al., 2015]. Such approaches were recently also proposed for tasks with non-Markovian reward functions [Vaezipoor et al., 2021]. However, the methods that specify the whole task usually rely on a blackbox neural network for planning when determining which subgoal to reach next. This makes it hard to interpret the plan to solve the task and although they show promising results in practice, it is unclear whether and when these approaches will generalize to a new task.

Instead, this chapter describes a method that uses task descomposition without sacrifing the global optimality of the solution while at the same times attains predictable generalization. The derived approach, SF-FSA-VI, is a two-stage algorithm in which (i) a set of *local* (sub)policies are learned for a specific choice of a feature representation and (ii) this set of policies is used in a planning step to retrieve the optimal solution to any problem that can be described with an FSA, without additional learning.

5.2 Contributions

More specifically, this work makes the following contributions

- To use successor features to learn a policy basis that is suitable for planning even in stochastic domains.
- To provide a planning framework that uses such policy bases for zero-shot generalization to complex temporal tasks described by an arbitrary FSA.
- To prove that if the policies in this basis are optimal, this framework produces a globally optimal solution even in stochastic domains.
- To provide empirical results that demonstate the improvement over other baselines.

5.3 Preliminaries

In this section we set the ground for our method. The subsequent paragraphs describe the relationship of the building blocks of the method, namely the FSA task specification, the feature vectors, and the policy basis for the low-level.

Our method assumes that the low-level is modeled by a family of MDPs (see section 2.3)

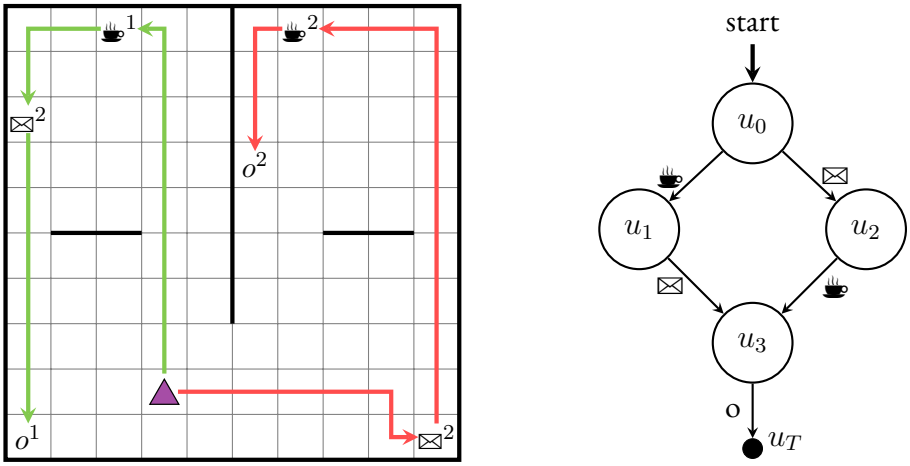
$$\mathcal{M}^{\boldsymbol{\phi}} \equiv \{\langle \mathcal{S}, \mathcal{E}, \mathcal{A}, \mathcal{R}_{\mathbf{w}}, \mathbb{P}_0, \mathbb{P}, \gamma \rangle | \mathcal{R}_{\mathbf{w}} = \mathbf{w}^{\intercal}\boldsymbol{\phi}, \forall \mathbf{w} \in \mathbb{R}^d\}, \tag{5.1}$$

where there is a propositonal vocabulary $\mathcal{P}$ that relates to the feature map $\boldsymbol{\phi}$ and the set of exit states $\mathcal{E}$.

Propositional Logic Environments are assummed to be endowed with a set of high-level, boolean-valued propositional symbols $\mathcal{P}$. Wihtout loss of generality, it is assumed that these symbols are observed when the agent transitions into some exit state $\varepsilon \in \mathcal{E}$ of the low-level MDP $\mathcal{M}^{\boldsymbol{\phi}}$, though this may not be the case. Every transition $(s, a, s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ induces some propositional valuation (assignment of truth values) $2^{\mathcal{P}}$. Such a valuation depends solely on the new state s' and occurs under a mapping $\mathcal{O} : \mathcal{S} \to 2^{\mathcal{P}}$ that is known to the agent. This implies that the agent can associate valuations $2^{\mathcal{P}}$ to transitions at every moment. Propositional symbols are assumed to be mutually exclusive, and the agent cannot observe two symbols evaluated with true in the same transition. A valuation Γ is said to satisfy a propositional symbol p, formally $\Gamma \vDash p$, if p is true in Γ.

Finite State Automaton High-level tasks are instructed via finite state automata. These are tuples $\mathcal{F} = \langle \mathcal{U}, u_0, \mathcal{T}, L, \delta \rangle$ where $\mathcal{U}$ is the finite set of states, $u_0 \in \mathcal{U}$ is the initial state, $\mathcal{T}$ is the set of terminal states with $\mathcal{U} \cap \mathcal{T} = \emptyset$, $L : \mathcal{U} \times (\mathcal{U} \cup \mathcal{T}) \to 2^{\mathcal{P}}$ is a labelling function that maps FSA states transitions to truth values for the propositions and $\delta : \mathcal{T} \to \mathbb{R}$ is a high-level reward function for terminal states. Each transition among FSA states (u, u') defines a subgoal. The agent has to observe some propositional valuation $L(u, u')$ in order to achieve it and FSA states can only be connected by a subgoal. E.g., in Figure 5.1b, the FSA state u_0 has two outgoing subgoals: getting mail (labeled as ✉) and getting coffee (labeled as ☕). Non-existing transitions (u, u') get mapped to $L(u, u') = \bot$. The reward function δ gives a reward only to terminal states.

Feature vectors For a family of MDPs $\mathcal{M}^{\boldsymbol{\phi}}$, the feature map is $\boldsymbol{\phi} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}^{|\mathcal{E}|}$. Each feature component $\boldsymbol{\phi}_j$ is associated with an exit state $\varepsilon_j \in \mathcal{E} = \{\varepsilon_1, \ldots, \varepsilon_{|\mathcal{E}|}\}$. Such vectors are built as follows. At terminal transitions $(s, a, \varepsilon_i) \in T$, $\boldsymbol{\phi}_j = 1$ when $j = i$ and $\boldsymbol{\phi}_j = 0$ when $j \neq i$. For non-terminal transitions, it is just required that



(a) (b)

Figure 5.1: (a) Depiction of the Office environment. The propositional symbols are $\mathcal{P} = \{☕, ✉, o\}$ while the set of exit states is $\mathcal{E} = \{☕^1, ☕^2, ✉^1, ✉^2, o^1, o^2\}$. The red and green paths show a suboptimal and optimal (resp.) trajectories for the task 'get coffee and mail in any order, then go to an office location' whose FSA is represented in (b).

$\mathbf{w}^\intercal \boldsymbol{\phi}(s, a, s') < 1$. In the case that $\boldsymbol{\phi}(s, a, s') = \mathbf{0} \in \mathbb{R}^{|\mathcal{E}|}$, the SF representation in Equation (2.34) of each policy consists of a discounted distribution over the exit states. This indicates how likely it is to reach each exit state following such a policy. Furthermore, it is required that $\mathcal{E} \subset \text{supp}(\mathbb{P}_0)$ so the value functions at exit states are well-defined. Additionally, one can consider extra symbols in $\mathcal{P}$ that are not necessarily satisfied at the exit states. Think, for example, about a propositional symbol $q \in \mathcal{P}$ that represents an obstacle which has a penalty effect. In such cases, the feature vector can be extended to be of dimension $|\mathcal{E}| + 1$, where the extra component is -1 when q is observed, and 0 otherwise.

Policy basis and convex coverage set The recent work of [Alegre et al., 2022] solves the optimal policy transfer learning problem. They draw the connection between the SF transfer learning problem and multi-objective RL (MORL). The pivotal fact is that the SF representation in Equation (2.34) can be interpreted as a multidimensional value function and the construction of the aforementioned set of policies Π can be cast as a multi-objective optimization problem.

