A Role-Assignment Approach to State Abstraction of Markov Decision Processes

by

Chaoping Guo

B.Sc. Hons., University of Hertfordshire, 2018

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE COLLEGE OF GRADUATE STUDIES

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

(Okanagan)

June 2022

The following individuals certify that they have read, and recommend to the College of Graduate Studies for acceptance, a thesis/dissertation entitled:

A Role-Assignment Approach to State Abstraction of Markov Decision Processes

submitted by Chaoping Guo in partial fulfilment of the requirements of the degree of Master of Science

Dr. Yong Gao, I. K. Barber Faculty of Sciences
Supervisor

Dr. Donovan Hare, I. K. Barber Faculty of Sciences
Supervisory Committee Member

Dr. Bowen Hui, I. K. Barber Faculty of Sciences
Supervisory Committee Member

Dr. Eric Foxall, I. K. Barber Faculty of Sciences
University Examiner

Abstract

This thesis extends the notion of role assignment on graphs to Markov decision processes (MDPs). A role is a categorical property of a state that describes the state's reward and structural position in the MDP. An MDP with a large state space can often be approximately summarized with a small number of roles, which can be used to build an abstract model, enabling much more efficient planning, with the performance loss bounded by the approximation error.

The previously known performance bound is based on the approximation errors indiscriminatively across all actions. We show that such a bound is too loose, and minimizing it does not necessarily lead to the best possible solution; instead, a tighter bound can be used by taking into consideration the policy defined on the roles.

Calculating the exact role similarities between states is very expensive in both memory and time. We develop a group of algorithms, referred to as assignment iteration and assignment update, that find role assignments using the role similarities between states and roles. Our methods are much more efficient, and we show that the state-state similarities are indirectly preserved with a notion of true similarity. Assignment update can also be applied to unknown MDPs with the experience sampled by a reinforcement learning agent. Using neural networks as assigners, it solves continuous-state-space environments such as CartPole and Catcher with sample efficiency comparable to a model-based method.

Lay Summary

Most reinforcement learning agents learn to directly answer questions like "if I observe this state, what action do I take/which action has the highest value". They cannot explain their decisions because they don't know how the environment works. While environment dynamics in detail are often hard to model and hard to use, learning an abstract version of it may be possible.

In this thesis, we use the structural position of a state in the problem as its abstraction, known as the *role*. The performance guarantee of reasoning with roles is examined. We also develop algorithms to efficiently assign states to their roles. The algorithms are further modified to be applied to unknown environments, where the agent learns an assignment by interacting with the environment.

Preface

The research reported in this thesis has been conducted at the University of British Columbia Okanagan to fulfill the graduation requirement of the Master's Degree in Computer Science. The thesis was written entirely by me, with feedback from my supervisor Dr. Yong Gao. Initial results of the work have been presented in TR(U)BC Summer 2020 Workshop on Optimization.

Table of Contents

List of Figures

Summary of Notations

General mathematical and statistical notations:

$f : X \mapsto Y$	a function f that maps set X to set Y
$x := y$	x is defined to be y
$x \leftarrow y$	to set/update the value of x to y
$y \rightarrow x$	same as above, but of the opposite direction
$\mathbb{P}(X = x)$	probability that a random variable X has a value of x
$\equiv$	equivalence relation
$\mathbf{X}$	a capital letter in bold represents a matrix; $\mathbf{X}_{i,j}$ is the element at i-th row and j-th column in $\mathbf{X}$; $\mathbf{X}_{i,\cdot}$ is all the elements in the i-th row, organized as a row vector; $\mathbf{X}_{\cdot,j}$ is all the elements in the j-th column, organized as a column vector;
$\mathbf{x}$	a small letter in bold represents a (column) vector; $\mathbf{x}_i$ is the i-th element in $\mathbf{x}$

Notations for (finite) Markov Decision Processes:

$M = (S, A, R, P, \gamma)$	the original MDP we apply role assignment on
S	the set of states
$s \in S$	is a state
$n = \lvert S \rvert$	number of states in the MDP
$1 \leq i \leq n$	states are by convention indexed with i; s_i is the i-th state
A	the set of actions; shared with the role MDP

$a \in A$	is an action
$R(s,a) \in \mathbb{R}$	is the reward function that outputs a real-valued reward from state s under action a
$\mathbf{r}^{(a)}$	is the vector representation of the reward function; $\mathbf{r}_i^{(a)}$ is the expected reward for state s_i taking action a
$P(s',s,a) \in [0,1]$	probability of moving from state s to state to state s' under action a; $P(s',s,a) = \mathbb{P}(S_{t+1} = s' \vert S_t = s, A_t = a)$
$\mathbf{P}^{(a)}$	is the transition matrix by taking action a; $\mathbf{P}_{i,i'}^{(a)}$ is the probability state s_i transitions to state $s_{i'}$ under action a
γ	discount factor
$\pi : S \mapsto A$	is a deterministic policy
$\pi : (S \times A) \mapsto [0,1]$	is a stochastic policy
$\pi^\star$	is an optimal policy
$V_\pi(s)$	state-value function of state s under policy π
$Q_\pi(s,a)$	action-value function of state s by taking action a, under policy π

Notations for role assignments:

$H = \{C_1, C_2, ..., C_m\}$	a (hard) partition of S is a set of non-empty, disjoint subsets of S satisfying $\bigcup_{i=1}^m C_i = S$
$W = \{C_1, C_2, ..., C_m\}$	is a soft partition of S; $C_j = \{w_{1,j}, w_{2,j}, ..., w_{n,j}\}$, where $w_{i,j}$ is a non-negative value called the weight of state s_i being assigned into class C_j; across the classes, $\sum_{j=1}^m w_{i,j} = 1$ for any i.
$\bar{S}$	an alias for H or W; used as the state set in the role MDP; known as the set of roles
$\bar{s}_j \in \bar{S}$	an alias for the j-th partition class; used as a state in the role MDP

$m = \lvert\bar{S}\rvert$	number of roles
$1 \leq j \leq m$	roles are usually indexed with j
$h : S \mapsto \bar{S}$	is the (hard) assignment function that assigns a state to its role
$w : S \times \bar{S} \mapsto [0, 1]$	is the (soft) assignment weight represented as a function; $w(s, \bar{s})$ is the weight with which state s is assigned to $\bar{s}$
$\mathbf{W}$	assignment weight matrix; $\mathbf{W}_{i,j} = w(s_i, \bar{s}_j)$
$\mathbf{L}$	likelihood matrix matrix; $\mathbf{L}_{i,j}$ is the likelihood that state s_i is assigned to role $\bar{s}_j$

Notations for role MDPs:

$\bar{M} = (\bar{S}, A, \bar{R}, \bar{P}, \gamma)$	the role MDP that is built upon an original MDP and a role assignment
$\bar{R}(\bar{s}, a)$	reward function for the role MDP
$\bar{P}(\bar{s}', \bar{s}, a)$	transition probability between roles
$\bar{\mathbf{P}}^{(a)}$	transition matrix of a role MDP; $\bar{\mathbf{P}}^{(a)}_{j,j'} = \bar{P}(\bar{s}_{j'}, \bar{s}_j, a)$
$\bar{\pi} : \bar{S} \mapsto A$	a policy for the role MDP; $\bar{\pi}^\star$ is an optimal policy for the role MDP

Notations for things connecting the original MDP and the role MDP:

$\tilde{\pi} : S \mapsto A$	the lifted policy defined as $\tilde{\pi}(s) = \bar{\pi}(h(s))$
$\hat{\pi}$	not a real policy, but is used denote value functions that is based on both the original MDP and the role MDP
$\hat{P}(\bar{s}, s, a)$	transition-to-roles function; the probability that the next state's role is $\bar{s}$ when the current state is s, and the action taken is a
$\hat{\mathbf{P}}^{(a)}$	matrix representation of transition-to-roles

Notations for assignment iteration and assignment update:

$d(s, \bar{s})$	distance between state s and role $\bar{s}$
c	concentration parameter
β	balance parameter; β_r scales reward differences and β_v scales value differences
$\Omega[\mathbf{W}]$	lower bound matrix of $\mathbf{W}$; for all role $\bar{s}_j$, $\frac{\mathbf{W}_{x,j}}{\mathbf{W}_{y,j}} \geq \Omega[\mathbf{W}]_{x,y}$
$\Omega[\mathbf{L}]$	lower bound matrix of $\mathbf{L}$; for all role $\bar{s}_j$, $\frac{\mathbf{L}_{x,j}}{\mathbf{L}_{y,j}} \geq \Omega[\mathbf{L}]_{x,y}$

Acknowledgements

I would like to express my gratitude to my supervisor Dr. Yong Gao for his wisdom, guidance, patience and support.

I would also like to thank everyone at the university who has taught me and everyone I have worked with. It has been a wonderful journey I will never forget.

Dedication

This work is dedicated to my family and Yiyu who have always encouraged and supported me.

Chapter 1

Introduction

The last 20 years has seen an explosion in the research and application of reinforcement learning. Combined with the development of deep neural networks, algorithms of deep reinforcement learning (DRL) have achieved super-human performance in many tasks.

The key component learned with these algorithms is typically an action-value function, from which the agent may choose the action that has the highest value, or a policy, i.e., a conditional probability distribution of actions given a state. In one way, these methods may seem a bottom-up approach, where the agent learns how to behave in the environment, or how to react to the stimuli, without understanding how the environment works. This causes obvious shortcomings. Learning an accurate value function or a well-performing policy often requires a large number of trials, and even with an optimal solution, they cannot explain their decision in a reasonable way.

As a comparison, human intelligence consists of both bottom-up and top-down behaviours. We have not only the involuntary actions to respond to quick events, but also the ability to reason with our understanding of the world dynamics when needed. Our reasoning is not based on every detail of the environment, but on abstract concepts that define situations. If AI can also identify situations, perhaps it can learn to reason efficiently.

We summarize the main contributions of this thesis as follows:

1. We generalize the notion of role assignment on vertices in graphs to states in MDPs. We consider that the role of a state best describes its situation because it is defined by what roles it transitions to. An agent can reason using the roles instead of the original states. Under the context of role assignment, MDP bisimulation [GDG03] and MDP homomorphisms [RB02] can be seen as role equivalence. We show that the performance bound using the overall approximation error [RB04] is rather loose, and minimizing it doesn't guarantee the best performance possible. Instead, a tighter bound can be derived by taking into consideration the policy defined on the roles. (Chapter 3)

2. We develop assignment iteration and assignment update methods that

approximately find each state a role. The algorithms are much more efficient than computing the bisimulation metrics [Fer07, FPP04], and the quality of the assignment can be bounded with the notion we refer to as true similarity. (Chapter 4)

3. We extend the assignment update algorithm to use agents' experience sampled through the interaction with the environment and also use artificial neural networks to approximate the assignment. These extensions allow role assignment to be performed on unknown and large MDPs. The performance is tested and benchmarked with other model-free and model-based algorithms. (Chapter 5)

Chapter 2

Background

In this chapter, we introduce the background and prerequisite materials for understanding the discussions of the following chapters.

2.1 AI and Reasoning

One of the key characteristics of classical AI is reasoning. With concepts represented in symbol structures, systems that manipulate the structures following the same rules of human intelligence can reason like humans [NS76, Buc05]. Classical AI was the dominant paradigm in AI research from the mid-1950s to the mid-1990s, with many profound achievements such as the general problem solver (GPS) and expert systems [RN10, Kol82, Nil09].

Despite the great success, classical AI has received many criticisms, especially at the attribution (i.e., the ownership) of intelligence. Although processing symbol structures can perform reasoning, it relies on the intelligence of the designer, rather than that of the system itself. This can be demonstrated by the example of "Chinese Room Argument" [Sea80]. By following a character mapping manual, a person who has no knowledge of the Chinese language can process Chinese characters as if they understood the language. The scenario has no essential difference from a machine translator, in which an input-to-output mapping is supplied as part of the program. Just like the person does not gain knowledge in the Chinese language, the machine translator should not be considered intelligent, either. Additionally, there are difficulties concerning the scalability of classical AI, in particular how we extract and represent every detail of human intelligence, and how to design the heuristics when the problem itself exceeds human knowledge.

On the other hand, intelligent behaviour may emerge without reasoning. Early examples of non-reasoning AI include Braitenberg's Vehicles [Bra86] and the subsumption architecture [Bro86]. Today's most popular AI paradigm is machine learning, typically with the use of deep neural networks (DNN). Powered by DNNs, deep reinforcement learning (DRL) systems have achieved super-human performance in many tasks, such as the game of GO [SHM+16, SSS+17], and video games[MKS+15]. These methods

are typically model-free and are also considered as non-reasoning methods because the DNNs they use output a value of each action or a probability distribution over the actions.[1]

Still, being able to reason has its advantages. Deep neural networks are already known as black-box models, and more so when combined with model-free reinforcement learning. A learned value function or a policy can choose actions, but cannot explain why such actions are chosen. The explainability provided through reasoning will not only increase users' trust[GMW08], but will also boost its safety[GGR19, BIF+21]. Secondly, reasoning makes it possible to plan ahead for what is likely to happen whereas model-free methods can only respond to given states. Furthermore, reasoning has the potential to improve reinforcement learning performance, especially in sample efficiency and generalizability.

The symbol representation and the rules for symbol manipulation in classical symbolic AI are designed with human knowledge, but learning makes it possible for AI to generate its own symbol system, just like how our ancestors invented language, logic, and mathematics. This is especially interesting in reinforcement learning because the learning agent itself interacts with the environment, and may come up with its own understanding of the problem. So far, the common approaches in this category involve the use of autoencoders, such as latent space planning (LatPlan) [AF18] and embed to control (E2C) [WSBR15]. These methods enable reasoning with the latent variables by discovering the relation between them.

We note that the latent space of an autoencoder is an information bottleneck [TE20, TPB99, AFDM17]. With the loss function defined as the difference between the decoded vector and the original input, the features extracted are encouraged to preserve as much information as possible for full restoration. Although these features may be seen as concepts extracted from the raw input, most of them are irrelevant to decision making. Perhaps a better target for the neural network's output would be an abstraction of the state that encapsulates the state's outcomes under actions.

2.2 Markov Decision Processes

In a sequential decision making problem, a solver is tasked to make a decision based on the current state and the dynamics of the environment. The

[1] Even when a model of the environment is used, its usage is to help train a policy network and/or a value network. Actions are chosen using those networks rather than through planning.

decision determines not only the immediate reward the agent can expect, but also changes the state, on which further decisions need to be made. A Markov decision process (MDP) is the mathematical framework to model such sequential decision making problems. It can be seen as an extension to the Markov chain with added actions on which the transition probability will be conditioned, and with rewards from states by taking actions.

2.2.1 MDP formulation

Throughout the rest of the work, we use the following definition of MDPs. An MDP is a tuple $M = (S, A, R, P, \gamma)$:

- S is a set of states.
- A is a set of actions.
- $R : S \times A \mapsto \mathbb{R}$ is the reward function representing the real-valued expected reward from a state under an action.
- $P : S \times S \times A \mapsto [0, 1]$ is the transition probability function; $P(s', s, a)$ is the probability that state s transitions to state s' under action a.[2]
- γ is the discount factor.

A policy is a collection of decisions at each state. Under a deterministic policy $\pi : S \mapsto A$, i.e., exactly one action is chosen given a state, the value of a state can be expressed by the Bellman equation[BBC57]:

$$V_\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} P(s', s, \pi(s)) V_\pi(s'), \tag{2.1}$$

and similarly for the value of a state-action pair:

$$Q_\pi(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s', s, a) V_\pi(s'). \tag{2.2}$$

A policy may also be stochastic. A stochastic policy is typically represented as probabilities. $\pi_{\text{stochastic}} : S \times A \mapsto [0, 1]$ defines the probability of an action is being chosen at a state: $\forall s \in S, \sum_{a \in A} \pi_{\text{stochastic}}(s, a) = 1$. Note that although a stochastic policy can be optimal, there always exists a deterministic optimal policy. For the rest of this work, we overload the notation π for both deterministic policies (implied by the form $a = \pi(s)$) and stochastic policies (implied by the form $0.3 = \pi(s, a)$).

[2] In this work, we treat the termination of an MDP as transitioning to a special "void" state.

2.2.2 Solving an MDP

If a problem's ultimate goal can be described as to maximize the expected value of the cumulative sum of the reward (the reward hypothesis [SB18]), then for an MDP, the solution is an optimal policy or an optimal value function that implies an optimal policy. By transforming an MDP of a finite state set to a linear program, the solution can be found in polynomial time [d'E63, PT87, LDK95]. The LP maximizes

$$\sum_{s \in S} V_\pi(s), \tag{2.3}$$

subject to

$$\forall s \in S, V_\pi(s) \leq R(s, \pi(s)) + \gamma \sum_{s' \in S} P(s', s, \pi(s)) V_\pi(s'). \tag{2.4}$$

In practice, there are two more widely used iterative dynamic programming methods. Policy iteration [How60] alternates between a policy evaluation step and a policy improvement step. In a policy evaluation step, the value of each state is calculated according to the current policy; in a policy improvement step, the policy is improved according to the latest value function. See Algorithm 1.