Consequently, the optimistic linear support (OLS) algorithm is extended with successor features in order to learn a set of policies that constitutes a *convex coverage set* (CCS) [Roijers et al., 2015]. Their main result is the SFOLS algorithm (see Supplementary Material[1]

[1]Supplementary Material at `https://arxiv.org/abs/2403.15301`

for a full, technical description) in which a set Π_{CCS} is built incrementally by adding (new) policies to such a set, until convergence. The set Π_{CCS} contains all non-dominated policies in terms of their multi-objective value functions, where the dominance relation is defined over scalarized values $V^{\pi}_{\mathbf{w}} = \mathbb{E}_{S_0 \sim \mathbb{P}_0}\left[V^{\pi}_{\mathbf{w}}(S_0)\right]$, and is characterized as

$$\begin{aligned}\Pi_{\text{CCS}} &= \{\pi \mid \exists \mathbf{w} \text{ s.t. } \forall \boldsymbol{\psi}^{\pi'},\ \mathbf{w}^{\intercal}\boldsymbol{\psi}^{\pi} \geq \mathbf{w}^{\intercal}\boldsymbol{\psi}^{\pi'}\} \\ &= \{\pi \mid \exists \mathbf{w} \text{ s.t. } \forall \pi',\ V^{\pi}_{\mathbf{w}} \geq V^{\pi'}_{\mathbf{w}}\}. \end{aligned} \tag{5.2}$$

In every iteration k, SFOLS proposes a new weight vector $\mathbf{w}^k \in \Delta(d)$ for which an optimal policy (and its corresponding SF representation) is learned and added to Π_{CCS} since it is sufficient to consider weights in $\Delta(d)$ to learn the full Π_{CCS}. The output of SFOLS is both Π_{CCS} and the SF representation $\boldsymbol{\psi}^{\pi}$ for every $\pi \in \Pi_{\text{CCS}}$.

Intuitively, all policies in Π_{CCS} are optimal in at least one task $\mathbf{w} \in \Delta(d)$. The set Π_{CCS} is combined with GPI, see Equation (2.35), and upon convergence, for any (new) given task $\mathbf{w}' \in \mathbb{R}^d$, an optimal policy can be identified [Alegre et al., 2022, cf. Theorem 2].

5.4 Using Successor Features to Solve non-Markovian Reward Specifications

SF-FSA-VI focuses on the setting in which a low-level MDP is equipped with a reward structure like in Equation (2.32). The low-level is represented by a family of MDPs $\mathcal{M}^{\phi}$, where each weight vector $\mathbf{w} \in \mathbb{R}^d$ specifies a low-level task. The agent receives high-level task specifications in the more flexible form of an FSA which permits the specification of non-Markovian reward structures. The combination of a low-level family of MDPs and a high-level FSA gives rise to a *product MDP* $\mathcal{M}' = \mathcal{F} \times \mathcal{M}^{\phi}$ that satisfies the Markov property, and where the state space is augmented to be $\mathcal{U} \times \mathcal{S}$. This product MDP follows the same philosophy as the cross product MDP proposed by Icarte et al. [2022].

A product MDP $\mathcal{M}'$ is a well-defined MDP, though in this case the agent now follows a policy $\mu : \mathcal{U} \times \mathcal{S} \to \Delta(\mathcal{A})$, that depends on both the FSA state and the underlying MDP state. $\mathcal{M}'$ can be solved with conventional RL methods such as Q-learning [Watkins and Dayan, 1992] by finding an optimal policy μ^* that maximizes

$$Q^{\mu}(u, s, a) = \mathbb{E}_{\mu}\left[\sum_{i=t}^{\infty} \gamma^{i-t}\mathcal{R}_i \,\middle|\, U_t = u, S_t = s, A_t = a\right].$$

This is, however, impractical since policies should be retrained every time a new high-level task is specified. Exploiting the problem structure is essential for tractable learning, where components can be reused for new task specifications. The special reward structure of

the low-level MDPs and the particular choice of feature vectors, previously introduced, allows defininining an algorithm able to achieve a solution by simply planning in the space of augmented exit states $\mathcal{U} \times \mathcal{E}$. This inherently makes obtaining an optimal policy more efficient than solving the whole product MDP, as the number of states on which it is necessary to compute the high-level value function is drastically reduced.

When presented with different task specifications (e.g. Figure 5.1b), the agent may have to perform the same subtask at different moments of the plan or in different FSAs. The aim is to provide agents with a collection of base behaviors that can be combined to retrieve the optimal behavior for the whole task.

In line with the previous reasoning, SF-FSA-VI a two-step algorithm in which the agent first learns a Π_{CCS} (a set of policies that constitute a CCS) on a well-specified representation of the environment. Then these (sub)policies are used to solve efficiently any FSA task specification on the propositional symbols of the environment.

Example In the office domain depicted in Figure 5.1a, the propositional symbols are $\mathcal{P} = \{☕, ✉, o\}$ while the exit states $\mathcal{E} = \{☕^1, ☕^2, ✉^1, ✉^2, o^1, o^2\}$. Consequently, the same propositional symbol is satisfied at different exit locations, this is $\mathcal{O}(☕^1) \vDash ☕$ and $\mathcal{O}(☕^2) \vDash ☕$. In this case, $\boldsymbol{\phi}(s, a, s') \in \mathbb{R}^6$, is defined as the zero vector $\mathbf{0} \in \mathbb{R}^6$ for every $s' \in \mathcal{S} \setminus \mathcal{E}$ and gets the corresponding vector component equal to 1 when $s' \in \mathcal{E}$. Figure 5.1b the FSA task specification for a 'composite' task in this domain, note that FSAs use symbols in $\mathcal{P}$ to define the subgoals. The natural language interpretation of this FSA is 'get coffee and mail in any order, then go to an office'. Figure 5.1a also shows two different (among multiple possibilities) ways of satisfying such a high-level task. Recursively optimal methods retrieve a suboptimal solution (such as the one in red). Even though this solution satisfies the first subtask in shorter number of steps, the overall solution is longer than the optimal one (in green).

5.4.1 Algorithm

The solution to an FSA task specification implies solving a product MDP $\mathcal{M}' = \mathcal{F} \times \mathcal{M}^{\phi}$. Since there is access to the CCS, the optimal Q-function can be represented by a weight vector $\mathbf{w}^*$:

$$Q^*_{\mathbf{w}}(u, s, a) = \max_{\pi \in \Pi_{\text{CCS}}} \mathbf{w}^*(u)^\intercal \boldsymbol{\psi}^{\pi}(s, a). \tag{5.3}$$

for all $(u, s, a) \in \mathcal{U} \times \mathcal{S} \times \mathcal{A}$. Here, $\mathbf{w}^*_j(u)$ indicates the optimal value from exit state $\varepsilon_j \in \mathcal{E}$ for FSA state u. Then an optimal policy is defined as

$$\mu^*_{\mathbf{w}}(u, s) \in \arg\max_{a \in \mathcal{A}} Q^*_{\mathbf{w}}(u, s, a) \; \forall (s, u) \in \mathcal{U} \times \mathcal{S}. \tag{5.4}$$

Algorithm 4 SF-FSA-VI

Input: Low-level MDP $\mathcal{M}^{\phi}$, task specification $\mathcal{F}$

1: Obtain Π_{CCS} on $\mathcal{M}^{\phi}$.
2: Initially $\mathbf{w}^0(u) = \mathbf{0} \in \mathbb{R}^{|\mathcal{E}|} \;\; \forall u \in \mathcal{U}$.
3: **while** not done **do**
4: **for** $u \in \mathcal{U}$ **do**
5: Update each $\mathbf{w}_j^{k+1}(u)$ with Equation (5.6).
6: **return** $\{\mathbf{w}^*(u) \; \forall u \in \mathcal{U}\}$

Therefore, a key observation is that that finding the optimal weight vectors $\mathbf{w}^*(u)$, $\forall u \in \mathcal{U}$ is enough for retrieving the optimal action value function of the product MDP $\mathcal{M}'$ and, thus, an optimal policy. This vector can be obtained using a dynamic-programming approach similar to value iteration:

$$\mathbf{w}_j^{k+1}(u) = \max_a Q^*_{\mathbf{w}}\big(\tau(u, \mathcal{O}(\varepsilon_j)), a\big) \tag{5.5}$$

$$= \max_{a,\pi} \mathbf{w}^k \big(\tau(u, \mathcal{O}(\varepsilon_j))\big)^{\intercal} \boldsymbol{\psi}^{\pi}(\varepsilon_j, a), \tag{5.6}$$

where $\tau(u, \mathcal{O}(\varepsilon)) \in \mathcal{U}$ is the FSA state that results from achieving the valuation $\mathcal{O}(\varepsilon)$ in u. Note that $\mathbf{w}_j^k(u) = 1$ if $\tau(u, \mathcal{O}(j)) = \mathbf{t}$, per definition, since the high-level reward function $\delta(\mathbf{t}) = 1$. As a result, SF-FSA-VI (see Algorithm 4) is proposed to extract an optimal policy for a product MDP. As $k \to \infty$, SF-FSA-VI converges to the optimal set of weight vectors $\{\mathbf{w}^*(u)\}_{u \in \mathcal{U}}$ and, hence, to the optimal value function in Equation (5.3).

5.4.2 Analysis

SF-FSA-VI converges to the optimal value and, thus, to the optimal policy. The proof starts by first restating the following theorem from [Alegre et al., 2022].