Algorithm 1: Policy iteration

$\pi \leftarrow$ some initial policy;
loop
  `/* Policy evaluation */`
  solve the value function $V_\pi(s)$ for all s using the Bellman equation (2.1);
  `/* Policy improvement */`
  compute a new policy: $\pi'(s) = \arg\max_a Q_\pi(s, a)$;
  if $\pi = \pi'$ **then**
    terminate;
  else
    $\pi \leftarrow \pi'$
  end
end

The second dynamic programming method is value iteration [BBC57], where the value of a state is calculated by choosing the greedy action based

on the value of the previous iteration. It is equivalent to calculating the cumulative reward of a finite horizon with one additional time step at every successive iteration. For any discounted MDPs ($\gamma < 1$), value iteration converges in the limit to the unique Bellman optimality equation. Value iteration is summarized in Algorithm 2.

Algorithm 2: Value iteration

initialize $v \leftarrow 0$;
while *termination condition not met* **do**
  update the value function for all s:
  $v(s) \leftarrow \max_a(R(s,a) + \gamma \sum_{s' \in S} P(s', s, a)v(s'))$
end
compute policy: $\pi(s) \leftarrow \arg\max_a(R(s,a) + \gamma \sum_{s' \in S} P(s', s, a)v(s'))$

For most of the problems, policy iteration and value iteration methods are much more efficient than the linear programming method [LDK95]. They approach the optimal solution very quickly, thus a relatively small number of iterations are often enough to produce a good approximate solution. Additionally, since the calculation at every iteration relies only on the results from the previous iteration, the update for the policy and value function on each state and state-action pair can easily be parallelized within an iteration.

2.2.3 Partially observable Markov decision processes

A partially observable Markov decision process (POMDP) is an MDP where states cannot be directly accessed[KLC98]. Instead, the environment returns observations, with which agents make decisions. An POMDP is typically formulated as a tuple of $(S, A, R, P, \Omega, O, \gamma)$. In addition to elements of an MDP, it also includes observations.

- Ω is a set of observations.
- $O : S \times \Omega \mapsto [0, 1]$ is the function defining the probability of a state emitting an observation.

The simplest form of POMDP is called the *known* POMDP, which exposes all the dynamics to the agent, hiding only the underlying states that generate the observations. It is then the agent's job to infer which state the environment is currently at, and use that *belief* to make a decision. Defining a known POMDPs requires expert knowledge of the environment

to accurately specify the structure and parameters, which can sometimes be infeasible.

A more general form is the *unknown/uncertain* POMDP[MPKK99, JJN22], where the knowledge of transitions between states and/or the rewards is absent or imperfect. Common approaches to solving uncertain POMDPs often involve Expectation-Maximization algorithms that learn both the model parameters and the belief of which state an observation comes from[TM13].

The belief is defined as a probability distribution. A belief MDP is built using beliefs as states and expected rewards and transitions under the belief as the reward set and transition set. A solution to a POMDP can be seen as an optimal policy defined on the belief MDP. Due to the continuity of the probability distribution, there are an infinite number of belief states even if the original state space in the POMDP is finite. As a result, exact solutions are intractable even for finite-horizon, and often undecidable for infinite-horizon [MHC99]. Fortunately, there is a wide range of tractable ways to produce approximate solutions.

2.3 Reinforcement Learning

Although many real-world interesting problems can be modelled as MDPs, linear programming or dynamic programming methods cannot be directly applied because of two obstacles. First, the number of states may be too large. One of the common causes is that a state is defined by a combination of sub-states. The number of sub-states doesn't have to be large, and each sub-state might only have a few possible values, but coming together, the state space of the MDP is exponentially large. This issue is known as the curse of dimensionality [BBC57]. Also, the defining sub-states may be continuous, leading to an infinite number of states. The second obstacle is that the MDP may be unknown. An agent starts with no information of the reward function and transition probability.

Reinforcement learning (RL) is the group of techniques that are used to solve problems with a large and/or unknown MDP. Many RL algorithms can be seen as the generalizations of dynamic programming methods. The main differences between RL and explicit planning on an MDP are summarized as follows:

- without knowing beforehand states' transition and rewards, RL samples the dynamics through the agents' interaction with the environment;

– instead of working directly with the components of Bellman equations on individual states, RL makes approximations on parts of them.

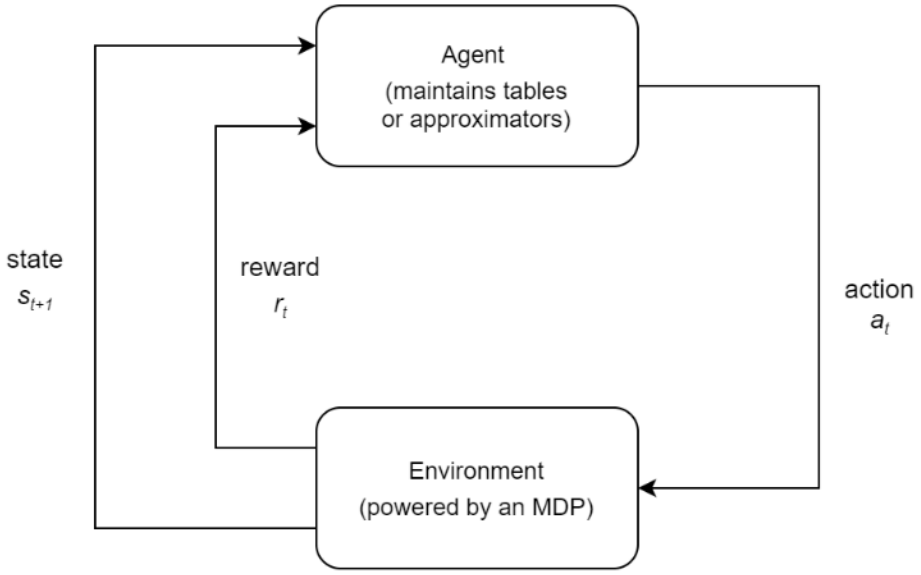


Figure 2.1: Reinforcement learning

2.3.1 Learning from experience

A piece of experience from the agent-environment interaction is a tuple of (s_t, a_t, r_t, s_{t+1}) which is a faithful sample of the reward and the transition of state-action pair (s_t, a_t). If the policy is unchanged, the discounted sum of a sequence of rewards is a faithful sample of the value at the starting state under the said policy. In RL algorithms, samples can be used to learn a value function, a policy or both.

With value-based RL algorithms, the value of a state-action pair is updated using the sampled rewards and the estimated value of the sampled next state:

$$Q(s_t, a_t)_{\text{new}} \leftarrow Q(s_t, a_t)_{\text{old}} + \alpha(r_t + \gamma V_{\text{old}}(s_{t+1}) - Q_{\text{old}}(s_t, a_t)), \qquad (2.5)$$

where $\alpha \in (0, 1]$ is the learning rate; $V_{\text{old}}(s_{t+1})$ is the value estimation for the next state, and it can be either an on-policy value (known as the *SARSA* methods, where the action used for the value estimation of s_{t+1} is chosen by the current policy) or an off-policy greedy value (known as the *Q-learning*

methods, where the highest action-value of s_{t+1} is chosen, regardless of the current policy that is used to generate the sample).

Policy-based algorithms originate from a simple idea: if the action-values at a state are known for all the actions, then we can improve a stochastic policy by increasing the probability of high-valued actions. The policy gradient theorem [SMSM00, SB18] states that for any differentiable (stochastic) policy π_θ, the gradients for its value function $J(\theta)$ can be calculated as

$$\nabla J(\theta) \propto \sum_{s \in S} \mu_\pi(s) \sum_{a \in A} Q_\pi(s,a) \nabla \pi_\theta(a,s), \tag{2.6}$$

where μ is the on-policy distribution, i.e., $\mu_\pi(s)$ is how often state s is visited under π. Since $Q_\pi(s,a)$ can be sampled with π, the gradients can be estimated. A small step of gradient ascent improves the current policy:

$$\theta \leftarrow \theta + \alpha \nabla J(\theta). \tag{2.7}$$

Actor-critic algorithms learn both a value function and a policy. Typically, the action-value estimation in the policy gradient (2.6) is replaced by a learnable value function.

2.3.2 Exploration-exploitation trade-off

Ideally, an agent should act greedily at each state, i.e., choose the action that has the highest expected value. However, without direct access to the true transition probabilities, rewards or value function, the agent only has estimates based on experiences. If the estimation isn't accurate, especially when an actually good action is estimated to give a lower value, always acting greedily prevents it from learning the optimal solution. This dilemma is commonly known as the exploration-exploitation trade-off or the multi-armed bandit problem [KP14].

It is a common practice for an agent to start with an exploration policy, e.g., choosing actions at random or the ones that are likely to result in less tested states, then as the estimates become more reliable, the policy gradually shifts to a greedy one. A policy is called "*greedy in the limit of infinite exploration*" (GLIE) if it satisfies the two conditions:

1. if a state is visited infinitely many times, then each action in that state is also chosen infinitely many times, and

2. as the number of interactions goes to infinity, it converges to the greedy policy.

A GLIE policy guarantees optimality in the sense that all state-action pairs are sufficiently sampled before it chooses the greedy action.

ϵ-greedy

ϵ-greedy is the simplest way to keep exploration. It chooses an exploration parameter $\epsilon \in (0, 1)$, which is the probability that a random action is picked. With probability $1 - \epsilon$, the greedy action is picked.

The parameter ϵ can be either fixed or decaying with a schedule. With certain schedules, such as $\epsilon_t = \frac{c}{t}$, the policy is GLIE. In practice, using a non-GLIE ϵ-greedy strategy is often sufficient to produce a decent result.

Boltzmann (softmax) policy

Boltzmann policy decides the probability of each action according to their value:

$$\pi^{(t)}(s, a) = \frac{\exp(Q_\pi^{(t)}(s, a)/\tau)}{\sum_{a'} \exp(Q_\pi^{(t)}(s, a')/\tau)}, \tag{2.8}$$

where τ is a decreasing temperature parameter. At the beginning of the exploration, τ is set to a large positive value, so that differences in action-values have a relatively small impact on the probabilities. As $t \to \infty$, $\tau \to 0$, and eventually the probability for the action with highest value converges to 1.

Upper confidence bound

Upper confidence bound method (UCB) chooses an action not only based on the sampled value, but also a bonus term. Formally, the action is decided as

$$\pi_{\text{UCB}}(s) = \arg\max_a \left(Q(s, a) + c\sqrt{\frac{2\log(t_s)}{T_{s,a}}} \right), \tag{2.9}$$

where t_s is the total times that state s has occurred so far, and $T_{s,a}$ is the number of times a has been chosen for the state.

The bonus has two properties:

- The less chosen actions will have a greater bonus, giving them better chances to be explored.
- The average bonus for all actions diminishes over time, so the choices are increasingly greedy.

2.3.3 Function approximation

In real-world reinforcement learning problems, the state space is usually too large and often continuous, so tabular representation of the value or the policy for each state is impossible. On the other hand, as empirical speculation, if the variables that define the two states have largely the same or similar values, with only very minor differences, their values and best actions should also be the same or very similar. To scale reinforcement learning up to solve more interesting problems, parameterized methods can be used to approximate the value function and/or the policy[SB18].

When learning value functions, parameter learning is similar to supervised learning, where for every input, there is an output given as target, and the loss function is defined based on the differences between the targets and the approximations[SB18]. The fundamental difference from supervised learning is that the targets are changing over time because they are calculated using the changing value approximation itself, or because they are sampled from a policy that is being updated.

When learning a policy, the policy gradients can be directly used for adjusting parameters (known as the *all-action* method). For an individual action a_t sampled under the current policy, the action-value or the advantage $G_\pi(s_t, a_t)$ can be estimated, and the gradients may be derived as

$$\nabla J(\theta) = G_\pi(s_t, a_t) \frac{\nabla \pi_\theta(s_t, a_t)}{\pi_\theta(s_t, a_t)}. \tag{2.10}$$

Equivalently, we can define a policy loss function for the sampled action as the negative product of the action's log-probability and its value/advantage[SB18]:

$$\mathcal{L} = -G_\pi(s_t, a_t) \log \pi_\theta(s_t, a_t). \tag{2.11}$$

In practice, artificial neural networks are the most common approximation method used for reinforcement learning. Their strength in dealing with non-linearity and handling a large amount of data are advantageous in learning the value function and the policy.

2.3.4 Model-free vs model-based

From experience, an agent can also learn a dynamics model, i.e, the reward function R and the transition probabilities P. RL Algorithms that learn such models are called model-based [Jan20, WBC+19].

One of the strongest advantages model-based methods have over model-free methods is sample efficiency. There are two main reasons behind it.

First, the training data are always ground truth. The transition probabilities and the reward function sampled from the experience are independent of the exploration policy and the estimated value function. Second, a model encapsulates more information about the environment than just a single value function or a single policy.

A model can be learned for analytical decision-making with methods from the theory of optimal control. For example, by using a linear approximator for transition and a quadratic approximator for rewards, an optimal action sequence that leads to a target state can be calculated with the linear quadratic regulator (LQR) [Abb15, TET12].

The ability to predict state transition and rewards enables sample-based planning. Random shooting algorithms [Ric05, NKFL18] generate random action sequences as candidates, and use the learned model to evaluate the candidates so that the action-values at the current state can be estimated. For problems with discrete action space, performing Monte Carlo tree search is also made possible.

Apart from direct decision-making, a model can also be used to generate imaginary roll-outs, which is then used as training data for model-free algorithms [KBM+20, Sut91, JFZL19]. This is typically useful to improve the sample efficiency of the associated model-free methods.

The disadvantage of model-based methods is that their performance heavily relies on the accuracy of the model, which is especially challenging in high-dimensional problems. When used for multi-step predictions and roll-outs, the error compounds, limiting the length of the horizon [AML18].

2.3.5 Mixed open-loop and closed-loop control

The feedback-control loop most reinforcement learning algorithms have focused on is known as *closed-loop*, where the agent (the controller) makes a one-step decision for the current state. It is assumed that the state is observable in real-time without any cost. However, this might not be the case for many problems. For example, the sensors on a robot may be energy-expensive, so that keeping them on greatly reduces the sustainability of the robot; or the robot may be remotely controlled by a computer, where the data transmission has a significant delay. For these cases, it is desirable to learn a policy as a *mixed open-loop and closed-loop controller*, which outputs an action sequence of some finite length before it observes the latest state.

Let π_l be a mixed open-loop and closed-loop policy with a fixed length l: $\pi_l(s) = \{a_0, a_2, ..., a_{l-1}\}$. The Bellman equation for the states and action sequences are similar to the ones for single actions (see (2.1) and (2.2)):

$$V_{\pi_l}(s) = R(s, \pi_l(s)) + \gamma^l \sum_{s' \in S} P(s', s, \pi_l(s)) V_{\pi_l}(s'), \tag{2.12}$$

and

$$Q_{\pi_l}(s, \{a_0, a_2, ..., a_{l-1}\}) = R(s, \{a_0, a_2, ..., a_{l-1}\}) + \gamma^l \sum_{s' \in S} P(s', s, \{a_0, a_2, ..., a_{l-1}\}) V_{\pi_l}(s'), \tag{2.13}$$

where $R(s, \{a_0, a_2, ..., a_{l-1}\})$ is the expected sum of discounted reward from the action sequence and $P(s', s, \{a_0, a_2, ..., a_{l-1}\})$ is the probability of the state after the execution of the action sequence.

It can be quite challenging for model-free reinforcement learning algorithms to learn a mixed open-loop and closed-loop policy because the value function or the policy must be learned over the exponentially growing action sequences [Pre00, HBZ96]. In a model-based approach, however, planning for multiple steps is easy because it can be formulated as a special POMDP problem [Mon82], thus approximate planning methods can be applied. Additionally, there is no need for separate models to perform mixed open-loop and closed-loop control—we only need to learn the dynamics model once, and it can be used for both closed-loop control and mixed open-loop and closed-loop control for any finite length.

2.4 Role Assignments on Graphs

One of the fundamental notions in social network analysis is the *positions* of individuals, which are commonly referred to as the *roles* [WF+94, Ler05, BE92]. The role is a property of a node, but unlike node attributes, it is defined by how the node is connected with others. For example, if a human social network is defined by using the "giving-birth-to" relation as edges, then two siblings in a family can be thought of as playing the same role because they share a common mother. The definition doesn't have to be restricted within each family, as the very words "children" and "mother" also define roles—children from different families are of the same role because every one of them is connected with some nodes who has a mother role.