Theorem 14 (Alegre, Bazzan, and Silva, 2022)**.** *Let* Π *be a set of policies such that the set of their expected SFs,* $\Psi = \{\boldsymbol{\psi}^{\pi}\}_{\pi \in \Pi}$*, constitutes a CCS. Then, given any weight vector* $\mathbf{w} \in \mathbb{R}^d$*, the GPI policy* $\pi_{\mathbf{w}}^{GPI}(s) \in \arg\max_{a \in A} \max_{\pi \in \Pi} Q^{\pi}_{\mathbf{w}}(s, a)$ *is optimal with respect to* $\mathbf{w}$ *:* $V_{\mathbf{w}}^{GPI} = V_{\mathbf{w}}^*$*.*

Applied to this setting, once the set of policies Π_{CCS} and associated SFs have been computed, one can define an arbitrary vector $\mathbf{w}$ of rewards on the exit states, and use the CCS to obtain an optimal policy $\mu^*_{\mathbf{w}}$ and an optimal value function $V^*_{\mathbf{w}}$ without learning. Then the composition property can be used by setting the reward of the exit states equal to the optimal value.

The goal is to show that for each augmented state $(u, s) \in \mathcal{U} \times \mathcal{S}$, the value function output by SF-FSA-VI equals the optimal value of (u, s) in the product MDP $\mathcal{M}' = \mathcal{F} \times \mathcal{M}^{\phi}$, i.e. that $V_{\mathbf{w}(u)}(s) = V^*_{\mathcal{M}'}(u, s)$. To do so, it is sufficient to show that the weight vectors $\{\mathbf{w}(u)\}_{u \in \mathcal{U}}$ are optimal.

Each element of $\mathbf{w}(u)$ is recursively defined as $\mathbf{w}_j(u) = V_{\mathbf{w}(\tau(u, \mathcal{O}(\varepsilon_j)))}(\varepsilon_j)$. If all weight vectors are optimal, it holds that $V_{\mathbf{w}(\tau(u, \mathcal{O}(\varepsilon_j)))}(\varepsilon_j) = V^*_{\mathcal{M}'}(\mathbf{w}(\tau(u, \mathcal{O}(\varepsilon_j))), \varepsilon_j)$ for each such exit state. Due to the above theorem, the value function $V_{\mathbf{w}(u)}$ is optimal for $\mathbf{w}(u)$. Due to composition that follows GPE and GPI, this means that the value of each internal state s is optimal, i.e. that $V_{\mathbf{w}(u)}(s) = V^*_{\mathcal{M}'}(u, s)$.

It remains to show that the weight vectors $\{\mathbf{w}(u)\}_{u \in \mathcal{U}}$ returned by the algorithm are indeed optimal. To do so it is sufficient to focus on the set of augmented exit states $\mathcal{U} \times \mathcal{E}$. A set of optimality equations is stated on the weight vectors as follows:

$$\begin{aligned}\mathbf{w}^*_j(u) &= V_{\mathbf{w}^*(\tau(u, \mathcal{O}(\varepsilon)))}(\varepsilon_j) = \max_a Q^*(\tau(u, \mathcal{O}(\varepsilon)), \varepsilon_j, a) \\ &= \max_a \max_\pi \boldsymbol{\psi}^\pi(\varepsilon_j, a)^\intercal \mathbf{w}^*(\tau(u, \mathcal{O}(\varepsilon))),\end{aligned}$$

where $\boldsymbol{\psi}^\pi(\varepsilon_j, a) = \sum_{s'} \mathbb{P}(s'|\varepsilon_j, a)\boldsymbol{\psi}^\pi(\varepsilon_j, a, s')$. The termination condition implies that all subtasks take at least one time step to complete, and due to the discount factor γ, then $\|\boldsymbol{\psi}(\varepsilon_j, a)\|_1 < 1$. Hence the update rule in Equation (5.6) is a contraction and converges to the set of optimal weight vectors due to the Contraction Mapping Theorem.

5.5 Experiments

SF-FSA-VI is tested in three complex discrete environments later described. At test time, the reward is changed to return -1 for every timestep and use the cumulative reward as the performance metric. This reward function effectively captures the number of steps to complete each task. Two types of results are reported. First, an important aspect is to observe the performance of the derived optimal policy, in Equation (5.4), during the **learning** phase. For this, the high-level policy is fully retrained (lines 2-6 in Algorithm 4) every several learning interactions with the environment as Π_{CCS} is being computed. Second, once the base behaviors are learned (this is once a complete Π_{CCS} has been computed), the number of **planning** iterations SF-FSA-VI needs to converge to an optimal solution is measured for different task specifications. In both cases, results are compared against similar baselines.

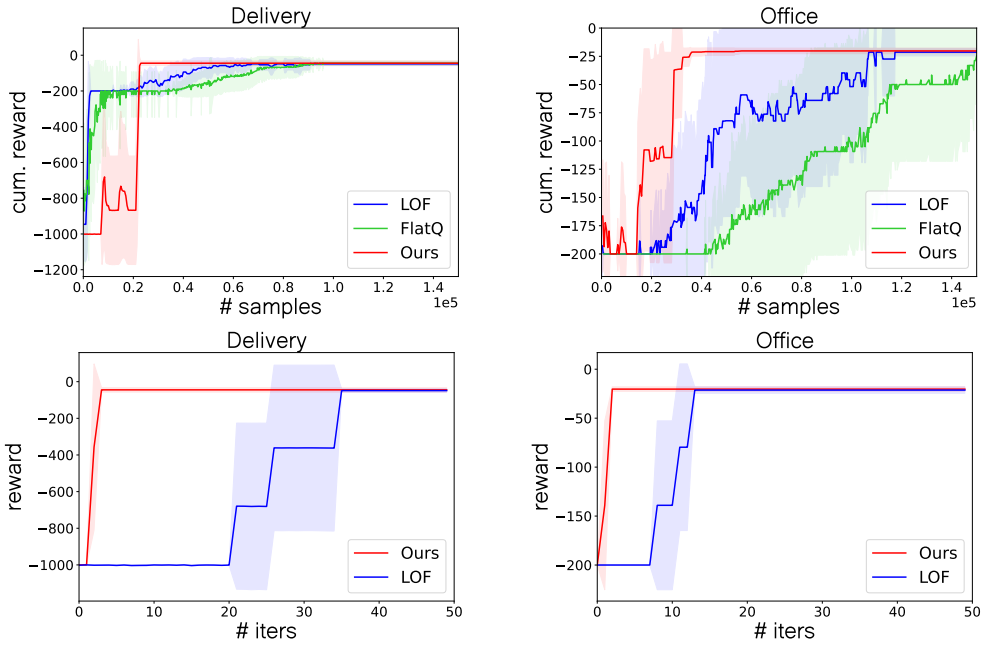


Figure 5.2: Experimental results for learning (Delivery, top-left and Office, bottom-left) and compositionality (Delivery, top-right and Office, bottom-right). Results show the average performance and standard deviation over the three tasks and 5 seeds per task.

5.5.1 Environments and tasks

Office A simplified version of the original Office domain [Toro Icarte et al., 2018] is used. In the context of this work, this is a complex environment since there are three propositional symbols $\mathcal{P} = \{☕, ✉, o\}$ which can be satisfied at different locations, namely $\mathcal{E} = \{☕^1, ☕^2, ✉^1, ✉^1, o^1, o^2\}$. Here, there are no obstacle states and $\phi(s, a, s') = \mathbf{0} \in \mathbb{R}^6$ for non-terminal transitions.

Delivery In the Delivery domain [Araki et al., 2021], shown in Figure 5.3, the set of exit states is $\mathcal{E} = \{A, B, C, H\}$ while the propositonal symbol are $\mathcal{P} = \{A, B, C, H, ■\}$. This means that features vectors are $\phi(s, a, s') \in \mathbb{R}^5$. There are four symbols that are satisfied at the corresponding exit states, the feature vector for trainstions into these exit states correspond to the one-hot encodings of the terminal states. Additionally, there exist a propositional symbol ■ that represents an obstacle. Upon entering one of these states the corresponding feature vector component gets a value of -1. When solving an FSA, the weight of this symbol can be set to $\mathbf{w}_■ = 1000$ which in practice corresponds to a very large negative reward. For regular states (in white in Figure 5.3), the feature vector is $\mathbf{0} \in \mathbb{R}^5$.