2.4.1 Graph role equivalences

Role assignment on graphs has a generic definition:

Definition 2.1 (Role assignment on graphs)**.** Given a graph $G = (V, E)$, a role assignment on G is a partition of the vertices $r : V \mapsto W$, such that any pair of vertices from the same partition block have equivalent neighborhood, i.e., for all $u, v \in V$,

$$r(u) = r(v) \implies N(u) \equiv N(v).$$

$N(u)$ and $N(v)$ are the neighbourhoods of u and v, respectively. The equivalence between neighbourhoods can be defined in multiple ways. The most common ones are structural equivalence[LW71, Bur76], automorphic equivalence[Eve85, Win88] and regular equivalence[WR83, BE89].

Structural equivalence is the most strict role equivalence, which is based on the idea that a node's position is defined by which nodes it is connected with:

Definition 2.2 (Structural equivalence)**.** Let $G = (V, E)$ be a graph. Two nodes $u, v \in V$ are said to be structurally equivalent if they have the same neighbors, i.e.,

$$N(u) = N(v).$$

A less strict form of role equivalence may be described as the interchangeability between nodes. If switching labels between nodes doesn't change the network structure, then any pair of nodes that are being switched should have an equivalent role. A pairing pattern of the nodes is known as a graph *automorphism*, and thus the equivalence is known as *automorphic equivalence*.

Definition 2.3 (Graph isomorphism and automorphism)**.** Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs. An isomorphism σ from G_1 to G_2 is a bijection between the vertice sets from two graphs $\sigma : V_1 \leftrightarrow V_2$, such that for all $u, v \in V_1$,

$$(u, v) \in E_1 \iff (\sigma(u), \sigma(v)) \in E_2.$$

An automorphism is an isomorphism between a graph and itself.

Definition 2.4 (Automorphic equivalence)**.** Let $G = (V, E)$ be a graph. Two nodes $u, v \in V$ are said to be automorphically equivalent if there exists an automorphism σ of G such that $\sigma(u) = v$.

Automorphic equivalence can be found from nodes regardless of their distance, but sometimes is still too restricted for many problems, especially when the position we are interested in is more about a node's "functionality" and less about its "importance". For example, despite that two mothers may

have different numbers of children, they should still be considered the same role. *Regular equivalence* is used to describe the equivalence relation between the two nodes whose neighbourhoods consist of the same roles.

Definition 2.5 (Regular equivalence)**.** Let $G = (V, E)$ be a graph. Let $r : V \mapsto W$ be a partition on V. Two nodes $u, v \in V$ are said to be regularly equivalent if the same roles appear in $N(u)$ and $N(v)$, i.e., if

$$\{r(u')|u' \in N(u)\} = \{r(v')|v' \in N(v)\}.$$

The three role equivalences have a clear hierarchy: a set of automorphic equivalences are also a set of regular equivalences;[3] a set of structural equivalences are also a set of automorphic equivalences and are thus also a set of regular equivalences. The opposite direction is not necessarily true.

2.4.2 Role graphs

Once a role assignment is discovered on a graph, a *role graph* can be built using the roles as nodes, and its edges between roles are decided by the edges of the nodes from the nodes that are assigned to the roles.

[Ler05] uses a simple definition, where two roles are connected if there exist connected nodes playing these roles. Formally,

Definition 2.6 (Role graph [Ler05])**.** Let $G = (V, E)$ be a graph and $r : V \mapsto W$ a role assignment. The role graph $R = (W, F)$ is the graph with vertex set W (the set of roles) and edge set $F \subseteq W \times W$ defined by

$$F := \{(r(u), r(v)); \exists u, v \in V, (u, v) \in E\}.$$

Note that under this definition, if the original graph is undirected, the role graph will also be undirected. For role assignments based on regular equivalence, the definition preserves the relation over the roles. However, it doesn't include the information of the degree or how many neighbours are playing each role, which may be useful for structural and automorphic equivalences. For example in a two-layered tree graph, i.e., a star, all leaves are assigned to the same role, while the root has a different one. The role graph using the above definition is a complete graph with two nodes. The

[3] This is a commonly adopted claim [HR05], but simplified. Since there can be multiple automorphisms for a graph, it is more accurate to say that the automorphic equivalences from an automorphism are also regular equivalences. In other words, for automorphically equivalent nodes to be regularly equivalent, the equivalence partition must be consistent. See more details in [EB94, Ler05]

information of the root's degree is lost. A simple way to include the additional information is using a directed role graph even if the original graph is undirected, such that the number of neighbours that play each role can be represented as an attribute on that directed edge.

A role graph can be seen as a smaller model of the original graph. It provides useful insights into the operations and analysis of the graph. For example, any path in the original graph has a corresponding path in the role graph, where the role of each node on the path is also on the corresponding path. If automorphic equivalence is used, the expected node's role of a random walk can be predicted using a random walk on the role graph.

2.4.3 Approximate role assignments

For empirical graph data collected from real-world problems, role equivalences are rarely used because of two reasons: computational intractability and the rare existence of non-trivial partitions [FDB11]. Therefore, it is common to relax the equivalence conditions to similarities so that the role assignments no longer have to be "exact", but rather "approximate" [HR05, BL10, LHN06].

Typically, the relaxations change the binary relation—equivalent or not equivalent—to a real-valued similarity measure in $[0, 1]$. There are several ways to measure the similarity, such as the Jaccard index and the cosine similarity. Under a partition, if all vertices from the same partition block have a similarity of 1, for all blocks, then the partition is an exact role assignment. If equivalence cannot be achieved between all pairs of the same role, as long as they have high similarity, the partition still makes a good approximation.

Approximate role assignment enables a different approach to the problem. Instead of searching for a role assignment without the knowledge about its existence, many methods [RW07, BDF92, DBF05] are developed to find approximate role assignments with a given number of roles m, such that the similarities between vertices of the same roles are high. This approach transforms an assignment problem into an optimization problem by defining an objective function that measures the overall similarity, and attempting to maximize it through the assignment function and sometimes also the edges in the role graph. Although finding the solution with the highest overall similarity may still be hard, near-optimal ones can be found efficiently.

Chapter 3

Role Assignment on MDPs

If the details of the MDP are known, an AI solver can theoretically plan its behaviour by evaluating the rewards it can collect and the future states it may reach. However, this logical and sensible reasoning is subject to the number of states, and thus often becomes impractical as the size of the state space grows.

Human intelligence also makes use of planning by considering the future ramifications, but in a different way. Instead of planning on the states themselves, we make sense of the states to form *concepts* that best explain the *situation*, and plan according to the possible outcomes of the situation. Since we only use problem-related concepts, the number of situations becomes so much smaller than that of the raw states.

What is a situation? We consider that a situation is similar to a role in a graph, which is a property shared by states with a similar structural position and functions. The states may often be different, but as long as the situation is the same, taking the same actions is expected to have the same consequences.

It is desirable to give AI the ability to understand situations because it is the key to efficient reasoning. In this chapter, we discuss the role assignment problem for states in MDPs.

In section 3.1, we review the work on MDP state abstraction [BD94, LWL06, GW92, Abe19]. We first discuss the exact state abstraction based on MDP bisimulation [GDG03] and MDP homomorphism [RB02], and then for the approximate state abstraction, we discuss the metrics[FPP04, FPP11, FPP05] and the performance guarantee for approximate state abstractions [Whi78, FPP04, TPP08, RB04].

In section 3.2, we show that by representing an MDP as a graph, an exact state abstraction is a role assignment. Bisimulation as a role equivalence is stronger than regular equivalence and weaker than automorphic equivalence. The role graph based on the role assignment is the abstract MDP which we refer to as the role MDP.

In section 3.3, we show that by taking into consideration the policy and value function in the role MDP, a tighter bound on the loss of performance

can be derived.

In section 3.4, we consider a more general type of role assignment referred to as the soft role assignment, where each state is assigned to not a single role, but to all roles with weights that sums up to 1.

3.1 State Abstraction

3.1.1 Overview

For many interesting problems, planning on the MDPs is impractical because the size of the state space is too large or continuous. However, we may find similarities among sub-structures, where applying the same policy on states from similar sub-structures gives a similar return. Instead of making decisions on individual states, an optimal policy may then be solved for similar structures all at once, and applied back to the original states.

State abstraction [BD94, LWL06, GW92, Abe19] is the process of creating an MDP with a smaller state space and defining a mapping from the original state space to the smaller state space, such that solving the smaller MDP may help solve the original MDP.

Definition 3.1. A state abstraction of an MDP $M = (S, A, R, P, \gamma)$ is

- an MDP $\bar{M} = (\bar{S}, A, \bar{R}, \bar{P}, \gamma)$ with $|\bar{S}| \leq |S|$ known as the *abstract MDP*, and
- a surjection $h : S \mapsto \bar{S}$ known as the *abstraction function*.

Under this definition, any MDP that shares the action space and has a smaller state space can be an abstract version of the original one. For an abstraction to be useful, it is common to require that the reward function and transition probability in the abstract MDP to be some weighted average of the states that are mapped to them. Specifically,

$$\bar{R}(\bar{s}, a) := \frac{\sum_{s \in h^{-1}(\bar{s})} \sigma(s) R(s, a)}{\sum_{s \in h^{-1}(\bar{s})} \sigma(s)}, \tag{3.1}$$

and

$$\bar{P}(\bar{s}', \bar{s}, a) := \frac{\sum_{s \in h^{-1}(\bar{s})} \sum_{s' \in h^{-1}(\bar{s}')} \sigma(s) P(s', s, a)}{\sum_{s \in h^{-1}(\bar{s})} \sigma(s)}, \tag{3.2}$$

where $\sigma(s)$ is the significance/weight of state s.

Applying a policy $\bar{\pi}$ from an abstract MDP to the original MDP is known as *lifting*, where for any state, the action is chosen according to its

corresponding abstract state under the policy. The lifted policy is denoted as $\tilde{\pi}$, with $\tilde{\pi}(s) = \bar{\pi}(h(s))$.

In [LWL06], 5 types of state abstraction are considered useful in the sense that lifting an optimal policy will also be optimal. In this thesis, we will only focus on the first type, known as the *model-irrelevance* abstraction because it preserves the model dynamics, rather than just the action value, which is essential to reasoning.

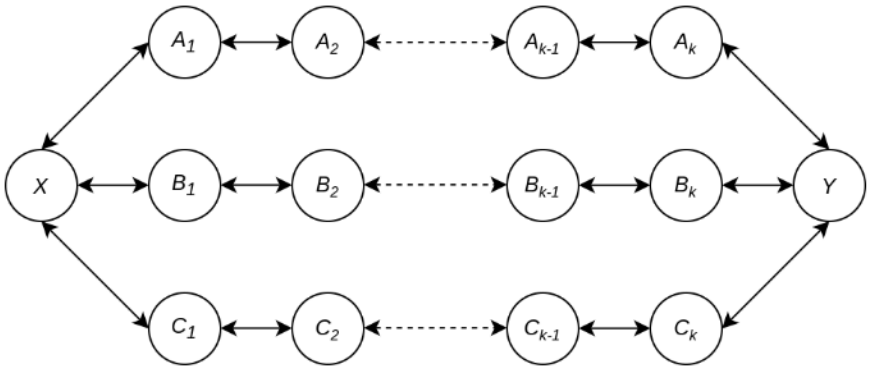
Consider a very simple example shown in Figure 3.1(a):

- an agent starts at location X, and needs to reach target location Y where it would receive positive reward and also terminates;
- the agent may choose one of the two actions, `move-left` or `move-right`;
- between X and Y, there are three paths, each of which consists of k locations that give no reward; locations on each of the paths are denoted as $\mathcal{A} = \{A_1, A_2, ..., A_k\}$, $\mathcal{B} = \{B_1, B_2, ..., B_k\}$, and $\mathcal{C} = \{C_1, C_2, ..., C_k\}$ respectively;
- the first `move-right` action taken at S may take the agent to the beginning location of any of three paths, namely A_1, B_1 or C_1.

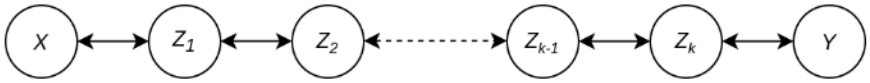
In this example, it is obvious that the three paths are identical, and every location on one path is equivalent to the two corresponding locations on the other two paths: $A_i \equiv B_i \equiv C_i$, because the next steps after taking any action are also equivalent. We can group equivalent locations together, and use a map $h : (\mathcal{A} \cup \mathcal{B} \cup \mathcal{C}) \mapsto \mathcal{Z}$ to indicate which group a location belongs to : $Z_i = h(A_i) = h(B_i) = h(C_i)$. Then using $\mathcal{Z}$ as state set, and aggregating their rewards and transitions, the original MDP can be summarized with a smaller MDP shown in Figure 3.1(b). Solving the smaller MDP gives an optimal policy to the original problem.

3.1.2 Exact state abstraction

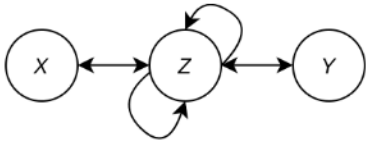
(Stochastic) bisimulation [GDG03] is an equivalence relation between the state spaces from two MDPs. Two states are said to be bisimilar if under any action, they have same reward and same *transition behavior*, which is compared not according to the actual states they transition to, but according to the probabilities of the equivalence classes. The formal definition is as follows.



(a) The original MDP with 3 paths



(b) An abstract MDP with only one path



(c) An abstract MDP with all intermediate states aggregated as one state

Figure 3.1: State space abstraction

Definition 3.2 (MDP stochastic bisimulation [GDG03])**.** Let

$$\bar{M}_1 = (\bar{S}_1, A, \bar{R}_1, \bar{P}_1, \gamma)$$

and

$$\bar{M}_2 = (\bar{S}_2, A, \bar{R}_2, \bar{P}_2, \gamma)$$

be two MDPs with a common action space and a common discount factor. Let $E \subseteq S_1 \times S_2$ be a relation. For any state $s \in S$, let s/E denote the equivalence class of s under the reflexive, symmetric, transitive closure of E restricted in S. $\hat{P}$ is defined to be the total probability that the next state is in an equivalent class:

$$\hat{P}(C, s, a) := \sum_{s' \in C} P(s', s, a).$$

Then E is a stochastic bisimulation if for each $s_1 \in S_1$ and $s_2 \in S_2$, whenever $E(s_1, s_2)$, for all actions $a \in A$:

- $R_1(s_1, a) = R_2(s_2, a)$, and
- for $(s_1' \in S_1, s_2' \in S_2)$ s.t. $E(s_1', s_2')$, $\hat{P}_1(s_1'/E, s_1, a) = \hat{P}_2(s_2'/E, s_2, a)$.

Bisimulation can be defined on one MDP's state space with itself, i.e., $E \subseteq S \times S$. The equivalence classes from E induce a partition and can be used to produce a state abstraction for the MDP, where the abstraction function is the surjection that maps a state to its equivalence class (also known as "block") and the blocks are used as the abstract states. Since bisimilar states have the same reward and same probabilities transitioning to blocks, an abstract state must have the same reward and the same transition behaviour to any state that belongs to the block regardless of the averaging weights. Therefore, we call the state abstractions based on MDP bisimulation "exact".

Exact state abstraction has an alternative characterization from the abstract MDP's viewpoint:

Definition 3.3 (Exact state abstraction for an MDP)**.** An exact state abstraction of an MDP $M = (S, A, R, P, \gamma)$ is a combination of an abstract MDP $\bar{M} = (\bar{S}, A, \bar{R}, \bar{P}, \gamma)$ with $|\bar{S}| \leq |S|$, and an abstraction function $h : S \mapsto \bar{S}$ that satisfy for all $s \in S$ and $a \in A$:

- $R(s, a) = \bar{R}(h(s), a)$, and
- for all $\bar{s}' \in \bar{S}$, $\sum_{s' : h(s') = \bar{s}'} P(s', s, a) = \bar{P}(\bar{s}', h(s), a)$.