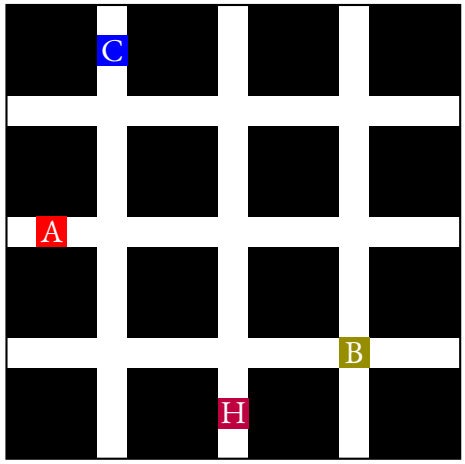


Figure 5.3: Depiction of the Office (a) and Delivery (b) environments, FSA task specification of the composite task in the Office domain and the FSA task specification of the sequential task in the Delivery domain (b). In (a) $\mathcal{P} = \{☕, ✉, o\}$ and $\mathcal{E} = \{☕^1, ☕^2, ✉^1, ✉^2, o^1, o^2\}$. In (b), $\mathcal{E} = \mathcal{P} = \{A, B, C, H\}$.

For each of the environments we define three different tasks: sequential, disjunction and composite (all described in the Supplementary Material). The sequential task is meant to show how our algorithm can indeed be effectively used to plan over long horizons, when the other two tasks show the ability of our method to optimally compose the base (sub)policies in complex settings. In natural language, the tasks in the Delivery domain correspond to: "go to A, then B, then C and finally H" (sequential), "go to A or B, then C and finally H" (disjunction) and "go to A and B in any order, then B, then C and finally H" (composite). The agent has to complete the tasks by avoiding obstacles. The counterpart of these tasks in the Office environment are: "get a coffee, then pick up mail and then go to an office" (sequential), "get a coffee or mail, and then go to an office" (disjunction) and "get a coffee and mail in any order, and then go to an office" (composite). Our agent never learns how to solve these tasks, but rather learns the set of (sub)policies that constitutes the CCS. At test time, we provide the agent with the FSA task specification, extract a high-level optimal policy and test its performance on solving the task.

5.5.2 Baselines

The logical options framework (LOF, Araki et al. [2021]) highlights three desirable properties that hierarchical planning approaches should have: satisfaction, optimality and composability. All of these are satisfied by SF-FSA-VI. Thus, LOF is used as a hierarchical baselines. LOF trains one option per exit state, which are trained simultaneously using

intra-option learning, and then uses a high-level value iteration algorithm to train a meta-policy that decides which option to execute in each of the MDP states. The implementation of LOF used for the experiments follows the description given by the authors. The other baseline is Q-learning on the flat product MDP. Under certain conditions, flat Q-learning converges to the optimal value function but, especially for longer tasks, it may take a large number of samples and it is expected to perform worse than any of the hierarchical alternatives. Additionally, it is trained for a specific task, so it is not able to generalize to other task specifications.

5.5.3 Results

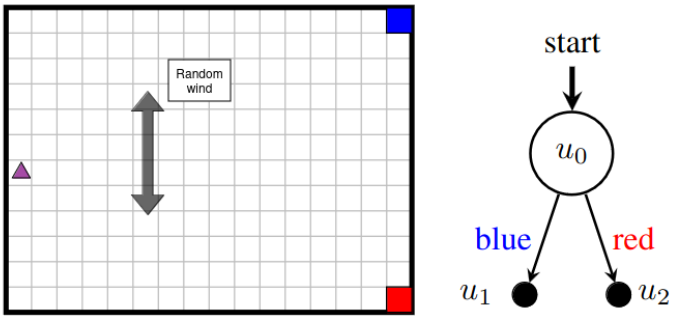


Figure 5.4: Double slit environment (right) and FSA to achieve either of the goal states (blue or red) (left.)

A motivating example

As it is shown later in the experimental results, in deterministic environments, it may suffice to learn a single (sub)policy (or option) per exit state to find an optimal policy via planning. In such cases, it may not be necessary to learn a full CCS. That is why, approaches that use the options framework such as LOF traditionally define one option per subgoal. However, there are scenarios, in which these approaches will not find an optimal policy. This is the case for most stochastic environments. For example, consider the very simple domain of Double Slit in and the FSA task specification in Figure 5.4. In this environment, there are two exit states $\mathcal{E} = \{\text{blue}, \text{red}\}$. The agent starts in the leftmost column and middle row. At every timestep, the agent chooses an action amongst {UP, RIGHT, DOWN} and is pushed one column to the right in addition to moving in the chosen direction, except in the last column. If the agent chooses RIGHT, it moves an extra column to the right. At every timestep there is a random wind that

can blow the agent away up to three positions in the vertical direction. The FSA task specification represents a task which is indifferently satisfied at either of the subgoal states. Note that this is indeed a Markovian reward specification. Since the RIGHT action brings the agent closer to both goals, the optimal behavior in this case is to commit to either goal as late as possible. In this setting, methods that use one policy per subgoal, such as LOF, train two policies to reach both goals. This means that the agent has to commit to one of the goals from the very beginning, which hurts the performance as it has to make up for the consequences of the random noise. On the other hand, the CCS used by SF-FSA-VI will contain an additional policy that is indifferent to both subgoals goals. This leads to a performance gap as our approach achieves an average accumulated reward of -19.7 ± 3.65 and LOF -22.70 ± 5.72.

Learning

Empirical results for learning are shown in Figure 5.2 (top-left and bottom-left). The plots reflect how the different methods (ours, LOF and flat Q-learning) perform at solving an FSA task specification during the learning phase. In the case of SF-FSA-VI and LOF, the learning phase corresponds to obtaining the low level (sub)policies for Π_{CCS} and the options, while for . Results are averaged over the three tasks (sequential, disjunction and composite) previously described for each environment. Each data point in the plots represent the cumulative reward obtained by a fully retrained policy with the current status of Π_{CCS} and options. In both environments, SF-FSA-VI is the first to reach optimal performance. There exist, however, some differences between LOF and SF-FSA-VI. LOF trains all options simultaneously with intra-option learning. This means that, every transition (s_t, a_t, s_{t+1}) is used to update all options' value functions and policies. The learning of a Π_{CCS}, on the other hand, is done sequentially. A fixed sample budget per (sub)policy is set prior to learning, which can be seen as a hyperparameter. We use a total of $8 \cdot 10^3$ samples per (sub)policy in both environments. A experience replay buffer is used to speed up the learning of the policy basis Π_{CCS}. Both options and the SF representation of (sub)policies are learned using Q-learning. Due to the incremental nature, at the beginning of the learning process there might be not enough policies in the basis Π_{CCS} to construct a feasible solution. This is clearly observed in the Delivery domain (Figure 5.2, top left), where at the early stages of the interaction, SF-FSA-VI achieves very low cumulative reward due to failing at delivering a solution. It is not until when there are enough (sub)policies in the basis that Algorithm 4 attains a policy that solves the problem, which eventually converges to an optimal policy. Similarly, LOF converges to an optimal policy albeit it takes slightly longer to learn. In the more complex Office environment, results follow the same pattern. However, this environment breaks one of the of LOF requirements for optimality: to have a single exit state associated with each propositional predicate. In this problem, for each predicate there exist two exit

states that can satisfy them. This makes LOF prone to converge to suboptimal solutions when SF-FSA-VI attains optimality. This is the case for the composite task, where LOF is short-sighted and returns a longer path (in red, Figure 5.1a) in contrast to ours that retrieves the optimal solution (in green, Figure 5.1a). This means that SF-FSA-VI is more flexible in the task specification. In this environment, our algorithm also converges faster with a more obvious gap with respect to LOF. In any case, learning (sub)policies or options is faster than learning directly on the flat product MDP, as flat Q-learning takes the longest to converge.

Planning

Figure 5.2 top-right and bottom-right show how fast SF-FSA-VI and LOF can plan for an optimal solution. Results are again averaged for the three tasks for each environment. Here, a complete policy basis Π_{CCS} has been previously computed, as well as the option's optimal policies. In LOF, the cost of each iteration of value iteration is $|\mathcal{U}| \times |\mathcal{S}| \times |\mathcal{K}|$, where $\mathcal{K}$ is the set of options, while for the Algorithm 4 we propose it is $|\mathcal{U}| \times |\mathcal{E}| \times |\Pi_{\text{CCS}}|$. By definition, the number of options is equivalent to the number of exit states $|\mathcal{K}| = |\mathcal{E}|$, so a single iteration of SF-FSA-VI is more efficient than LOF whenever $|\Pi_{\text{CCS}}| \ll |\mathcal{S}|$. In our experiments, the sizes of the CCS are 15 and 12 for the Delivery and Office domains, respectively, while the sizes of the state spaces are of 225 and 121. Therefore, since our algorithm needs fewer, shorter iterations during planning, it outperforms LOF in terms of planning speed in both domains when composing the global solution. This can be observed in the plots for both environments.