It is useful to represent the conditions of an exact state abstraction in matrix form. Let $n = |S|$ be the number of states and $m = |\bar{S}|$ be the number of equivalence classes. Let $\mathbf{H}$ be an $n \times m$ state abstraction matrix, where

$$\mathbf{H}_{i,j} = \begin{cases} 1, & h(s_i) = \bar{s}_j; \\ 0, & \text{otherwise.} \end{cases}$$

Let $\mathbf{P}^{(a)}$ be the transition matrix representation of P under action a, such that $\mathbf{P}^{(a)}_{i,i'} = P(s_{i'}, s_i, a)$. Let $\mathbf{r}^{(a)}$ be the states' rewards represented in a column vector, such that $\mathbf{r}^{(a)}_i = R(s_i, a)$. $\bar{\mathbf{P}}^{(a)}$ and $\bar{\mathbf{r}}^{(a)}$ are defined in the same way for the transitions and rewards in the abstract MDP. We use $\mathbf{1}$ to represent a column vector of all 1s. By definition, the elements of $\mathbf{H}$ are binary. Then, an exact state abstraction is a matrix $\mathbf{H}$ that satisfies the following equations:

$$\begin{aligned}
&\left.\begin{aligned} \mathbf{r}^{(a_1)} &= \mathbf{H}\bar{\mathbf{r}}^{(a_1)} \\ &\vdots \\ \mathbf{r}^{(a_k)} &= \mathbf{H}\bar{\mathbf{r}}^{(a_k)} \end{aligned}\right\} && \text{(reward condition)} \\
&\left.\begin{aligned} \mathbf{P}^{(a_1)}\mathbf{H} &= \mathbf{H}\bar{\mathbf{P}}^{(a_1)} \\ &\vdots \\ \mathbf{P}^{(a_k)}\mathbf{H} &= \mathbf{H}\bar{\mathbf{P}}^{(a_k)} \end{aligned}\right\} && \text{(transition condition)} \\
&\left.\begin{aligned} \mathbf{H}\mathbf{1}_{(n)} &= \mathbf{1}_{(m)} \end{aligned}\right\} && \text{(a state maps to one abstract state)}
\end{aligned} \tag{3.3}$$

This matrix form may be seen as an extension to the matrix characterization of Markov chain lumpability [Kem76] by taking into account also the equivalences of rewards and the multiplicity of actions.

For an MDP M and its abstract version $\bar{M}$, it is possible for a policy directly defined on M to output different actions for equivalent states, but a lifted policy from $\bar{M}$ can have only one action for all states in the same block. Despite such limitations, if the abstraction is exact, the value function for a policy in the abstract MDP equals to the value function for the same policy lifted to the original MDP. More importantly, lifting an optimal policy gives an optimal policy (Theorem 7 in [GDG03]).

It is worth mentioning another line of work under the name of MDP homomorphism [RB02]. The definition of MDP homomorphism includes a surjection from one MDP's state space to another of a smaller size known

as the "homomorphic image". A state in the homomorphic image perfectly summarizes the rewards and transition of the original states that are mapped to it, implying the equivalence between them. Compared to bisimulation, MDP homomorphisms also include the equivalence of actions that give identical outcomes, so the abstraction is performed on state-action pairs. Lax bisimulation[TPP08] relates stochastic bisimulation with MDP homomorphisms by incorporating the equivalence of actions.

3.1.3 Model minimization

An exact state abstraction produces a perfect abstract model of the original MDP. Its efficiency when used for planning, like any other MDPs, directly depends on its size. Model minimization with exact state abstraction is to find the partition with the fewest equivalence classes. We briefly describe the SPLIT algorithm developed in [GDG03].

Let $H = \{C_1, C_2, ..., C_m\}$ be a partition on S. A block C_j is said to be "*stable*" with respect to block C_k if for all state pairs $(u, v) \in C_j$ it is true that for all actions $a \in A$,

$$R(u, a) = R(v, a) \quad \text{and} \quad \hat{P}(C_k, u, a) = \hat{P}(C_k, v, a).$$

The main procedure of the algorithm, denoted as $\text{SPLIT}(C_j, C_k, H)$, checks the stability of C_j with respect to C_k. If it's not stable, the procedure breaks C_j into sub-blocks $\{C'_{j,1}, C'_{j,2}, ..., C'_{j,b}\}$ such that each $C'_{j,i}$ is a maximal sub-block of C_j that is stable with respect to C_k. Since every $C'_{j,i}$ is maximal, the way of such split is unique, and u and v not being in the same sub-block implies there exists some action a, that

$$R(u, a) \neq R(v, a) \quad \text{or} \quad \hat{P}(C_k, u, a) \neq \hat{P}(C_k, v, a).$$

If a partition H_t satisfies the condition that for any pair of states that are not in the same block, there is no bisimulation between them, then a new partition created by any refining operation, $H_{t+1} = \text{SPLIT}(C_j, C_k, H_t)$, also satisfies the condition (Lemma 8.1 in [GDG03]).

The process starts with an initial trivial partition $H_0 = \{C_1\}$ that has all states in one block. It then iteratively checks the stability of all pairs of blocks under the current partition, breaking one of the unstable blocks at a time to refine the partition. Since H_0 satisfies the above condition, any subsequently refined partition must also satisfy it. The process stops when the partition is stable (every block is stable with respect to every block), and it is the stable partition with the fewest blocks because of the condition.

There are at most linearly many splits of blocks (the number of blocks is limited by the number of states), and at most quadratically many stability checks (checks performing on pairs of states). As a result, the SPLIT algorithm runs in a polynomial of $|S|$ and $|A|$. If an MDP of moderate size is present, SPLIT can partition it efficiently. However, due to the curse of dimensionality, the size of the state space grows exponentially in the number of sub-states that define the states.

If these sub-states are accessible, a state can be directly represented with the sub-states; if not, we can also artificially use a set of (usually binary) variables to represent the states. MDPs with such representations are called "factored MDPs". Many problems described in MDPs are made easier with factored representations [GKPV03, KK99, SDL07] because factoring is exponentially smaller than enumerating, and also because such representation often allows more direct access to the problem structure. However, for model minimization with exact state abstraction, [GDG03] has shown that, by using a factored representation with binary variables, the problem is NP-hard.

3.1.4 Approximate state abstraction

Complexity is not the only reason that makes exact model minimization impractical. For many problems, it is common that states may only find equivalence with themselves; for MDPs with very large or continuous state space, the minimum number of equivalence classes may also be very large, or even infinite.

Naturally, a more useful approach is to relax the requirements of equivalence to similarity. We call a non-exact state abstraction an *approximate state abstraction* if the partition groups *similar* states into blocks and each abstract state has similar outcomes to the states that are mapped to it. The *approximation error* is the general term for the reward differences and aggregated transition differences between states from the same blocks and between states and their corresponding abstract states.

In the example at the beginning of the chapter, the three paths are aggregated into an abstract path that is of the same length. It perfectly reflects the dynamics of the environment, but is also redundant in terms of the agent's decision making. Instead, we can map all states on the paths to a single state Z, then the original MDP can be aggregated into an abstract MDP of 3 states shown in Fig 3.1(c). At Z, `move-left` has some chance to get back to the starting state, and some chance to remain in Z; `move-right` has some chance to reach the target state, and some chance to remain in Z. Despite not being a perfect model in predicting values, lifting the optimal

policy in the abstract MDP produces a policy that is also optimal in the original MDP.

Without the strict requirements for equivalence, the rewards and transitions of abstract states depend on the averaging weights. To ensure the attributes are unique, a simple (unweighted) average is typically used [TPP08, RB04].

3.1.5 Performance bound

The question of interest is what makes an approximate state abstraction perform well in terms of achieving the two goals:

- the value function on the abstract MDP under a certain policy accurately estimates the value function on the original MDP by lifting that policy, i.e., $\|(V_{\bar{\pi}} \circ h) - V_{\hat{\pi}}\|$ is small; and
- the value function of an optimal policy lifted back to the original MDP is close to the value function of an actual optimal policy directly defined on the original MDP, i.e., $\|V_{\pi^\star} - V_{\hat{\pi}^\star}\|$ is small.

The importance of having a small approximation error is quite intuitive. That is, the more similar states of the same blocks are, the better the approximate abstraction is. This idea can be seen in several related studies [Whi78, FPP04, TPP08, RB04], where the differences in the rewards and transitions between states and their abstract copies are measured.

Let $Q_{\hat{\pi}}$ be a special action-value function that uses the actual one-step reward and transition probability for a state-action, but also uses values of abstract states for subsequent values:

$$Q_{\hat{\pi}}(s,a) = R(s,a) + \gamma \sum_{s'} P(s',s,a) V_{\bar{\pi}}(h(s')). \tag{3.4}$$

For a state-action pair (s,a), we interpret the difference between $Q_{\hat{\pi}}(s,a)$ and $Q_{\bar{\pi}}(h(s),a)$ as the *assignment(abstraction) temporal difference.*

Let $K(Q_{\hat{\pi}^\star})$ be the largest assignment temporal difference under an optimal policy[4]:

$$K(Q_{\hat{\pi}^\star}) = \max_{a \in A, s \in S} |Q_{\hat{\pi}^\star}(s,a) - Q_{\bar{\pi}^\star}(h(s),a)|. \tag{3.5}$$

$K(Q_{\hat{\pi}^\star})$ can be bounded as follows:

[4]Note that $\hat{\pi}^\star$ isn't an actual policy in either M or $\bar{M}$, and $Q_{\hat{\pi}^\star}$ is artificially constructed for the purpose of defining $K(Q_{\hat{\pi}^\star})$.

Theorem 3.4 (Bounded assignment temporal difference of abstract MDP (Corollary (b) of Theorem 6.1 from [Whi78]))**.**

$$K(Q_{\bar{\pi}^\star}) \leq K_R + \frac{\gamma}{1-\gamma}\delta_{\bar{R}}\frac{K_P}{2} \tag{3.6}$$

where

$$K_R = \max_{s\in S, a\in A} \left|R(s,a) - \bar{R}(h(s),a)\right|,$$

$$K_P = \max_{s\in S, a\in A} \sum_{\bar{s}'\in\bar{S}} \left| \sum_{s':h(s')=\bar{s}'} P(s',s,a) - \bar{P}(\bar{s}',h(s),a)\right|,$$

$$\delta_{\bar{R}} = \max_{\bar{s}\in\bar{S}, a\in A} \bar{R}(\bar{s},a) - \min_{\bar{s}'\in\bar{S}, a'\in A} \bar{R}(\bar{s}',a').$$

Notice that $\frac{K_P}{2}$ is the maximum total variation distance (TVD) between $\hat{P}(\cdot,s,a)$ and $\bar{P}(\cdot,h(s),a)$, and can also be interpreted as the maximum disagreement between $\hat{P}$ and $\bar{P}$ using the following proposition:

Proposition 3.5 (Proposition 5.2 in [LP17])**.** *Let μ and ν be two probability distributions,*

$$\delta_{\mathrm{TV}}(\mu,\nu) = \frac{1}{2}\sum_{x\in\mathcal{X}} \left|\mu(x) - \nu(x)\right| = \max_{X\subseteq\mathcal{X}} \left|\mu(X) - \nu(X)\right|. \tag{3.7}$$

[Whi78] has developed two important bounds on the value differences that make use of $K(Q_{\bar{\pi}^\star})$:

Theorem 3.6 (Bounded value difference between an optimal policy in the original MDP and an optimal policy in the abstract MDP (Theorem 3.1 in [Whi78]))**.**

$$|V_{\pi^\star}(s) - V_{\bar{\pi}^\star}(h(s))| \leq \frac{1}{1-\gamma}K(Q_{\bar{\pi}^\star}), \quad \forall s\in S \tag{3.8}$$

Theorem 3.7 (Bounded value difference between an optimal policy in the abstract MDP and its lifted policy in the original MDP (Theorem 3.2 in [Whi78]))**.**

$$|V_{\bar{\pi}^\star}(h(s)) - V_{\bar{\pi}^\star}(s)| \leq \frac{1}{1-\gamma}K(Q_{\bar{\pi}^\star}), \quad \forall s\in S \tag{3.9}$$

Both bounds hold for any state s and any discount factor $\gamma < 1$.

By combining the above results, [RB04] has established a bound for the max norm of the performance loss of a lifted optimal policy compared to an actual optimal policy:

Theorem 3.8 (Bounded performance loss of lifted optimal policy [RB04])**.**

$$\|V_{\pi^\star} - V_{\tilde{\pi}^\star}\|_\infty \leq \frac{2}{1-\gamma}\left(K_R + \frac{\gamma}{1-\gamma}\delta_{\bar{R}}\frac{K_P}{2}\right). \tag{3.10}$$

If both K_P and K_R are 0, every two states from the same blocks are equivalent, and the bound indeed indicates no loss of performance. However, this bound itself is extremely sensitive to small changes of the differences, thus is often too loose to accurately reflect a model's performance. With a reasonable number of abstract states, it is often impossible to achieve desirably small K_P and K_R. Later we will also see that the approximate state abstraction with the smallest K_P and K_R does not necessarily give the best possible performance.

3.1.6 Bisimulation metrics

MDP bisimulation [GDG03] is a binary relation, i.e., two states are either bisimilar or not bisimilar. Bisimulation induces a partition based on equivalence, but cannot differentiate the more similar state pairs from the less similar.

The bisimulation metrics [FPP04, FPP11, FPP05] are a class of distances that can be used to measure how far away states are from each other. The primary form of bisimulation metrics can be written as:

$$\delta_{\text{Bis}}(s_x, s_y) = \max_{a \in A}\Big((1-c)|R(s_x,a) - R(s_y,a)| + cT_K(\delta_{\text{Bis}})(\mathbf{P}^{(a)}_{x,\cdot}, \mathbf{P}^{(a)}_{y,\cdot})\Big), \tag{3.11}$$

where $c \in (0,1)$ is the transition discount factor that balances the importance between reward differences and transition differences. T_K is the Kantorovich metric (illustrated in Figure 3.2) on the two states' transition probabilities.

The Kantorovich metric originates from the theory of optimal transport. It measures the optimal total cost of transporting the mass from one distribution to another. Between two states' transition probability distributions

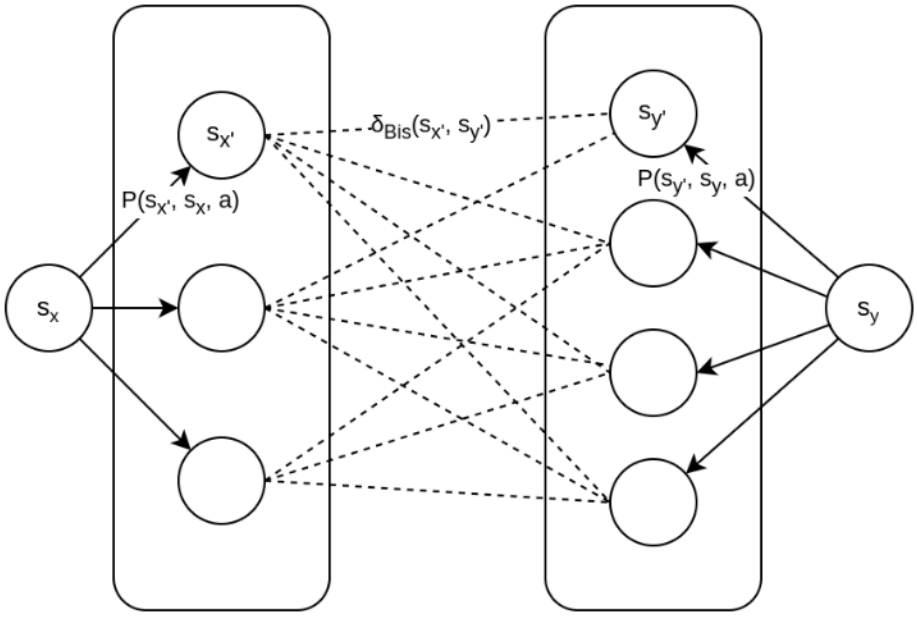


Figure 3.2: Kantorovich metric between two transition probability distributions. It can be viewed as the minimum cost flow from $\mathbf{P}^{(a)}_{x,\cdot}$ to $\mathbf{P}^{(a)}_{y,\cdot}$, where the costs are the distances defined with δ_{Bis}.

μ and ν, its value can be solved with the following linear program:

$$\begin{aligned} T_K(\delta_{\text{Bis}})(\mu,\nu) \leftarrow \min_{\lambda_{k,j}} & \sum_{i=1}^{|S|} \lambda_{k,j}\delta_{\text{Bis}}(s_k,s_j) \\ \text{subject to for every } k, \quad & \sum_j \lambda_{k,j} = \mu(s_k) \\ \text{for every } j, \quad & \sum_k \lambda_{k,j} = \nu(s_j) \\ \text{for every } k,j, \quad & \lambda_{k,j} \geq 0 \end{aligned} \tag{3.12}$$

Notice that bisimulation metrics appear on both sides of equation (3.11). It has been shown that, through repetitively calculating bisimulation metrics using previous bisimulation metrics, the process converges at a fixed point δ^*_{Bis} where pairs of bisimilar states have 0 distance, and the distance smoothly grows for those who are less bisimilar [FPP05].

The main issue with bisimulation metrics is the computation complexity. It requires the metrics for all state pairs to be computed, just like the SPLIT algorithm for bisimulation. Solving the linear program for Kantorovich is also quite expensive, once again due to the large number of state pairs. Each iteration has a complexity of $O\big(|A||S|^4 \log(|S|)\big)$. It is also shown in [FPP11] that, to acquire the distances within an error of ϵ, it needs at most $\lceil \frac{\log(\epsilon)}{\log(c)} \rceil$ iterations. One way to speed up the computation is to remember the solution for each Kantorovich linear program as the minimum cost flow, so at the next iteration, optimization can start on top of the old one [Fer07]. However, the practicality is still limited by the massive size of the state space.

An alternative form of bisimulation metric is defined by replacing the Kantorovich metric with the total variation difference between the two states' transition probabilities to partitions. Bisimulation metric in this form computes much faster, but since the partition is also based on the metric, the results are often unstable [FPP11, FPP04].

3.2 MDP Role Assignment

An MDP can be represented by a combination of a discount factor and a graph which uses the states represented as vertices, and transitions as edges. An MDP-graph is

– **edge-weighted and directed**: the direction and the weight of an edge represent the probability of moving from one state to another;

- **vertex-attributed**: the real-valued attributes on the vertices represent the rewards;
- **multiplex** (i.e., multilayer with the same set of vertices on every layer [KAB+14]): every layer represent an action.

An example of an MDP-graph is shown in Figure 3.3.

It is in this sense that we consider a state in an MDP takes a position and the role assignment of a state is the role assignment of the corresponding vertex in the MDP-graph. The definitions for role equivalences in [EB91, BE92, Ler05] have focused on simple, undirected graphs but can easily be extended to MDP-graphs. Specifically, two vertices in an MDP-graph have an equivalent role if

- they have equivalent vertex attributes (rewards);
- they have equivalent out-neighbours (transitions); and
- the equivalence conditions are satisfied on all layers.

The equivalence of rewards between state s_1 and s_2 is simply $R(s_1, a) = R(s_2, a)$. The equivalence of transitions, like in simple graphs, can be defined in different levels.

The first form of transition equivalence can be seen as the direct adaptation from the structural equivalence for graph vertices:

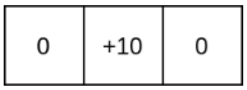
Definition 3.9 (Structural equivalence of transition)**.** Let $M = (S, A, R, P, \gamma)$ be an MDP. From two states $s_1, s_2 \in S$, under an action $a \in A$, their transition $P(\cdot, s_1, a), P(\cdot, s_2, a)$ are structurally equivalent if

$$\forall s' \in S, \quad P(s', s_1, a) = P(s', s_2, a)$$

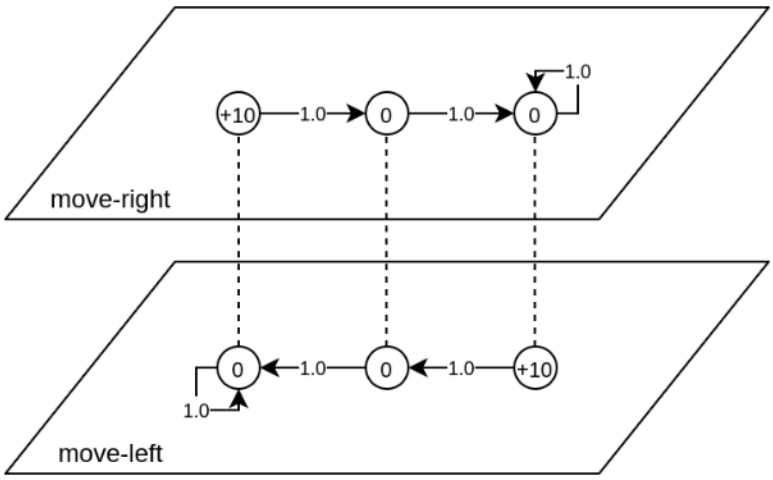
We can also define the isomorphisms and automorphisms for MDPs:

Definition 3.10 (Isomorphisms and automorphisms for MDPs)**.** An isomorphism σ from MDP $M_1 = (S_1, A, R_1, P_1, \gamma)$ to $M_2 = (S_2, A, R_2, P_2, \gamma)$ is a bijection between the two state sets $\sigma : S_1 \mapsto S_2$, such that for any $a \in A$ and $s \in S_1$ we have $R_1(s, a) = R_2(\sigma(s), a)$, and for any $s, s' \in S_1$, we have $P_1(s', s, a) = P_2(\sigma(s'), \sigma(s), a)$. An automorphism is an isomorphism that maps a state set to itself.

Automorphic equivalence of transition requires only the the automorphism between two states' out-neighbors:



(a) MDP of three states



(b) MDP-graph

Figure 3.3: Graph representation of an MDP. The original MDP has three states. `move-left` and `move-right` are the two available actions. Landing on the state in the middle gives a reward of 10 while other states have no reward. The MDP can be represented by a two-layered multiplex, directed, attributed graph.

Definition 3.11 (Automorphic equivalence of transition)**.** Let $M = (S, A, R, P, \gamma)$ be an MDP. From two states $s_1, s_2 \in S$, under an action $a \in A$, their transition $P(\cdot, s_1, a), P(\cdot, s_2, a)$ are automorphically equivalent if there exists an automorphism $\sigma : S \mapsto S$, such that

$$\forall s' \in S, \quad P(s', s_1, a) = P(\sigma(s'), s_2, a)$$

Since an MDP-graph can be treated as a complete graph,[5] all states have regularly equivalent transition because the differences in the probability are ignored. Therefore, the regular equivalence between two states is simply reward equivalence.

Following the same fashion, MDP bisimulation can also be considered a role equivalence because it requires the corresponding vertices of two equivalent states to have the same attributes and equivalent out-neighbourhoods on all layers.

There is a clear hierarchy of the MDP role equivalences: states that are automorphically equivalent are bisimilar; states that are structurally equivalent are automorphically equivalent and thus bisimilar; all the three equivalences satisfy regular equivalence (reward equivalence). The opposite direction is not true. Figure 3.4 illustrates examples of the transition probabilities that satisfy the three equivalence conditions.

The abstract MDP induced by a bisimulation is the same as its role graph. Therefore, we refer to such abstract model as *role MDP*. Formally, we define role MDP as follows:

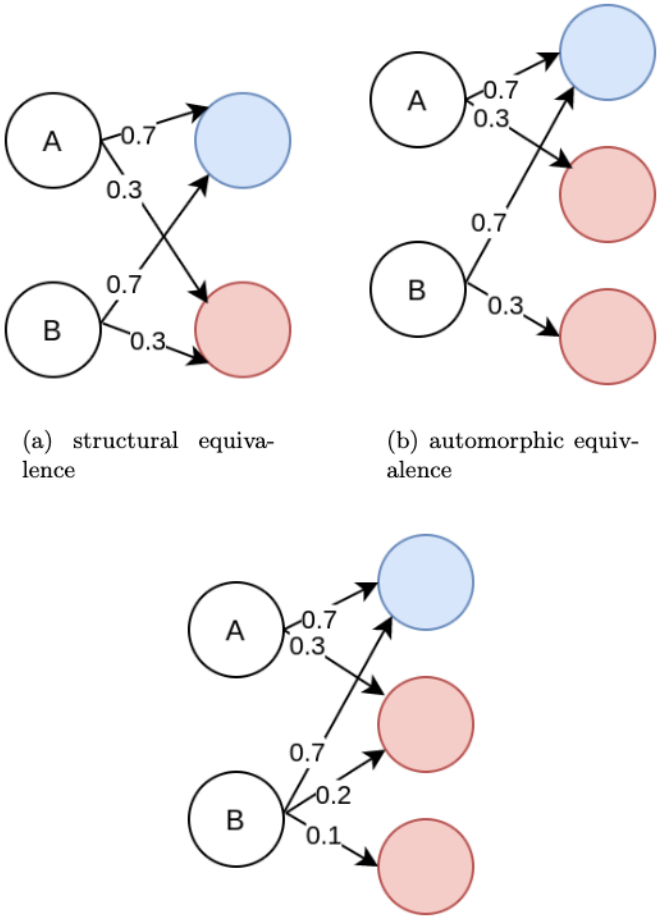
Definition 3.12 ((Exact) role MDP)**.** Let $M = (S, A, R, P, \gamma)$ be an MDP. Let a partition $H = \{C_1, C_2, ..., C_m\}$ be an exact role assignment on S. $\bar{S} = \{\bar{s}_1, \bar{s}_2, ...\bar{s}_m\}$ is an alias for H, referred to as the set of roles. $h : S \mapsto \bar{S}$ is the assignment function that assigns a state to its role. Then, another MDP $\bar{M} = (\bar{S}, A, \bar{R}, \bar{P}, \gamma)$ is the Role MDP of M under H where

- $\bar{S}$ is the set of states,
- A is the set of actions that is shared with M,
- $\bar{R}$ is the reward function that satisfies

$$\forall s \in S, \quad s \in C_j \implies (\forall a \in A)(\bar{R}(\bar{s}_j, a) = R(s, a)),$$

and

[5] There is always an edge for any directed pair of vertices, with the transition probability represented by the weight



(a) structural equivalence

(b) automorphic equivalence

(c) bisimulation

Figure 3.4: Examples of the three levels of role equivalences. State A and state B in (a) are structurally, automorphically, and bisimilar; in (b) are automorphically and bisimilar, but not structurally equivalent; in (c) are only bisimilar. The colours represent roles that are already determined. The numbers on the edges represent the transition probability. Note that the examples only describe the transitions under one action, so for the states to be equivalent, the transitions under other actions, as well as rewards must also satisfy the equivalence conditions.

– $\bar{P}$ is the transition function that satisfies

$$\forall s \in S, \quad s \in C_j \implies (\forall C_{j'} \in H, a \in A)(\bar{P}(\bar{s}_{j'}, \bar{s}_j, a) = \sum_{s' \in C_{j'}} P(s', s, a))$$

It is useful to define the *transition-to-roles* probability function $\hat{P} : \bar{S} \times S \times A \mapsto [0, 1]$:

$$\hat{P}(\bar{s}', s, a) = \mathbb{P}(\bar{S}_{t+1} = \bar{s}' | S_t = s, A_t = a) = \sum_{\{s' | h(s') = \bar{s}'\}} P(s', s, a) \tag{3.13}$$

Then the last condition in the definition is interpreted as "for each state, its role's transition is the same as its transition-to-roles, under any action".

Having a small approximation error is not the only way to achieve good performance when lifting an optimal policy from the abstract MDP. In [LWL06], the conditions for optimality are summarized as five types of abstraction. Now with bisimulation understood as a role equivalence, we can see that exact and approximate state abstractions are a special type because unlike other abstraction options, they do not require knowing an optimal policy or an optimal value function in the original MDP. Rather, the roles are the discovered situations states are in, and the performance depends on how accurate the situation describes the states.

The unique feature of recognizing the situation and planning on the outcomes of the situation resembles how human intelligence performs abstract thinking. This is particularly interesting when the role assignment function is learned as a neural network. Recall from section 2.1 that the latent representations extracted with autoencoders may be seen as concepts that are both relevant and irrelevant to solving the problem because they have to preserve as much information as possible to restore the original input. When the neural networks are used for role assignment, the concepts extracted are only required to describe the situation.

3.3 Policy-Specific Role Assignments

3.3.1 Small error-performance inconsistency

Bisimulation metrics [FPP04, Fer07] are defined with the maximum difference between the two states' rewards and transitions, among all actions. Similarly, the performance loss of approximate MDP homomorphisms [RB04] is bounded using K_R and K_P, which are also the maximums of all state-action pairs.

Indiscriminatively measuring the differences of all state-action pairs gives only a minimum guarantee of the performance, and is too loose to indicate the actual performance when an optimal policy is lifted. This may lead to an undesirable situation—given a number of roles as the budget, the assignment that gives smaller maximum reward/transition differences may perform worse than the one that has larger maximum differences.

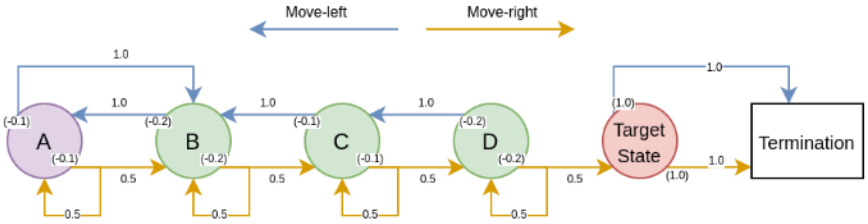
Figure 3.5 shows a simple example of such an undesirable situation. There are 5 states in this example MDP. The agent starts on one of the first four states A, B, C or D, and the goal is to reach the target state, where taking any action gives a positive reward and terminates the MDP. The numbers within parenthesis are the expected rewards from the states; the numbers not in parenthesis indicate the transition probabilities. Since all states except the target have negative rewards, the optimal policy is the one that reaches the target as soon as possible. That is, `move-left` at state A, and `move-right` at state B, C and D.

Suppose we are given a budget of 3 roles, what is the best way to assign the states? One role is reserved for the target state because of its distinct rewards and transitions, leaving us the problem to assign states A, B, C, and D to two roles.

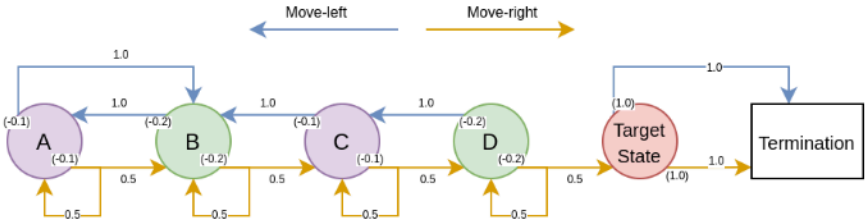
The best performing assignment is shown in 3.5(a). State A is assigned to a role coloured in purple, while B, C and D are assigned to the other role coloured in green. The optimal actions are `move-left` for the purple role, and `move-right` for the green role, which is also optimal when lifted.

The worst performing assignment is shown in 3.5(b). States A and C are assigned to role purple, while B and D are assigned to role green. The optimal actions are `move-left` for the purple role, and `move-right` for the green role. Following the lifted policy at states A, B and C results in an infinite loop between B and C. The value of those states can only be bounded by the discount factor.

Notice that in 3.5(a), the maximum transition difference between states of the same role is at B and C taking action `move-left`, a total variation distance of 1, and the maximum reward difference is also found here at a value of 0.1. Both the differences are greater than those found in 3.5(b), where the maximum transition difference is found between B and D by taking action `move-right`, and the maximum reward difference is 0 for every state-action pair.



(a) Assignment producing optimal lifted policy



(b) Assignment producing worst lifted policy

Figure 3.5: An example of how approximation quality can be inconsistent with performance. The assignment that gives the smallest difference (b) has the worst performance.

3.3.2 Error bound on value estimation with a given policy

We now show that the error of value estimation using a role MDP depends on the reward and transition errors associated with the policy that is being lifted.

In the graph representation, a policy chooses which layer to traverse for every vertex. This can be modelled as the weights on the inter-layer connections, so the whole process is known as a *physical random walk in multiplex* (RWP). With an RWP, the jumping between nodes and jumping between layers take place independently at every time step. An RWP can be reduced to a classic random walk (RWC) [DDSGA13] on a monoplex. The node attributes and edge weights of the monoplex depend on the weights on the inter-layer connections of the multiplex. Therefore, to compare the value difference between estimation using the role MDP and the action value with the lifted policy, we only need to compare the RWC on the two monoplex graphs.