5.6 Related Work

One of the key distinctions of this work compared to prior studies is the optimality of the final solution. As noted by [Dietterich, 2000], hierarchical methods usually have the capability to achieve hierarchical, recursive, or global optimality. The challenge that often arises when subtask policies are trained in isolation is that the combination of these locally optimal policies does not lead to a globally optimal policy but a recursively [Dayan and Hinton, 1992] or hierarchically optimal policy [Sutton et al., 1999, Mann et al., 2015, Araki et al., 2021]. To tackle this challenge, our approach relies on acquiring a set of low-level policies for each subtask and employing planning to identify the optimal combination of low-level policies when solving a particular task. By learning the CCS with OLS [Roijers et al., 2014] in combination with high-level planning, SF-FSA-VI ensures that globally optimal policy is achieved. In this regard, the work of [Alegre et al., 2022] is of particular interest as it was the first work that used OLS and successor features *Barreto2017* for optimal policy transfer learning. However, this method has only applied

in a setting with Markovian reward function and has not been used with non-Markovian task specifications or high-level planning.

On the other hand, many recent approaches proposed to use high-level task specifications in the form of LTL [Toro Icarte et al., 2018, Kuo et al., 2020, Vaezipoor et al., 2021, Jothimurugan et al., 2021], or similar formal language specifications [Toro Icarte et al., 2019, Camacho et al., 2019, Araki et al., 2021, Icarte et al., 2022] to learn policies. However, the majority of the methods in this area are designed for single-task solutions, with only several focusing on acquiring a set of policies that is capable of addressing multiple tasks [Toro Icarte et al., 2018, León et al., 2020, Kuo et al., 2020, Araki et al., 2021, Vaezipoor et al., 2021]. Nonetheless, these approaches cannot guarantee optimality.

From these works, our approach is the most similar to the logical options framework (LOF, Araki et al. [2021]). The main difference is that LOF trains a single policy for each subgoal, resulting in a set of learned policies that is either smaller than or equal to the set acquired through SF-FSA-VI. While employing one policy per subgoal proves sufficient for obtaining a globally optimal policy through planning in deterministic environments [Wen et al., 2020], this may not hold true in stochastic environments, as our experiments demonstrate. In such instances, the policies generated by LOF are hierarchically optimal but fall short of global optimality.

Two notable examples from aforementioned works on multi-task learning with formal language specifications are the works of [Toro Icarte et al., 2018] and [Vaezipoor et al., 2021]. The former struggles with generalizing to unseen tasks, because it uses LTL progression to determine which subtasks need to be learned to solve given tasks. The Q-functions that are subsequently learned for each LTL subtask will therefore not be useful for a new task if its subtasks were not part of the training set. Such limitation does not apply to the latter as it instead encodes the remaining LTL task specification using a neural network and conditions the policy on this LTL embedding. While this approach may be more adaptable to tasks with numerous propositions or subgoals, it risks generating sub-optimal policies as it relies solely on the neural network to select the next proposition to achieve, without incorporating planning. Additionally, since the planning is implicitly done by the neural network, the policy is less interpretable than when explicit planning is used.

The method we propose can be viewed as a method for composing value functions through successor features, akin to previously proposed approaches for composition of value functions and policies [van Niekerk et al., 2019, Barreto et al., 2019, Nangue Tasse et al., 2020, Infante et al., 2022]. In the work of [Infante et al., 2022], which is the closest to our work, the authors propose to learn a basis of value functions that can be combined to form an optimal policy. However, unlike SF-FSA-VI, their approach only works in a restricted class of linearly-solvable MDPs. Lastly, since our approach uses the values of exit states for planning it is also related to planning with exit profiles [Wen

et al., 2020]. The CCS that we propose to use as a policy basis in our work can be seen as a collection of policies that are optimal for all possible exit profiles.

5.7 Discussion and Conclusion

In this work, we address the problem of finding optimal behavior for new non-Markovian goal specifications in known environments. To do so, we introduce a novel approach that uses successor features to learn a policy basis, that can subsequently be used to solve any unseen task specified by an FSA with the set of given predicates $\mathcal{P}$ by planning. SF-FSA-VI is the first that can provably generalize to such new task specification without sacrificing optimality in both deterministic and stochastic environments.

The experiments show that SF-FSA-VI offers several advantages over previous methods. First, due to the use of SF, it allows for faster composition of the high-level value function since it drastically reduces the number of states to plan on. Secondly, thanks to using a CCS over a set of options SF-FSA-VI achieves optimality even in stochastic environments (as shown in the Double Slit example). Lastly, we do not require that there exists a single exit state per predicate which permits more flexible task specification while at the same time allowing deployment in more complex environments.

A limitation of our approach could be the need to construct a full CCS if one wants to attain global optimality. While the construction of CCS is not timecomsuming for environments with several exit states presented in our work, the computation cost of finding the full CCS could become too large for environments with many exit states. In such case one could instead learn a partial CCS at the cost of a bounded decrease in performance [Alegre et al., 2022] or consider splitting the environment into smaller parts with fewer exit states. While our experiments only considered discrete environments, SF-FSA-VI should also be applicable in continuous environments with minor adjustments. These include: using an contiguous set of states instead of a single exit state and using reward shaping to facilitate learning in sparse reward setting.

Bibliography

Jinane Abounadi, Dimitrib Bertsekas, and Vivek S Borkar. Learning algorithms for markov decision processes with average cost. *SIAM Journal on Control and Optimization*, 40(3):681–698, 2001.

Lucas Nunes Alegre, Ana Bazzan, and Bruno C. Da Silva. Optimistic Linear Support and Successor Features as a Basis for Optimal Policy Transfer. pages 394–413. PMLR, June 2022. URL https://proceedings.mlr.press/v162/alegre22a.html.

Brandon Araki, Xiao Li, Kiran Vodrahalli, Jonathan Decastro, Micah Fry, and Daniela Rus. The Logical Options Framework. pages 307–317. PMLR, July 2021. URL https://proceedings.mlr.press/v139/araki21a.html.

André Barreto, Will Dabney, Rémi Munos, Jonathan J. Hunt, Tom Schaul, David Silver, and Hado van Hasselt. Successor features for transfer in reinforcement learning. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 4055–4065, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/350db081a661525235354dd3e19b8c05-Abstract.html.

André Barreto, Diana Borsa, Shaobo Hou, Gheorghe Comanici, Eser Aygün, Philippe Hamel, Daniel Toyama, Jonathan J. Hunt, Shibl Mourad, David Silver, and Doina Precup. The option keyboard: Combining skills in reinforcement learning. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 13031–13041, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/251c5ffd6b62cc21c446c963c76cf214-Abstract.html.

André Barreto, Shaobo Hou, Diana Borsa, David Silver, and Doina Precup. Fast rein-

forcement learning with generalized policy updates. *Proc. Natl. Acad. Sci. USA*, 117 (48):30079–30087, 2020. doi: 10.1073/pnas.1907370117.

Andrew G. Barto and Sridhar Mahadevan. Recent Advances in Hierarchical Reinforcement Learning. *Discrete Event Dynamic Systems*, 13(4):341–379, October 2003. ISSN 1573-7594. doi: 10.1023/A:1025696116075. URL https://doi.org/10.1023/A:1025696116075.

Richard Bellman. Dynamic programming and stochastic control processes. *Information and Control*, 1(3):228–239, 1958. ISSN 0019-9958. doi: https://doi.org/10.1016/S0019-9958(58)80003-0. URL https://www.sciencedirect.com/science/article/pii/S0019995858800030.

Vivek S Borkar. Asynchronous stochastic approximations. *SIAM Journal on Control and Optimization*, 36(3):840–851, 1998.

Vivek S Borkar. *Stochastic approximation: a dynamical systems viewpoint*, volume 48. Springer, 2009.

Craig Boutilier, Richard Dearden, Moisés Goldszmidt, et al. Exploiting structure in policy construction. In *IJCAI*, volume 14, pages 1104–1113, 1995.

Alberto Camacho, Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. Ltl and beyond: Formal languages for reward function specification in reinforcement learning. In *International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 6065–6073, 7 2019. doi: 10.24963/ijcai.2019/840. URL https://doi.org/10.24963/ijcai.2019/840.

Marco da Silva, Frédo Durand, and Jovan Popović. Linear Bellman combination for control of character animation. In *ACM SIGGRAPH 2009 papers*, SIGGRAPH '09, pages 1–10, New York, NY, USA, July 2009. Association for Computing Machinery. ISBN 9781605587264. doi: 10.1145/1576246.1531388. URL https://dl.acm.org/doi/10.1145/1576246.1531388.

Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Comput.*, 5(4):613–624, 1993. doi: 10.1162/neco.1993.5.4.613.

Peter Dayan and Geoffrey E Hinton. Feudal Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 5. Morgan-Kaufmann, 1992. URL https://proceedings.neurips.cc/paper/1992/hash/d14220ee66aeec73c49038385428ec4c-Abstract.html.