Intuitively, a policy defines a probability distribution of actions (or a single action if it is deterministic) at each state and only the outcomes from the actions are relevant. Let $V_{\hat{\pi}^\star}$ be the value function obtained by restricting $Q_{\hat{\pi}^\star}$ to pairs (s, a) for which $a = \tilde{\pi}^\star(s)$ (the actions are always selected by the lifted policy):

$$V_{\hat{\pi}^\star}(s) = Q_{\hat{\pi}^\star}(s, \tilde{\pi}^\star(s)) = R(s, \tilde{\pi}^\star(s)) + \gamma \sum_{s'} P(s', s, \tilde{\pi}^\star(s)) V_{\bar{\pi}^\star}(h(s')) \quad (3.14)$$

then $K(V_{\tilde{\pi}^\star})$ is a special case of $K(Q_{\tilde{\pi}^\star})$:

$$\begin{aligned} K(V_{\tilde{\pi}^\star}) = &\max_{s \in S} \left| V_{\hat{\pi}^\star}(s) - V_{\bar{\pi}^\star}(h(s)) \right| \\ = &\max_{s \in S} \left| Q_{\hat{\pi}^\star}(s, \tilde{\pi}^\star(s)) - Q_{\bar{\pi}^\star}(h(s), \tilde{\pi}^\star(s)) \right| \end{aligned} \quad (3.15)$$

We can replace $K(Q_{\tilde{\pi}^\star})$ with $K(V_{\tilde{\pi}^\star})$ in Theorem 3.7, and the bound will still hold. In fact, it is true not only for the optimal policy, but also for any policy in the role MDP and their lifted policy. Let

$$\begin{aligned} K(V_{\tilde{\pi}}) = &\max_{s \in S} \left| V_{\hat{\pi}}(s) - V_{\bar{\pi}}(h(s)) \right| \\ = &\max_{s \in S} \left| Q_{\hat{\pi}}(s, \tilde{\pi}(s)) - Q_{\bar{\pi}}(h(s), \tilde{\pi}(s)) \right| \end{aligned} \quad (3.16)$$

then,

Theorem 3.13 (Bounded value difference between an policy in the role MDP and its lifted policy in the original MDP with policy-specific assignment temporal difference)**.**

$$|V_{\tilde{\pi}}(s) - V_{\bar{\pi}}(h(s))| \leq \frac{1}{1-\gamma} K(V_{\tilde{\pi}}), \quad \forall s \in S \quad (3.17)$$

Proof. To show that the inequality holds for all states, we only need to show that it holds for the the state that gives the largest value on the left hand side. Let

$$x = \arg\max_{x \in S} |V_{\tilde{\pi}}(x) - V_{\bar{\pi}}(h(x))|. \quad (3.18)$$

By subtracting and adding $V_{\hat{\pi}}(x)$, we have:

$$\begin{aligned} |V_{\tilde{\pi}}(x) - V_{\bar{\pi}}(h(x))| &= |V_{\tilde{\pi}}(x) - V_{\hat{\pi}}(x) + V_{\hat{\pi}}(x) - V_{\bar{\pi}}(h(x))| \\ &\leq |V_{\tilde{\pi}}(x) - V_{\hat{\pi}}(x)| + K(V_{\tilde{\pi}}). \end{aligned} \quad (3.19)$$

Since the rewards and state transitions that contribute to $V_{\tilde{\pi}}(x)$ and $V_{\hat{\pi}}(x)$ are the same, the only difference between them is how the value for future states are calculated:

$$\begin{aligned} |V_{\tilde{\pi}}(x) - V_{\hat{\pi}}(x)| &= \gamma |\sum_{s' \in S} P(s', x, \tilde{\pi}(x))(V_{\tilde{\pi}}(s') - V_{\bar{\pi}}(h(s')))| \\ &\leq \gamma \sum_{s' \in S} P(s', x, \tilde{\pi}(x)) |V_{\tilde{\pi}}(s') - V_{\bar{\pi}}(h(s'))|. \end{aligned} \tag{3.20}$$

Since x is the state that gives the largest difference between $V_{\tilde{\pi}}$ and $V_{\bar{\pi}} \circ h$, and $\sum_{s' \in S} P(s', x, \tilde{\pi}(x)) = 1$:

$$\begin{aligned} \sum_{s' \in S} P(s', x, \tilde{\pi}(x)) |V_{\tilde{\pi}}(s') - V_{\bar{\pi}}(h(s'))| &\leq \sum_{s' \in S} P(s', x, \tilde{\pi}(x)) |V_{\tilde{\pi}}(x) - V_{\bar{\pi}}(h(x))| \\ &= |V_{\tilde{\pi}}(x) - V_{\bar{\pi}}(h(x))|. \end{aligned} \tag{3.21}$$

Combining the results from above, we have:

$$\begin{aligned} |V_{\tilde{\pi}}(x) - V_{\bar{\pi}}(h(x))| &\leq \gamma |V_{\tilde{\pi}}(x) - V_{\bar{\pi}}(h(x))| + K(V_{\tilde{\pi}}) \\ &\leq \frac{1}{1-\gamma} K(V_{\tilde{\pi}}). \end{aligned} \tag{3.22}$$

□

There is a clear advantage of using $K(V_{\tilde{\pi}})$ over $K(Q_{\tilde{\pi}})$, because instead of using K_P and K_R, the two "global" suprema on the transition differences and reward differences between all state-action pairs, we only need policy-specific suprema $K_{R,\tilde{\pi}}$ and $K_{P,\tilde{\pi}}$ to bound it:

Theorem 3.14 (Bounded policy-specific assignment temporal difference of role MDP)**.**

$$K(V_{\tilde{\pi}}) \leq K_{R,\tilde{\pi}} + \frac{\gamma}{1-\gamma} \delta_{\bar{R}} \frac{K_{P,\tilde{\pi}}}{2}, \tag{3.23}$$

where

$$K_{R,\tilde{\pi}} = \max_{s \in S} |R(s, \tilde{\pi}(s)) - \bar{R}(h(s), \tilde{\pi}(s))|,$$

$$K_{P,\tilde{\pi}} = \max_{s \in S} \sum_{\bar{s}' \in \bar{S}} \left| \sum_{s' \in h(s')} P(s', s, \tilde{\pi}(s)) - \bar{P}(\bar{s}', h(s), \tilde{\pi}(s)) \right|.$$

In general, we say an approximate role MDP is "well-aligned" with an policy if its policy-specific assignment temporal difference bound $K(V_{\tilde{\pi}})$ is small or

its policy-specific suprema $K_{R,\bar{\pi}}$ and $K_{P,\bar{\pi}}$ are small, and "perfectly-aligned" if they are 0.

Additionally, for both $K(Q_{\bar{\pi}})$ and $K(V_{\bar{\pi}})$, the value difference of the states that are not accurately modelled by $\bar{P}$ is bounded using $\frac{1}{1-\gamma}\delta_{\bar{R}}$, which is the difference between the highest possible value i.e., receiving the greatest reward repeatedly, and the lowest possible value i.e., receiving the least reward repeatedly, even if there are no policies that can achieve those values. If a policy is given, all the rewards are based on actions selected by the policy. A tighter bound can be achieved by replacing $\delta_{\bar{R}}$ with policy-specific maximum reward difference $\delta_{\bar{R}_{\bar{\pi}}}$, that is,

$$\delta_{\bar{R}_{\bar{\pi}}} = \max_{\bar{s}\in\bar{S}} \bar{R}(\bar{s}, \bar{\pi}(\bar{s})) - \min_{\bar{s}'\in\bar{S}} \bar{R}(\bar{s}', \bar{\pi}(\bar{s}')).$$

In practice, the action-values of all role-action pairs are solved or easily solvable. It is then useful to also replace $\frac{1}{1-\gamma}\delta_{\bar{R}}$ with policy-specific value difference bound $\delta_{V_{\bar{\pi}}}$:

$$\delta_{V_{\bar{\pi}}} = \max_{\bar{s}\in\bar{S}} V_{\bar{\pi}}(\bar{s}) - \min_{\bar{s}'\in\bar{S}} V_{\bar{\pi}}(\bar{s}').$$

3.3.3 Optimality alignment

An approximate role MDP being well-aligned with an optimal policy $\bar{\pi}^\star$ has a guarantee that its value function makes a good estimate of its lifted policy $\bar{\pi}^\star$ in the original MDP. However, using $K(V_{\bar{\pi}})$ alone is not enough to track the value function of an optimal policy $\pi^\star$ because the optimal policy may select actions that are different from $\tilde{\pi}^\star$. Can we still "align" an approximate role assignment to reach a better performance?

Obviously $\pi^\star$ and $\bar{\pi}^\star$ are not two arbitrary policies. A policy is optimal if it uniformly dominates all other policies. That is to say, these two policies always give the highest value at any state/role. The optimality condition allows us to define the alignment of optimality despite that the two policies may disagree with each other.

We define $K(Q^\dagger_{\tilde{\pi}^\star})$ as an extension to $K(Q_{\tilde{\pi}^\star})$, in which the advantage of choosing the optimal action $\tilde{\pi}^\star(s)$ over a is also subtracted from every value difference between $Q_{\hat{\pi}^\star}$ and $Q_{\bar{\pi}^\star}$ bounded in $K(Q_{\tilde{\pi}^\star})$:

$$\begin{aligned} K(Q^\dagger_{\tilde{\pi}^\star}) = \max_{a\in A, s\in S} \Big[& \big| Q_{\hat{\pi}^\star}(s,a) - Q_{\bar{\pi}^\star}(h(s),a) \big| \\ & - \big(V_{\bar{\pi}^\star}(h(s)) - Q_{\bar{\pi}^\star}(h(s),a) \big) \Big]. \end{aligned} \tag{3.24}$$

$K(Q^\dagger_{\tilde{\pi}^\star})$ can replace $K(Q_{\tilde{\pi}^\star})$ in Theorem 3.6 and the bound will still hold:

Theorem 3.15 (Bounded value difference between an optimal policy in the original MDP and an optimal policy in the role MDP)**.**

$$V_{\pi^\star}(s) - V_{\bar{\pi}^\star}(h(s)) \leq \frac{1}{1-\gamma} K(Q^\dagger_{\bar{\pi}^\star}), \quad \forall s \in S \tag{3.25}$$

Proof. To show that the inequality holds for all states, we only need to show that it holds for the the state that gives the largest value on the left hand side. Let

$$x = \arg\max_{x \in S} V_{\pi^\star}(x) - V_{\bar{\pi}^\star}(h(x)), \tag{3.26}$$

and let $a^\star = \pi^\star(x)$ and $\tilde{a}^\star = \tilde{\pi}^\star(x) = \bar{\pi}^\star(h(x))$. By subtracting and adding two terms $Q_{\hat{\pi}^\star}(x, a^\star)$ and $Q_{\bar{\pi}^\star}(h(x), a^\star)$, we have:[6]

$$\begin{aligned} V_{\pi^\star}(x) - V_{\bar{\pi}^\star}(h(x)) = &\quad Q_{\pi^\star}(x, a^\star) - Q_{\bar{\pi}^\star}(h(x), \tilde{a}^\star) \\ = &\quad Q_{\pi^\star}(x, a^\star) - Q_{\bar{\pi}^\star}(h(x), a^\star) \\ &\quad + Q_{\bar{\pi}^\star}(h(x), a^\star) - Q_{\bar{\pi}^\star}(h(x), \tilde{a}^\star) \\ = &\quad Q_{\pi^\star}(x, a^\star) - Q_{\hat{\pi}^\star}(x, a^\star) \\ &\quad + Q_{\hat{\pi}^\star}(x, a^\star) - Q_{\bar{\pi}^\star}(h(x), a^\star) \\ &\quad + Q_{\bar{\pi}^\star}(h(x), a^\star) - Q_{\bar{\pi}^\star}(h(x), \tilde{a}^\star) \\ \leq &\quad Q_{\pi^\star}(x, a^\star) - Q_{\hat{\pi}^\star}(x, a^\star) + K(Q^\dagger_{\bar{\pi}^\star}). \end{aligned} \tag{3.27}$$

Since the rewards and state transitions that contribute to $Q_{\pi^\star}(x, a^\star)$ and $Q_{\hat{\pi}^\star}(x, a^\star)$ are the same, the only difference between them is how the value for future states are calculated:

$$\begin{aligned} &\quad Q_{\pi^\star}(x, a^\star) - Q_{\hat{\pi}^\star}(x, a^\star) \\ = &\quad \gamma \sum_{s' \in S} P(s', x, a^\star)(V_{\pi^\star}(s') - V_{\bar{\pi}^\star}(h(s'))). \end{aligned} \tag{3.28}$$

Since x is the state that gives the largest difference between $V_{\pi^\star}$ and $V_{\bar{\pi}^\star} \circ h$, and $\sum_{s' \in S} P(s', x, a^\star) = 1$:

$$\sum_{s' \in S} P(s', x, a^\star)(V_{\pi^\star}(s') - V_{\bar{\pi}^\star}(h(s'))) \leq V_{\pi^\star}(x) - V_{\bar{\pi}^\star}(h(x)). \tag{3.29}$$

Combining the results, we have:

$$\begin{aligned} V_{\pi^\star}(x) - V_{\bar{\pi}^\star}(h(x)) \leq &\quad \gamma\big(V_{\pi^\star}(x) - V_{\bar{\pi}^\star}(h(x))\big) + K(Q^\dagger_{\bar{\pi}^\star}) \\ \leq &\quad \frac{1}{1-\gamma} K(Q^\dagger_{\bar{\pi}^\star}). \end{aligned} \tag{3.30}$$

[6] Note that if we omit the disadvantage term in (3.27), we can essentially come to the same bound as Theorem 3.6.

□

The max norm of the performance loss can be bounded by combining Theorem 3.13 and Theorem 3.15:

Theorem 3.16 (Bounded value difference between an optimal policy and an lifted optimal policy)**.**

$$\|V_{\pi^\star} - V_{\tilde{\pi}^\star}\|_\infty \leq \frac{1}{1-\gamma}\big(K(Q^\dagger_{\tilde{\pi}^\star}) + K(V_{\tilde{\pi}^\star})\big) \tag{3.31}$$

For an optimal policy, $K(V_{\tilde{\pi}^\star}) \leq K(Q^\dagger_{\tilde{\pi}^\star})$ because the advantage of taking an optimal action over itself is zero. Therefore, the loss of performance of a lifted optimal policy can be bounded as

Corollary 3.17 (Bounded value difference between an optimal policy and an lifted optimal policy)**.**

$$\|V_{\pi^\star} - V_{\tilde{\pi}^\star}\|_\infty \leq \frac{2}{1-\gamma}K(Q^\dagger_{\tilde{\pi}^\star}) \tag{3.32}$$

We say an approximate role MDP is "optimally-well-aligned" if $K(Q^\dagger_{\tilde{\pi}^\star})$ is small, and "optimally-perfectly-aligned" if $K(Q^\dagger_{\tilde{\pi}^\star}) = 0$. It is possible for an optimally-perfectly-aligned approximate role MDP to have a relatively large $K(Q_{\tilde{\pi}^\star})$ because $K(V_{\tilde{\pi}^\star}) \leq K(Q^\dagger_{\tilde{\pi}^\star}) \leq K(Q_{\tilde{\pi}^\star})$. The only case of equality is when the maximum assignment temporal difference occurs at an optimal role-action pair. In other cases, bounding performance loss with $K(V_{\tilde{\pi}^\star})$ and $K(Q^\dagger_{\tilde{\pi}^\star})$ is always tighter than $K(Q_{\tilde{\pi}^\star})$.

The alignment with any policy requires the policy to be known. The alignment with optimality requires the role MDP to be solved. Here we make an assumption that the optimal policy, as well as the value function associated with the policy, are known or easily solvable. This is a reasonable assumption in practice because the number of roles we use is often very small. Since the advantage term for every role-action pair is different, it is not easy to give a separate bound for $K(Q^\dagger_{\tilde{\pi}^\star})$. Fortunately, with the optimal solution in the approximate role MDP known, finding the exact value of $K(Q^\dagger_{\tilde{\pi}^\star})$ is as easy as finding K_R and K_P.

Bounding the performance loss with $K(Q^\dagger_{\tilde{\pi}^\star})$ has an interesting interpretation: for any state, the higher value an action gives to its role, the more easily the reward and transition difference of that state-action pair may cause the loss of performance; if an action gives a relatively low value to its role, even if its transition and reward are not as accurate, the inaccuracy

may be compensated by a large advantage of taking the role MDP's optimal action over it. This property can be exploited for learning and improving an approximate role assignment, where the difference minimization is prioritized for actions that give higher values, and less for actions with lower values.

3.4 Soft Role Assignments

3.4.1 Assignment by weights

So far we have only discussed role assignment and approximate role assignment as partitions on the set of states. Since partitions are disjoint subsets, each state is assigned to exactly one role. We refer to such role assignment and approximate role assignment as "hard".

For many problems, we may encounter states' roles that can be seen as the "intermediate" between multiple roles. Consider the example shown in Figure 3.6, where states A, B and C yield the same reward while transitioning to 3 other states X, Y and Z. For simplicity, the roles of X, Y and Z are said to be all different and are coloured in yellow, pink and purple, respectively. Taking action `move-left`, state A transitions to state X and state C transitions to state Y with probability 1.0, while state C may transition to either X or Y with equal probability 0.5. Taking action `move-right`, A, B and C transition to Y and Z in a way that is symmetrical to `move-left`. If there is a blue role that state A can be assigned to, and there is a green role that state C can be assigned to, then without needing a separate role for state B, it can be perfectly described as half blue role and half green.