T. G. Dietterich. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, November

2000. ISSN 1076-9757. doi: 10.1613/jair.639. URL https://www.jair.org/index.php/jair/article/view/10266.

Ronan Fruit and Alessandro Lazaric. Exploration-exploitation in mdps with options. In Aarti Singh and Xiaojin (Jerry) Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*, volume 54 of *Proceedings of Machine Learning Research*, pages 576–584. PMLR, 2017. URL http://proceedings.mlr.press/v54/fruit17a.html.

Ronan Fruit, Matteo Pirotta, Alessandro Lazaric, and Emma Brunskill. Regret Minimization in MDPs with Options without Prior Knowledge. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://papers.nips.cc/paper_files/paper/2017/hash/90599c8fdd2f6e7a03ad173e2f535751-Abstract.html.

Mohammad Ghavamzadeh and Sridhar Mahadevan. Hierarchical average reward reinforcement learning. *Journal of Machine Learning Research*, 8(87):2629–2669, 2007. URL http://jmlr.org/papers/v8/ghavamzadeh07a.html.

Tuomas Haarnoja, Vitchyr Pong, Aurick Zhou, Murtaza Dalal, Pieter Abbeel, and Sergey Levine. Composable Deep Reinforcement Learning for Robotic Manipulation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6244–6251, May 2018a. doi: 10.1109/ICRA.2018.8460756. URL https://ieeexplore.ieee.org/abstract/document/8460756?casa_token=PZaKv88YacAAAAAA:T8mgFXl1wrM3ARqilO8zfGQ1v-6a8NN14DNZQZgfmPb8KO2LvFZ5Zw0PFEjb2YbL11gnZHsR. ISSN: 2577-087X.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. pages 1861–1870. PMLR, July 2018b. URL https://proceedings.mlr.press/v80/haarnoja18b.html.

Ronald A Howard. Dynamic programming and markov processes. 1960.

Jonathan Hunt, Andre Barreto, Timothy Lillicrap, and Nicolas Heess. Composing Entropic Policies using Divergence Correction. pages 2911–2920. PMLR, May 2019. URL https://proceedings.mlr.press/v97/hunt19a.html.

Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. Reward Machines: Exploiting Reward Function Structure in Reinforcement Learning. *Journal of Artificial Intelligence Research*, 73:173–208, January 2022. ISSN

1076-9757. doi: 10.1613/jair.1.12440. URL https://www.jair.org/index.php/jair/article/view/12440.

Guillermo Infante, Anders Jonsson, and Vicenç Gómez. Globally optimal hierarchical reinforcement learning for linearly-solvable markov decision processes. volume 36, pages 6970–6977, Jun. 2022. doi: 10.1609/aaai.v36i6.20655. URL https://ojs.aaai.org/index.php/AAAI/article/view/20655.

Anders Jonsson and Vicenç Gómez. Hierarchical Linearly-Solvable Markov Decision Problems. *Proceedings of the International Conference on Automated Planning and Scheduling*, 26:193–201, March 2016. ISSN 2334-0843. doi: 10.1609/icaps.v26i1.13750. URL https://ojs.aaai.org/index.php/ICAPS/article/view/13750.

Kishor Jothimurugan, Suguman Bansal, Osbert Bastani, and Rajeev Alur. Compositional Reinforcement Learning from Logical Specifications. In *Advances in Neural Information Processing Systems*, volume 34, pages 10026–10039, 2021. URL https://papers.nips.cc/paper_files/paper/2021/hash/531db99cb00833bcd414459069dc7387-Abstract.html.

L. Kaelbling. Learning to Achieve Goals. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1094–1099, 1993. URL https://www.semanticscholar.org/paper/Learning-to-Achieve-Goals-Kaelbling/6df43f70f383007a946448122b75918e3a9d6682.

Hilbert J. Kappen. Linear Theory for Control of Nonlinear Stochastic Systems. *Physical Review Letters*, 95(20):200201, November 2005. doi: 10.1103/PhysRevLett.95.200201. URL https://link.aps.org/doi/10.1103/PhysRevLett.95.200201.

Hilbert J. Kappen, Vicenç Gómez, and Manfred Opper. Optimal control as a graphical model inference problem. *Machine Learning*, 87(2):159–182, May 2012. ISSN 1573-0565. doi: 10.1007/s10994-012-5278-7. URL https://doi.org/10.1007/s10994-012-5278-7.

Daphne Koller and Ronald Parr. Policy iteration for factored MDPs. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, UAI'00, pages 326–334, San Francisco, CA, USA, June 2000. Morgan Kaufmann Publishers Inc. ISBN 9781558607095.

Andrey Kolobov, Mausam, and Daniel S. Weld. Discovering hidden structure in factored MDPs. *Artificial Intelligence*, 189:19–47, September 2012. ISSN 0004-3702. doi: 10.1016/j.artint.2012.05.002. URL https://www.sciencedirect.com/science/article/pii/S0004370212000598.

Yen-Ling Kuo, Boris Katz, and Andrei Barbu. Encoding formulas as deep networks: Reinforcement learning for zero-shot execution of ltl formulas. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5604–5610, 2020.

Borja G. León, Murray Shanahan, and Francesco Belardinelli. Systematic generalisation through task temporal logic and deep reinforcement learning. *CoRR*, abs/2006.08767, 2020. URL `https://arxiv.org/abs/2006.08767`.

Sridhar Mahadevan. Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Mach. Learn.*, 22(1-3):159–195, 1996. doi: 10.1023/A:1018064306595.

Timothy A Mann, Shie Mannor, and Doina Precup. Approximate value iteration with temporally extended actions. *Journal of Artificial Intelligence Research*, 53:375–438, 2015.

Geraud Nangue Tasse, Steven James, and Benjamin Rosman. A Boolean Task Algebra for Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 9497–9507. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/hash/6ba3af5d7b2790e73f0de32e5c8c1798-Abstract.html`.

Gergely Neu, Anders Jonsson, and Vicenç Gómez. A unified view of entropy-regularized markov decision processes. *CoRR*, abs/1705.07798, 2017.

Ronald Parr and Stuart Russell. Reinforcement Learning with Hierarchies of Machines. In *Advances in Neural Information Processing Systems*, volume 10. MIT Press, 1997. URL `https://proceedings.neurips.cc/paper/1997/hash/5ca3e9b122f61f8f06494c97b1afccf3-Abstract.html`.

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley, 1994. ISBN 978-0-47161977-2. doi: 10.1002/9780470316887. URL `https://doi.org/10.1002/9780470316887`.

Diederik M Roijers, Shimon Whiteson, and Frans A Oliehoek. Linear support for multi-objective coordination graphs. In *International Conference on Autonomous Agents & Multiagent Systems*, volume 2, pages 1297–1304, 2014.

Diederik Marijn Roijers, Shimon Whiteson, and Frans A Oliehoek. Computing convex coverage sets for faster multi-objective coordination. *Journal of Artificial Intelligence Research*, 52:399–443, 2015.

Andrew M. Saxe, Adam C. Earle, and Benjamin Rosman. Hierarchy Through Composition with Multitask LMDPs. In *Proceedings of the 34th International Conference on Machine Learning*, pages 3017–3026. PMLR, July 2017. URL https://proceedings.mlr.press/v70/saxe17a.html.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1312–1320, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/schaul15.html.

John Schulman, Sergey Levine, Pieter Abbeel, Michael I. Jordan, and Philipp Moritz. Trust region policy optimization. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1889–1897. JMLR.org, 2015. URL http://proceedings.mlr.press/v37/schulman15.html.

Alexander L. Strehl, Carlos Diuk, and Michael L. Littman. Efficient structure learning in factored-state MDPs. In *Proceedings of the 22nd national conference on Artificial intelligence - Volume 1*, AAAI'07, pages 645–650, Vancouver, British Columbia, Canada, July 2007. AAAI Press. ISBN 9781577353232.

Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3:9–44, 1988.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112 (1-2):181–211, 1999. doi: 10.1016/S0004-3702(99)00052-1.

Emanuel Todorov. Linearly-solvable markov decision problems. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems*, volume 19. MIT Press, 2006. URL https://proceedings.neurips.cc/paper/2006/file/d806ca13ca3449af72a1ea5aedbed26a-Paper.pdf.

Emanuel Todorov. Efficient computation of optimal actions. *Proceedings of the national academy of sciences*, 106(28):11478–11483, 2009a.

Emanuel Todorov. Compositionality of optimal control laws. In *Advances in Neural Information Processing Systems*, volume 22. Curran Associates, Inc., 2009b. URL https://papers.nips.cc/paper_files/paper/2009/hash/3eb71f6293a2a31f3569e10af6552658-Abstract.html.