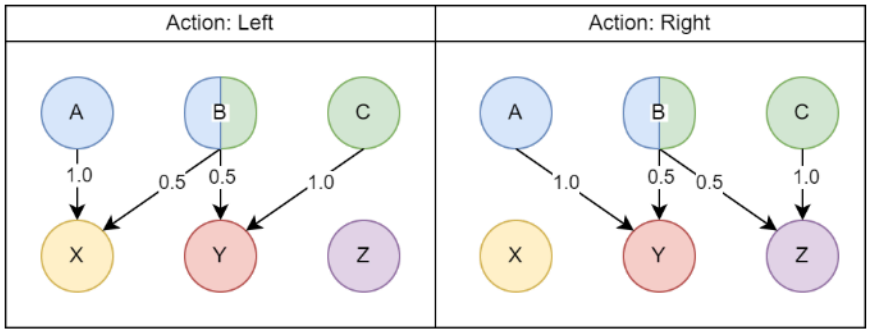


Figure 3.6: Soft role assignment. State B can be seen as 50% of blue role and 50% of green role.

Such "soft" assignment can be described with a generalization of partitions:

Definition 3.18 (Soft partition)**.** Let S be a set of states. A soft partition $W = \{C_1, C_2, ..., C_m\}$ on S is a set of sets whose size is the same as S. In every class, $C_j = \{w_{1,j}, w_{2,j}, ..., w_{n,j}\}$, with $w_{i,j}$ being a non-negative value called the weight of state s_i being partitioned into class C_j. Across the classes, $\sum_{j=1}^{m} w_{i,j} = 1$ for any i.

Similar to hard role partition, the soft partition W can be used as a state space $\bar{S}$ to define a soft role MDP: $\bar{M} = (\bar{S}, A, \bar{R}, \bar{P}, \gamma)$. The notation of weights is also overloaded to represent a soft role assignment function $w : S \times \bar{S} \mapsto [0, 1]$.

For soft role assignment and soft role MDP, we do not explicitly measure the equivalence or similarity between states. Instead, we make direct comparison between a state and its assigned roles' attributes, weighted by the assignment. A soft role assignment is *exact* if there exists a role MDP $\bar{M} = (\bar{S}, A, \bar{R}, \bar{P}, \gamma)$ such that

$$R(s, a) = \sum_{\bar{s} \in \bar{S}} w(s, \bar{s}) R(\bar{s}, a), \quad \forall s \in S, a \in A, \tag{3.33}$$

and

$$\sum_{s' \in S} P(s', s, a) w(s', \bar{s}') = \sum_{\bar{s} \in \bar{S}} w(s, \bar{s}) \bar{P}(\bar{s}', \bar{s}, a), \quad \forall s \in S, a \in A, \bar{s}' \in \bar{S}. \tag{3.34}$$

The soft partition W can be organized as an assignment matrix, $\mathbf{W}_{i,j} = w(i, j)$, then the matrix form of exact generalized role assignment is the same as equations 3.3 by replacing $\mathbf{H}$ with $\mathbf{W}$. The only difference between $\mathbf{W}$ and $\mathbf{H}$ is that values no longer have to be binary, but in range of $[0, 1]$. In this form, it is obvious that any exact hard role assignment is also an exact soft role assignment, but not the other way around.

An exact soft role assignment is considered to satisfy the role equivalence condition because if two states have the same role assignment, i.e., same weight distribution, then applying a same action sequence on them is guaranteed to have same expected returns, in terms of the expected rewards and the expected role distribution at each timestep. Additionally, even if a state finds no other equivalent states, its outcomes are still perfectly captured by its weighted roles' outcomes (see Appendix A for more details.).

3.4.2 Approximate planning

With soft role assignment, a policy defined on the role MDP cannot be directly lifted to the orignal MDP because it selects actions for individual roles, while the assignment distributes weights of the roles to each state. In other words, a policy or a value function need to be defined on weight distributions.

Learning policies and value functions for role distributions resembles solutions for POMDPs. For a POMDP, an optimal solution is a value function or a policy on belief states, i.e., the probability distributions of actual underlying states at different time steps. Exact solution for infinite horizon is generally undecidable in POMDPs [MHC99], and even for finite horizons, the computation is intractable due to the exponential growth of belief state space. In practice, POMDPs are typically solved with approximation [MHC99, PT87]. A line of work that uses the solution to the MDP as heuristics to solve the POMDP [Cas98, KLC95] can be adopted for lifting a policy in the role MDP under a soft role assignment.

The simplest approximation, referred to as Most Likely Role (MLR), is similar to the Most Likely State (MLS) heuristic for POMDPs. The lifted policy is simply taking the optimal action of the role with the highest weight:

$$\tilde{\pi}^{\star}_{\text{MLR}}(s) = \bar{\pi}^{\star}(\arg\max_{\bar{s}} w(s, \bar{s})). \tag{3.35}$$

MLR lifts a policy from a soft role MDP as if it were a hard role MDP.

A more flexible heuristic is Q-MDP. It approximates an action-value of a state using the weighted mean action-values of the roles:

$$Q_{\pi^{\star}}(s, a) \approx Q_{\bar{\pi}^{\star}_{\text{Q-MDP}}}(s, a) = \sum_{\bar{s} \in \bar{S}} w(s, \bar{s}) Q_{\bar{\pi}^{\star}}(\bar{s}, a). \tag{3.36}$$

Thus a lifted optimal policy picks an action that gives the maximum Q-MDP value:

$$\tilde{\pi}^{\star}_{\text{Q-MDP}}(s) = \arg\max_{\tilde{a}^{\star}} Q_{\bar{\pi}^{\star}_{\text{Q-MDP}}}(s, \tilde{a}^{\star}). \tag{3.37}$$

We use $V_{\bar{\pi}^{\star}_{\text{Q-MDP}}}$ as the value function of a state's role value with Q-MDP approximation:

$$V_{\bar{\pi}^{\star}_{\text{Q-MDP}}}(s) = \max_{\tilde{a}^{\star}} Q_{\bar{\pi}^{\star}_{\text{Q-MDP}}}(s, \tilde{a}^{\star}). \tag{3.38}$$

Obviously, for both MLR and Q-MDP, the performance of approximate planning depends on how spread the assignment weights are distributed.

We call the role a state is assigned to with the most weight the *dominating role* of the state, and the corresponding weight the *dominance*:

$$D(s) = \max_{\bar{s} \in \bar{S}} w(s, \bar{s}).$$

It is easy to see that if every state has a dominance of 1, lifting a Q-MDP policy is the same as lifting an optimal policy from a hard role assignment, and there is no performance loss from approximate planning.

If the soft role assignment is exact, then the role MDP is a soft homomorphism of the original MDP and the performance loss from approximate planning using Q-MDP can be bounded [SS09]. We now give a more general bound that is also applicable to non-exact soft assignments:

$$V_{\pi^\star} - V_{\tilde{\pi}^\star_{\text{Q-MDP}}} \leq \frac{2}{1-\gamma} K(Q_{\tilde{\pi}^\star_{\text{Q-MDP}}}), \tag{3.39}$$

where

$$K(Q_{\tilde{\pi}^\star_{\text{Q-MDP}}}) \leq K_R + \gamma\big((1-\mathcal{D})\delta_{\bar{A}^\star} + \delta_{V_{\bar{\pi}^\star}} \frac{K_P}{2}\big), \tag{3.40}$$

where $\mathcal{D} = \min_{s \in S} D(s)$ is the smallest dominance among all states and $\delta_{\bar{A}^\star}$ is the largest advantage among all roles for all actions. Here K_R and K_P are the maximum reward and transition differences between a state and the weighted average of its roles:

$$K_R = \max_{s \in S, a \in A} \Big| R(s,a) - \sum_{\bar{s} \in \bar{S}} w(s, \bar{s}) R(\bar{s}, a) \Big|, \tag{3.41}$$

and

$$K_P = \max_{s \in S, a \in A} \sum_{\bar{s}' \in \bar{S}} \Big| \sum_{s' \in S} P(s', s, a) w(s', \bar{s}') - \sum_{\bar{s} \in \bar{S}} w(s, \bar{s}) \bar{P}(\bar{s}', \bar{s}, a) \Big|. \tag{3.42}$$

See more details on this in Appendix B.

The advantage of using a soft role assignment over a hard role assignment is that it can often achieve a much smaller approximation error by giving up a little bit of dominance. A desired soft role assignment needs to balance the trade-off between small approximation error and large dominance. For soft MDP homomorphisms, [SS09] has developed a quadratic programming method and a gradient ascent method to minimize the quadratic forms of K_R and K_P. However, such methods are not applicable to a soft role assignment problem because unlike the source MDP in a soft homomorphism

problem, the role MDP is not given, but is determined along with the role assignment. More importantly, minimizing the approximation error alone isn't ideal because it will likely cost too much the overall dominance of the states.

It is important to note that role MDPs with soft role assignments are not the same as POMDPs because the actual dynamics of the problem are among the original states rather than the roles. Although both a weight distribution to roles and a belief state are a probability simplex[BBV04] of the roles/hidden states, they are fundamentally different. A weight distribution is determined by the assignment function—as long as the assignment function is not changed, a state's assignment weights are not changed. However, a belief state is inferred from not only the observation emission probability, but also from its rewards and transitions between other belief states. Even with the same observation, the belief states may vary. We adopt MLS and Q-MDP medthods only because they enable the use of solutions for the role MDPs for soft role assignments the same way they use the solutions for the hidden MDPs. There are also other methods for POMDPs that do not involve the solutions [RPPCD08], and they are not useful for lifting role MDP policies.

Chapter 4

Assignment Iteration and Update

In this chapter, we develop an approximation method using a soft role assignment. This method allows us to utilize the similarity between a state and a role, as opposed to relying on the computationally expensive state-state similarities.

4.1 State-Role Similarity Score

As the end product of state space abstraction, the role MDP has been given different names in different literatures — the quotient model [FPP11], aggregated MDP [TPP08], the image MDP [RB04], etc. However, its usefulness in finding a role assignment has often been overlooked. This is understandable because once the assignment is determined, the roles' attributes can easily be decided. For example, in [RB04], a role's attributes are the unweighted average of all states that are assigned to it.

The typical approach that can be used to find the partition before determining the role MDP is to define a metric that measures the distance between states, thus the partition can be based on how close states are to each other [FPP04, FPP05, FPP11, Fer07, KAA21]. While methods of this class have a strong performance guarantee, the computation is very expensive.

Instead of directly measuring the distance between states, we now propose a method based on the distance between a state and a role. This is made possible with the assumption that a role MDP of size m is always present, and we solve the assignment problem while changing the parameters of the role MDP. We define the distance between a state and a role as follows:

$$d(s, \bar{s}) = \beta \sum_{a \in A} \left| R(s, a) - \bar{R}(\bar{s}, a) \right| + \frac{1}{2} \sum_{a \in A, \bar{s}' \in \bar{S}} \left| \sum_{s' \in S} P(s', s, a) w(s', \bar{s}') - \bar{P}(\bar{s}', \bar{s}, a) \right|. \quad (4.1)$$

The first term is the sum of reward differences. The second term is the total variation distance between the state's transition-to-roles and the role's transition. Function w is the weight that a state is assigned to a role. Since the range of reward differences may vary from problem to problem, while the total variation distance is always within the range $[0, 1]$, the rewards are re-scaled and balanced by $\beta > 0$.[7]

Given a state and its distances to all roles, it would be tempting to simply assign it to the role with the smallest distance. However, doing so has a problem. Notice that in Equation (4.1), the assignment weights are a part of a state's attributes, thus also a part of the distance. Changing the assignment of one state may also change the most similar role of all states, including the state itself.

We tackle this problem by using a soft role assignment, where a state's assignment weight is distributed according to its distances to the roles—the shorter the distance a role has, the more weight. This gives a relatively small soft role assignment approximation error (compared to the exact soft role assignment conditions in Equation (3.33) and Equation (3.34).).

As we will see later in this chapter, our method also indirectly encapsulates the distance information between states, and has several computational and applied advantages. We summarize the advantages as follows:

1. There is no need to store the large number of state-state distances. The distances between states and roles do not have to be stored either, because by storing the attributes of $|S|$ roles, state-role can be computed very quickly.

2. There is no need to compute the expensive Kantorovich distances between two states' transitions because the distances are bounded through their role assignments. We will discuss this bound in detail throughout the rest of the section.

[7] The fraction $\frac{1}{2}$ can be merged with β; we choose to separate it so the term has a form of total variation difference.

3. A state's assignment can be updated without explicitly comparing it with all other states. This makes asynchronous updates possible. In the next chapter, we will further extend the update method to work with incomplete samples (the agent's interaction with the environment), so that model minimization can be done online.

The similarity score of a state-role pair is defined as a scaled, negative distance:

$$\psi(s, \bar{s}) = -cd(s, \bar{s}). \tag{4.2}$$

c is a positive scalar which we refer to as the *concentration parameter*. It controls the overall magnitude of the score, and there is more to that. The importance of the concentration parameter will be explained throughout the rest of the chapter.

The weighted role assignment of a state is approximated with the softmax of the state's score across all roles:

$$\hat{w}(s, \bar{s}) = \frac{\exp(\psi(s, \bar{s}))}{\sum_{\bar{s}' \in \bar{S}} \exp(\psi(s, \bar{s}'))}. \tag{4.3}$$

There are two things being approximated. First, the hard assignment is approximated with a soft assignment. It can generally be interpreted as a state being assigned to roles that are more similar to it with higher weights. The concentration parameter c works like the inverse of the softmax temperature and determines the dominance of the assignment. When $c \to 0$, the assignment approaches a uniform distribution; when $c \to \infty$, the weights approach a hard assignment and the state is assigned to only the most similar role.

The second approximation is the decoupling of the assignment for the current state and the assignment for the next states. Ideally, we want to find out the assignment w such that across all states the maximum difference or the total difference is minimized. However, finding the exact solution is hard because d is also defined with w. With the two assignments treated separately, the minimization can be performed locally and temporally: for each state, we simply assign it to the roles with weights based on the distances and the concentration. Obviously, the quality of the approximation depends on the assignment used for the next states. In the next section we will show that through an iterative process, the assignment will encapsulate important information of the original MDP to make a good approximation.

Applying softmax on the similarity scores has a probabilistic interpretation. Given a role, the exponentiated score it has to a possible state is a re-scaled product of Laplace distributions. This is a desirable property since

the role MDP is typically used as an abstract model of the original MDP. Each role is expected to *predict* the outcomes of the states that are assigned to it. The prediction is not made deterministically, but rather forms a probability distribution that has a higher probability for states that are more similar to the role, and lower probability for the less similar ones.

It is in this sense that we refer to the exponentiated score as likelihood: $\mathcal{L}(\bar{s}|s) = \exp(\psi(s, \bar{s}))$. The softmax assignment of a state is equivalent to the conditional probability $\mathbb{P}(\bar{s}|s)$ with a uniform prior for all roles.

4.2 Assignment Iteration and True Similarity

4.2.1 Assignment iteration

We now define the iterative process referred to as *assignment iteration*. Let $\mathbf{W}^{(t)}$ and $\mathbf{L}^{(t)}$ be the assignment and likelihood matrix, respectively, at iteration t. $\mathbf{W}^{(0)}$ is initialized uniformly, i.e., $\mathbf{W}^{(0)}_{i,j} = \frac{1}{m}$ for all (i, j). From $t = 1$ and onward, we can compute the likelihood matrix as follows:

$$\begin{aligned}\mathbf{L}^{(t)}_{i,j} &= \exp(-cd^{(t-1)}(s_i, \bar{s}_j)) \\ &= \exp\big(-c\beta \sum_{a \in A} |R(s_i, a) - \bar{R}(\bar{s}_j, a)|\big) \\ &\quad \times \prod_{a \in A} \exp\big(-c\delta_{\mathrm{TV}}(\mathbf{P}^{(a)}_{i,\cdot}\mathbf{W}^{(t-1)}, \bar{\mathbf{P}}^{(a)}_{j,\cdot})\big).\end{aligned} \tag{4.4}$$

With a uniform prior for all roles, the new assignment matrix can be acquired by normalizing the likelihood matrix by rows:

$$\mathbf{W}^{(t)}_{i,j} \leftarrow \frac{\mathbf{L}^{(t)}_{i,j}}{\sum_{k=1}^{m} \mathbf{L}^{(t)}_{i,k}} = \frac{1}{1 + \frac{\sum_{k \in \{1 \dots m\} \setminus i} \mathbf{L}^{(t)}_{i,k}}{\mathbf{L}^{(t)}_{i,j}}}. \tag{4.5}$$

4.2.2 True similarity

Let's consider how the similarities between states change over the assignment iteration process. Conventionally, the similarity is measured by the differences in the states' attributes. Since transition-to-roles are part of a state's attributes and they change at each iteration, the similarity between a pair of states also changes. However, there is a certain restriction on how it can change. Intuitively, if two states directly transition to some common

state, or the difference between their rewards is small under the same action, they must always be similar to a degree, no matter how many times the assignment has been changed; if the similarity between the states is guaranteed, for other states that transition to these two states, the similarity may also be guaranteed. We regard this guaranteed similarity as *true similarity*.