Emanuel Todorov. Policy gradients in linearly-solvable mdps. In John D. Lafferty, Christopher K. I. Williams, John Shawe-Taylor, Richard S. Zemel, and Aron Culotta, editors, *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010. Proceedings of a meeting held 6-9 December 2010, Vancouver, British Columbia, Canada*, pages 2298–2306. Curran Associates, Inc., 2010. URL https://proceedings.neurips.cc/paper/2010/hash/69421f032498c97020180038fddb8e24-Abstract.html.

Rodrigo Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. Teaching multiple tasks to an rl agent using ltl. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 452–461, 2018.

Rodrigo Toro Icarte, Ethan Waldie, Toryn Klassen, Rick Valenzano, Margarita Castro, and Sheila McIlraith. Learning Reward Machines for Partially Observable Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://papers.nips.cc/paper_files/paper/2019/hash/532435c44bec236b471a47a88d63513d-Abstract.html.

Pashootan Vaezipoor, Andrew C. Li, Rodrigo A. Toro Icarte, and Sheila A. Mcilraith. LTL2Action: Generalizing LTL Instructions for Multi-Task RL. pages 10497–10508. PMLR, July 2021. URL https://proceedings.mlr.press/v139/vaezipoor21a.html.

Benjamin van Niekerk, Steven D. James, Adam Christopher Earle, and Benjamin Rosman. Composing value functions in reinforcement learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 6401–6409. PMLR, 2019. URL http://proceedings.mlr.press/v97/van-niekerk19a.html.

Nino Vieillard, Olivier Pietquin, and Matthieu Geist. Munchausen Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 4235–4246. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/hash/2c6a0bae0f071cbbf0bb3d5b11d90a82-Abstract.html.

Yi Wan, Abhishek Naik, and Richard S Sutton. Learning and planning in average-reward markov decision processes. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10653–10662. PMLR, 18–24 Jul 2021a. URL https://proceedings.mlr.press/v139/wan21a.html.

Yi Wan, Abhishek Naik, and Richard S. Sutton. Average-reward learning and planning with options. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 22758–22769, 2021b. URL https://proceedings.neurips.cc/paper/2021/hash/c058f544c737782deacefa532d9add4c-Abstract.html.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

Zheng Wen, Doina Precup, Morteza Ibrahimi, Andre Barreto, Benjamin Van Roy, and Satinder Singh. On Efficiency in Hierarchical Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 6708–6718. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/hash/4a5cfa9281924139db466a8a19291aff-Abstract.html.

D.J White. Dynamic programming, markov chains, and the method of successive approximations. *Journal of Mathematical Analysis and Applications*, 6(3):373–376, 1963. ISSN 0022-247X. doi: https://doi.org/10.1016/0022-247X(63)90017-9. URL https://www.sciencedirect.com/science/article/pii/0022247X63900179.

**

Appendix A

Proof of Theorem 8

A.1 Preliminaries

We introduce the notation:

- $\mathbb{1}$ denotes an all-ones vector of length $|\mathcal{S}|$.
- $\mathbb{I}\{p\}$ is the indicator function that takes 1 when predicate p is true and 0 otherwise.

We assumme an underlying continuing LMDP $\mathcal{L} = \langle \mathcal{S}, \mathcal{P}, \mathcal{R} \rangle$ where $\mathcal{S}$ represents the state space, $\mathcal{P}$ the passive dynamics and $\mathcal{R}$ the reward function. Similarly to [Section B.1] in Wan et al. [2021a], we also assume there exists a set-valued process $\{X_t\}$ where X_t is a non-empty subset defined as $X_t = \{(s) : s \text{ component of } v \text{ was updated at timestep } t\}$.

We recall that the TD updates in the asynchronous case are

$$\widehat{v}_{t+1}(s) \leftarrow \widehat{v}_t(s) + \alpha_t(s)\delta_t(s)\mathbb{I}\{s \in X_t\}, \tag{A.1}$$

$$\widehat{\rho}_{t+1} \leftarrow \widehat{\rho}_t + \lambda \sum_s \alpha_t(s)\delta_t(s)\mathbb{I}\{s \in X_t\}. \tag{A.2}$$

The indicator $\mathbb{I}\{s \in X_t\}$ specifies whether the value of state s updates at timestep t. The TD error for state s is

$$\delta_t(s) = r_t(s) - \widehat{\rho}_t + \frac{1}{\eta} \log \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s) e^{\eta \widehat{v}_t(s')} - \widehat{v}_t(s).$$

A.2 Assumptions

We introduce a series of necessary assumptions for convergence. We adapt Assumptions B.1-B.5 in Wan et al. [2021a] to the case of LMDPs. Assumptions 3 and 15 are standard in average-reward settings, while Assumption 16 is the standard Robbins-Monro conditions for step sizes. Assumptions 17 and 18 are introduced in the convergence argument of RVI Q-learning by Borkar [1998] and specify some requirements for the learning rates when asynchronous updates are performed. For more details we refer the reader to Section B.1 of Wan et al. [2021a].

Assumption 15. *(Value function uniqueness) There exists a unique solution to v in equation* (2.41) *up to a constant shift.*

Assumption 16. *(Stepsize assumption)*

$$\alpha_t > 0,\ \sum_{t=0}^{\infty} \alpha_t = \infty,\ \sum_{t=0}^{\infty} \alpha_t^2 < \infty.$$

Assumption 17. *(Asynchronous Stepsize 1) Let $[\cdot]$ denote the integer part of $(\cdot)$, for $x \in (0,1)$*

$$\sup_i \frac{\alpha_{[xi]}}{\alpha_i} < \infty$$

and

$$\frac{\sum_{j=0}^{[yi]} \alpha_j}{\sum_{j=0}^{i} \alpha_j} \to 1$$

uniformly in $y \in [x, 1]$.

Assumption 18. *(Asynchronous Stepsize 2) There exists $\Delta > 0$ such that*

$$\liminf_{t\to\infty} \frac{\nu(t,s)}{t+1} \geq \Delta$$

almost surely, for all $s \in \mathcal{S}$. Here $\nu(t,s)$ represents the visitation count for state s up to timestep t. Furthermore, for all $x > 0$, let

$$N(t,x) = \min\left\{m > t : \sum_{i=t+1}^{m} \alpha_i \geq x\right\}$$

the limit

$$\lim_{t\to\infty} \frac{\sum_{i=\nu(t,s)}^{\nu(N(t,x),s)} \alpha_i}{\sum_{i=\nu(t,s')}^{\nu(N(t,x),s')} \alpha_i}$$

exists for all $s, s' \in \mathcal{S}$.

A.3 The proof

Under the communication assumption, the system

$$v(s) = \mathcal{R}(s) - \rho + \frac{1}{\eta}\log\sum \mathcal{P}(s'|s)e^{\eta v(s')}, \quad \forall s \in \mathcal{S}, \tag{A.3}$$

$$\rho - \hat{\rho}_0 = \lambda\Big(\sum_s v(s) - \sum_s \hat{v}_0(s)\Big), \tag{A.4}$$

has a unique solution for v which we denote as v_∞, where ρ is the optimal gain.

At each timestep the increment to $\hat{\rho}_t$ is λ times the increment to $\hat{v}_t$, and thus, to $\sum_s \hat{v}_t(s)$. The cumulative increment at t can be expressed as

$$\begin{aligned}
\hat{\rho}_t - \hat{\rho}_0 &= \lambda \sum_{i=0}^{t-1}\sum_s \alpha_i(s)\delta_i(s)\mathbb{I}\{s \in X_t\} \\
&= \lambda\Big(\sum_s \hat{v}_t(s) - \sum_s \hat{v}_0(s)\Big) \\
\implies \hat{\rho}_t &= \lambda\sum_s \hat{v}_t(s) - \lambda\sum_s \hat{v}_0(s) + \hat{\rho}_0 \qquad (A.5)\\
&= \lambda\sum_s \hat{v}_t(s) - c, \qquad (A.6)\\
\text{where } c &= \lambda\sum_s \hat{v}_0(s) - \hat{\rho}_0. \qquad (A.7)
\end{aligned}$$

If we replace A.6 in A.1, we obtain

$$\widehat{v}_{t+1}(s) \leftarrow \widehat{v}_t(s) + \alpha_t(s)\widetilde{\delta}_t(s)\mathbb{I}\{s \in X_t\}, \quad \forall s \in \mathcal{S}, \tag{A.8}$$

where

$$\widetilde{\delta}_t(s) = r_t(s) + c - \lambda\sum_s \hat{v}_t(s) - \frac{1}{\eta}\log\sum_{s'\in\mathcal{S}} \mathcal{P}(s'|s)e^{\eta\widehat{v}_t(s')} - \widehat{v}_t(s). \tag{A.9}$$

This can be interpreted as the TD error of an alternative LMDP $\widetilde{\mathcal{L}} = \langle \mathcal{S}, \mathcal{P}, \widetilde{\mathcal{R}}\rangle$ in which the reward is defined as $\widetilde{\mathcal{R}}(s) = \mathcal{R}(s) + c$ and the gain estimate equals $\lambda\sum_{s\in\mathcal{S}} \widehat{v}_t(s)$. The gain of $\widetilde{\mathcal{L}}$ satisfies

$$\widetilde{\rho} = \rho + c. \tag{A.10}$$

The former expression, combined with (A.6) gives

$$\widetilde{\rho} = \lambda\sum_s v_\infty. \tag{A.11}$$

It is easy to verify that v_∞ is not only the solution for the original LMDP $\mathcal{L}$, but also for the alternative LMDP $\widetilde{\mathcal{L}}$,

$$\begin{aligned} v_\infty(s) &= \mathcal{R}(s) - \rho + \frac{1}{\eta}\log\sum_{s'}\mathcal{P}(s'|s)e^{\eta v_\infty(s')} \quad \forall s \in \mathcal{S} \\ &= \mathcal{R}(s) - \widetilde{\rho} + c + \frac{1}{\eta}\log\sum_{s'}\mathcal{P}(s'|s)e^{\eta v_\infty(s')} \quad \forall s \in \mathcal{S} \text{ (by (A.10))} \\ &= \widetilde{\mathcal{R}}(s) - \widetilde{\rho} + \frac{1}{\eta}\log\sum_{s'}\mathcal{P}(s'|s)e^{\eta v_\infty(s')} \quad \forall s \in \mathcal{S}. \end{aligned}$$

Now consider $\hat{\rho}_t$. If we can prove that $\hat{v}_t \to v_\infty$ then by (A.6) we have $\hat{\rho}_t \to \lambda\sum v_\infty - c$. By (A.11), we know that $\lambda\sum v_\infty = \widetilde{\rho}$, then we have $\hat{\rho}_t \to \widetilde{\rho} - c$. Using (A.10), we get

$$\hat{\rho}_t \to \rho \text{ almost surely as } t \to \infty.$$

The idea is to prove the convergence of differential soft TD-learning for the alternative LMDP $\widetilde{\mathcal{L}}$, which is the same solution as for the original LMDP $\mathcal{L}$.

We adapt Theorem B.2 in Wan et al. [2021a].

Theorem 19. *(Convergence of differential TD learning) For any $v_0 \in \mathbb{R}^{|\mathcal{S}|}$, let r_t, X_t, α_t be properly defined and consider the update rule*

$$\widehat{v}_{t+1}(s) \leftarrow \widehat{v}_t(s) + \alpha_t(s)\big(r_t(s) - \lambda\sum_s \hat{v}_t(s) - \frac{1}{\eta}\log\sum_{s'\in\mathcal{S}}\mathcal{P}(s'|s)e^{\eta\widehat{v}_t(s')} - \widehat{v}_t(s)\big)\mathbb{I}\{s \in X_t\}, \tag{A.12}$$

If the following assumptions hold:

1. *Assumptions 4 and 15-18 hold.*
2. *$f : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}$ is Lipschitz and there exists some $u > 0$ such that $\forall c \in \mathbb{R}$ and $x \in \mathbb{R}^{|\mathcal{S}|}$, $f(\mathbb{1}) = u$, $f(x + c\mathbb{1}) = f(x) + cu$ and $f(cx) = cf(x)$*

then $\hat{v}_t$ converges almost surely to v_∞.

We observe that (A.12) is in the same form of equationB.24 in Wan et al. [2021a] and equation7.1 in Borkar [2009]. Thus the results in Section 7.4 in Borkar [2009] and Theorem 3.2 Borkar [1998] apply to show convergence of (A.12). Due to Assumptions 17 and 18, to show convergence of (A.8) suffices to show convergence of the following synchronous update

$$\widehat{v}_{t+1} \leftarrow \widehat{v}_t + \alpha_t\left(T(\widehat{v}_t) - f(\widehat{v}_t)\mathbb{1} - \widehat{v}_t + M_{t+1}\right). \tag{A.13}$$

Here, $\hat{v}_t \in \mathbb{R}^{|\mathcal{S}|}$ is interpreted as a vector, and the operator T is a mapping $T : \mathbb{R}^d \to \mathbb{R}^d$ defined for each state s as

$$T(v)(s) = \mathcal{R}(s) + \frac{1}{\eta} \log \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s) e^{\eta v(s')}.$$

We also define the following operators,

$$\begin{aligned} T^1(v) &= T(v) - \rho \mathbb{1} \\ T^2(v) &= T(v) - f(v)\mathbb{1} = T^1(v) + (\rho - f(v))\mathbb{1}. \end{aligned}$$

The function f is given by $f(v) = \lambda \sum_s v(s)$, which satisfies condition 2 of Theorem 19. The error term $M_{t+1} = r_t - \mathcal{R}$ is only needed in case the reward vector r_t is sampled from a distribution with mean $\mathcal{R}$; if reward is deterministic, M_{t+1} can be omitted. If the reward is sampled from a distribution, it is easy to show that M_{t+1} has zero mean and bounded variance.

The operator T is a non-expansion in the max-norm:

$$\begin{aligned} T(x)(s) - T(y)(s) &= \frac{1}{\eta} \log \sum_{s'} \mathcal{P}(s'|s) e^{\eta x(s')} - \frac{1}{\eta} \log \sum_{s'} \mathcal{P}(s'|s) e^{\eta y(s')} \\ &= \frac{1}{\eta} \log \frac{\sum_{s'} \mathcal{P}(s'|s) e^{\eta x(s')}}{\sum_{s'} \mathcal{P}(s'|s) e^{\eta y(s')}} \\ &\leq \frac{1}{\eta} \log \frac{\sum_{s'} \mathcal{P}(s'|s) e^{\eta \left(y(s') + \|x-y\|_\infty\right)}}{\sum_{s'} \mathcal{P}(s'|s) e^{\eta y(s')}} \\ &= \frac{1}{\eta} \log e^{\eta \|x-y\|_\infty} \frac{\sum_{s'} \mathcal{P}(s'|s) e^{\eta y(s')}}{\sum_{s'} \mathcal{P}(s'|s) e^{\eta y(s')}} \\ &= \|x - y\|_\infty . \end{aligned}$$

Hence we have

$$\|T(x) - T(y)\|_\infty = \max_s |T(x)(s) - T(y)(s)| \leq \|x - y\|_\infty .$$

We also show the following property of T:

$$\begin{aligned} T(x + c\mathbb{1})(s) &= \mathcal{R}(s) + \frac{1}{\eta} \log \sum_{s'} \mathcal{P}(s'|s) e^{\eta (x(s') + c)} \\ &= \mathcal{R}(s) + \frac{1}{\eta} \log e^{\eta c} \sum_{s'} \mathcal{P}(s'|s) e^{\eta x(s')} \end{aligned}$$

$$= \mathcal{R}(s) + \frac{1}{\eta} \sum_{s'} \mathcal{P}(s'|s) e^{\eta x(s')} + c$$
$$= T(x)(s) + c.$$

Hence it follows that $T(x + c\mathbb{1}) = T(x) + c\mathbb{1}$.

We consider the following ordinary differential equations (ODEs),

$$\dot{y}_t = T^1(y_t) - y_t \tag{A.14}$$
$$\dot{x}_t = T^2(x_t) - x_t. \tag{A.15}$$

Such equations are well-defined since both RHS's are Lipschitz thanks to the properties of f and T.

We complete the proof as a succession of lemmas.

Lemma 20. *Let $\bar{y}$ be equilibrium point of the ODE defined in* (A.14)*. Then $\|y_t - \bar{y}\|_\infty$ is non-increasing and $y_t \to y_\infty$ for some equilibrium point of* (A.15)*.*

Proof. See Lemma 3.1 in Abounadi et al. [2001]. □

Lemma 21. *Equation* (A.15) *has a unique equilibrium at v_∞.*

Proof. See Lemma 3.2 in Abounadi et al. [2001]. □

Lemma 22. *Let $x_0 = y_0$, then $x_t = y_t + z_t\mathbb{1}$ satisfies the ODE $\dot{z}_t = -uz_t + (\rho - f(y_t))$.*

Proof. See Lemma 3.3 in Abounadi et al. [2001]. □

Lemma 23. *v_∞ is the globally asymptotically stable equilibrium for* (A.15)

Proof. See Lemma B.4 in Wan et al. [2021a]. □

Lemma 24. *Equation* (A.13) *converges almost surely $\hat{v}_t$ to v_∞ as $t \to \infty$.*

Proof. See Lemma B.5 in Wan et al. [2021a] and Lemma 3.8 in Abounadi et al. [2001]. □

Therefore, stability and convergence of equation (A.12) is proved.