As an informal definition, two states are truly similar if they have truly similar outcomes, i.e., the rewards and the transitions are truly similar. Since rewards are stationary, similarities from rewards are always true. The true similarity of transition depends on how truly similar the next states they transition to are. A state is always truly equivalent (or the most truly similar) to itself. It is easy to see that true similarity is directly related to bisimulation and bisimulation metrics discussed in subsections 3.1.2 and 3.1.6.

Formally, we define the true similarity as a lower bound:

Definition 4.1 (True similarity)**.** The true similarity between two states is the largest lower bound on the ratio from one state's likelihood of a role, to the other state's likelihood of the same role, based on their reward similarity and the true similarity between the states they transition to. All states have a true similarity of 1 with themselves.

Of course, the true similarity is unknown at the beginning of the process, except for the self-similarities, but starting from a uniform assignment matrix, the subsequent assignment matrices acquired throughout the assignment iteration process are always bounded by true similarity. We show this by constructing an n by n *candidate true similarity matrix* $\Omega[\mathbf{L}]^{(t)}$ at each iteration t, in which

$$\text{for all roles } \bar{s}_j \ , \quad \frac{\mathbf{L}^{(t)}_{x,j}}{\mathbf{L}^{(t)}_{y,j}} \geq \Omega[\mathbf{L}]^{(t)}_{x,y}. \tag{4.6}$$

True similarity may also be defined as the lower bounds on the ratios of assignment weights. They can easily be established based on the lower

bounds on the ratios of likelihoods:

$$\begin{aligned}\frac{\mathbf{W}^{(t)}_{x,j}}{\mathbf{W}^{(t)}_{y,j}} &= \left(\frac{\mathbf{L}^{(t)}_{x,j}}{\sum_{k=1}^{m}\mathbf{L}^{(t)}_{x,k}}\right) \div \left(\frac{\mathbf{L}^{(t)}_{y,j}}{\sum_{k=1}^{m}\mathbf{L}^{(t)}_{y,k}}\right) \\ &= \frac{\mathbf{L}^{(t)}_{x,j}}{\mathbf{L}^{(t)}_{y,j}} \times \frac{\sum_{k=1}^{m}\mathbf{L}^{(t)}_{y,k}}{\sum_{k=1}^{m}\mathbf{L}^{(t)}_{x,k}} \\ &\geq \frac{\Omega[\mathbf{L}]^{(t)}_{x,y}\mathbf{L}^{(t)}_{y,j}}{\mathbf{L}^{(t)}_{y,j}} \times \frac{\Omega[\mathbf{L}]^{(t)}_{y,x}\sum_{k=1}^{m}\mathbf{L}^{(t)}_{x,k}}{\sum_{k=1}^{m}\mathbf{L}^{(t)}_{x,k}} \\ &= \Omega[\mathbf{L}]^{(t)}_{x,y}\Omega[\mathbf{L}]^{(t)}_{y,x}.\end{aligned} \tag{4.7}$$

Like $\Omega[\mathbf{L}]$, we also organize the lower bounds on the ratios of assignment weights in a matrix $\Omega[\mathbf{W}]$, where

$$\text{for all roles } \bar{s}_j \ , \quad \frac{\mathbf{W}^{(t)}_{x,j}}{\mathbf{W}^{(t)}_{y,j}} \geq \Omega[\mathbf{W}]^{(t)}_{x,y}. \tag{4.8}$$

With the result from (4.7), $\Omega[\mathbf{W}]^{(t)}$ can be computed easily from $\Omega[\mathbf{L}]^{(t)}$:

$$\Omega[\mathbf{W}]^{(t)} = \Omega[\mathbf{L}]^{(t)} \circ \Omega[\mathbf{L}]^{(t)\intercal}, \tag{4.9}$$

where $\circ$ means element-wise or the Hadamard product. Although $\Omega[\mathbf{L}]$ may or may not be symmetric depending on how the bound is derived,[8] $\Omega[\mathbf{W}]$ are always symmetric at any iteration.

The two lower bound matrices have intuitive interpretations. At iteration t,

- if a role predicts state s_y with probability p_y, it must also predicts state s_x with at least a probability of $p_y\Omega[\mathbf{L}]^{(t)}_{x,y}$; and
- if state s_y is assigned to a role with weight w_y, the weight state s_x is assigned to the same role is at least $w_y\Omega[\mathbf{W}]^{(t)}_{x,y}$.

The idea behind candidate true similarity matrices $\Omega[\mathbf{L}]^{(t)}$ is that, if the likelihood ratios are bounded at one iteration, the bounds can be used for deriving new bounds for the likelihood ratios at the next iteration. $\Omega[\mathbf{L}]^{(t)}$ monotonically decreases with respect to t, and will converge at a certain point, which by definition, is the true similarity we need.

[8] In this thesis we primarily consider the symmetric case.

Since $\mathbf{W}^{(0)}$ is uniform, we set:

$$\Omega[\mathbf{W}]^{(0)} = \Omega[\mathbf{L}]^{(0)} = \mathbf{1}. \tag{4.10}$$

From $t = 1$ and onward, the likelihood ratio for each state pair over each role can be calculated as:

$$\begin{aligned} &\mathbf{L}^{(t)}_{x,j} \div \mathbf{L}^{(t)}_{y,j} \\ =& \exp\left(c\beta\big(\sum_{a\in A}|R(s_y,a) - \bar{R}(\bar{s}_j,a)| - \sum_{a\in A}|R(s_x,a) - \bar{R}(\bar{s}_j,a)|\big)\right) \\ &\prod_{a\in A}\exp\left(c\big(\delta_{\mathrm{TV}}(\mathbf{P}^{(a)}_{y,\cdot}\mathbf{W}^{(t-1)}, \bar{\mathbf{P}}^{(a)}_{j,\cdot}) - \delta_{\mathrm{TV}}(\mathbf{P}^{(a)}_{x,\cdot}\mathbf{W}^{(t-1)}, \bar{\mathbf{P}}^{(a)}_{j,\cdot})\big)\right). \end{aligned} \tag{4.11}$$

To give a lower bound on the ratio, we only need to give each reward distance and transition distance an upper bound. Since the two states' reward distances under all actions are stationary, their reward likelihood ratio on any role is a constant. Without specifying a role, we may also apply the triangle inequality, and bound it directly by the difference between the states' rewards.

The more interesting part is the likelihood ratio of two states' transition-to-roles. Total variation distance is a special form of 1-norm, so it also follows triangle inequality. Under one action, the ratio is bounded as:

$$\begin{aligned} &\exp\left(c\big(\delta_{\mathrm{TV}}(\mathbf{P}^{(a)}_{y,\cdot}\mathbf{W}^{(t-1)}, \bar{\mathbf{P}}^{(a)}_{j,\cdot}) - \delta_{\mathrm{TV}}(\mathbf{P}^{(a)}_{x,\cdot}\mathbf{W}^{(t-1)}, \bar{\mathbf{P}}^{(a)}_{j,\cdot})\big)\right) \\ \geq& \exp\left(-c\delta_{\mathrm{TV}}(\mathbf{P}^{(a)}_{x,\cdot}\mathbf{W}^{(t-1)}, \mathbf{P}^{(a)}_{y,\cdot}\mathbf{W}^{(t-1)})\right). \end{aligned} \tag{4.12}$$

By Bretagnolle-Huber inequality [BH79], a total variation distance between two probability distributions is bounded with their KL-divergence:[9]

$$\delta_{\mathrm{TV}}(A,B) \leq \sqrt{1 - \exp\big(-\delta_{\mathrm{KL}}(B|A)\big)} \tag{4.13}$$

Thus,

$$\begin{aligned} &-c\delta_{\mathrm{TV}}\big(\mathbf{P}^{(a)}_{x,\cdot}\mathbf{W}^{(t-1)}, \mathbf{P}^{(a)}_{y,\cdot}\mathbf{W}^{(t-1)}\big) \\ \geq& -c\sqrt{1 - \exp\left(\sum_{j'=1}^{m}\mathbf{P}^{(a)}_{y,\cdot}\mathbf{W}^{(t-1)}_{\cdot,j'}\log\left(\frac{\mathbf{P}^{(a)}_{x,\cdot}\mathbf{W}^{(t-1)}_{\cdot,j'}}{\mathbf{P}^{(a)}_{y,\cdot}\mathbf{W}^{(t-1)}_{\cdot,j'}}\right)\right)}. \end{aligned} \tag{4.14}$$

[9] KL-divergence isn't necessarily symmetric, i.e., $\delta_{\mathrm{KL}}(B\|A) \neq \delta_{\mathrm{KL}}(A\|B)$, but both directions can be used to bound the total variation distance.

Notice that $\mathbf{W}^{(t-1)}_{\cdot,j'}$ is a column of the assignment matrix at iteration $t-1$, so the ratios between its elements are constrained by $\Omega[\mathbf{W}]^{(t-1)}$. We do not need to know the actual values in the column; knowing only $\Omega[\mathbf{W}]^{(t-1)}$ is sufficient to give both a lower bound on the ratio of the two linear combinations. Let $\eta^{(t-1,a)}_{(s_x,s_y)}$ denote the largest lower bound on the ratio of a given action that can be derived. Then,

$$\forall j' \in \{1, ..., m\}, \quad \frac{\mathbf{P}^{(a)\intercal}_{x,\cdot}\mathbf{W}^{(t-1)}_{\cdot,j'}}{\mathbf{P}^{(a)\intercal}_{y,\cdot}\mathbf{W}^{(t-1)}_{\cdot,j'}} \geq \eta^{(t-1,a)}_{(s_x,s_y)}, \tag{4.15}$$

and combining all the roles j', we have

$$-c\delta_{\mathrm{TV}}(\mathbf{P}^{(a)}_{x,\cdot}\mathbf{W}^{(t-1)}, \mathbf{P}^{(a)}_{y,\cdot}\mathbf{W}^{(t-1)}) \geq -c\sqrt{1-\eta^{(t-1,a)}_{(s_x,s_y)}}. \tag{4.16}$$

The values of $\eta^{(t-1,a)}_{(s_x,s_y)}$ can be found with linear programming. It is unnecessary to solve the LP; what we need are the two important properties of $\eta^{(t-1,a)}_{(s_x,s_y)}$ on an arbitrary state pair (s_x, s_y):

- It is independent of the exact values in $\mathbf{W}^{(t-1)}$, as long as they are constrained by $\Omega[\mathbf{W}]^{(t-1)}$.
- It can be expressed as a linear combination of elements from $\Omega[\mathbf{W}]^{(t-1)}$ with non-negative coefficients.

For the details of the linear program and the coefficients, see Appendix C.

To derive a new candidate true similarity between s_x and s_y, we simply combine the reward distance and (4.16):

$$\frac{\mathbf{L}^{(t)}_{x,j}}{\mathbf{L}^{(t)}_{y,j}} \geq \exp\left(-c\sum_{a\in A}\left(\beta|R(s_x,a)-R(s_y,a)| + \sqrt{1-\eta^{(t-1,a)}_{(s_x,s_y)}}\right)\right) \to \Omega[\mathbf{L}]^{(t)}_{x,y}. \tag{4.17}$$

Deriving new lower bounds from the old results in their monotonic decrease. The decrease may be seen as the invalidation of true similarity between states that cannot be similar.

With the properties of the linear program, the following can be established:

$$\Omega[\mathbf{W}]^{(t)} \leq \Omega[\mathbf{W}]^{(t-1)} \implies \forall(s_x,s_y), \eta^{(t,a)}_{(s_x,s_y)} \leq \eta^{(t-1,a)}_{(s_x,s_y)}. \tag{4.18}$$

With (4.18) and the fact that the distance between rewards is a constant for each state pair, we have

$$\Omega[\mathbf{W}]^{(t)} \leq \Omega[\mathbf{W}]^{(t-1)} \implies \Omega[\mathbf{L}]^{(t+1)} \leq \Omega[\mathbf{L}]^{(t)}, \tag{4.19}$$

and with (4.9), we have

$$\Omega[\mathbf{L}]^{(t)} \leq \Omega[\mathbf{L}]^{(t-1)} \implies \Omega[\mathbf{W}]^{(t)} \leq \Omega[\mathbf{W}]^{(t-1)}. \tag{4.20}$$

Together with the obvious base case at the first iteration, due to the reward differences, $\Omega[\mathbf{W}]^{(1)} \leq \Omega[\mathbf{W}]^{(0)}$, the true similarity is monotonically decreasing with respect to t. That is, for all $t \geq 1$,

$$\Omega[\mathbf{W}]^{(t)} \leq \Omega[\mathbf{W}]^{(t-1)}, \tag{4.21}$$

and

$$\Omega[\mathbf{L}]^{(t)} \leq \Omega[\mathbf{L}]^{(t-1)}. \tag{4.22}$$

Since the maximum distance between rewards and the total variation distance are both bounded, the decrease of candidate true similarity will certainly converge, at which point $\Omega[\mathbf{L}]^{(\infty)}$ can be used to derive itself. Namely, $\Omega[\mathbf{L}]^{(\infty)}$ is the true similarity. True similarity fully preserves bisimulation:

Theorem 4.2. *If there exists an exact role assignment, in which two states are bisimilar, through assignment iteration, the candidate true similarity between the two states always remains 1.*[10]

Proof. Without loss of generality, we denote two equivalent states as s_x and s_y. We then take all the states they can transition to in any number of steps, and put them in two subsets S_x and S_y accordingly. Expanding Definition 3.3, the exact role assignment finds each state in S_x an equivalent state S_y and vice versa.

By induction on the iteration counter t. At any t, taking any equivalent state pair, $s_{x'} \equiv s_{y'}$, we match the states they transition to by their equivalence. If every matched pair has a candidate true similarity equal to 1 at iteration $(t-1)$, for any action a, $\eta^{(t,a)}_{(s_{x'}, s_{y'})}$ must be 1. The equivalence condition also implies identical rewards. Therefore $\Omega[\mathbf{L}]^{(t)}_{x',y'} = 1$ for all $x' \in S_x, y' \in S_y$. Since all states x' and y' transition to are all included in S_x and S_y, respectively, after normalization, $\Omega[\mathbf{W}]^{(t)}_{x',y'} = 1$ for all $x' \in S_x, y' \in S_y$. For the base case, notice the universal initial $\Omega[\mathbf{W}]^{(0)} = 1$. □

[10] Note that the opposite direction of the theorem is not true. Two states may have a true similarity of 1, while not being equivalent in any hard assignment. However, these cases are rare and only exist with certain values of concentration c.

True similarity is useful because it works as a lower bound. We don't need to solve for true similarities, but the assignment matrix acquired by assignment iteration is always bounded by them. If two states are bisimular, they must have an identical assignment on all roles; if two states are nearly bisimular, the difference between their role assignment is bounded according to how similar they are. This can be seen as a guarantee of the approximation quality.

Note that the true similarity discussed in this section may be considered "conservative" because for each pair of states, there is only one universal lower bound across all roles. We may use a more "aggressive" true similarity by defining $\Omega[\mathbf{L}]$ and $\Omega[\mathbf{W}]$ as $n \times m \times n$ tensors, such that the bounds are treated differently on every role. Aggressive true similarity is a tighter bound, but can only be applied when roles' attributes are not changed. Being able to bound the ratio regardless of the role MDP is an advantage when we interleave assignment iteration with role MDP update.

4.2.3 Role MDP update

Since assignment iteration is based on the state-role similarity, it is important that the role MDP is updated frequently. Typically, between assignment iterations, at a certain frequency, the roles' attributes are set to the average of the states, weighted by the states' assignment weights:

$$\theta_{\bar{s}} \leftarrow \frac{\sum_{s \in S} w(s, \bar{s}) \theta_s}{\sum_{s \in S} w(s, \bar{s})}. \tag{4.23}$$

Updating the role MDP ensures that it reflects the most recent role assignment, and there is more. Throughout the assignment iteration process, any two states' assignment weight ratio is lower bounded by their true similarity. However, true similarity itself doesn't give any meaningful upper bound, and trivial solutions cannot be prevented. For example, if all states have the same role assignment, any true similarity is always satisfied.

Intuitively, we want states to have assignments that are as different as possible, except for those that are nearly bisimular (guaranteed by their true similarity). Interleaving the role MDP update along with the assignment iteration helps diversify the assignment. If a role is highly similar to some states, it then gets assigned by these states with larges weights, and will likely remain similar to them after a role MDP update. Conversely, if a role is generally different from all states, it gets small assignment weights from them, thus the update is sensitive to small weight differences. In this way, every role is encouraged to become the dominating role for at least some

states, so that the assignments are different from states to states, while bounded by true similarity.

Alternating between assignment iteration and role MDP update may appear similar to solving a mixture model using an expectation-maximization algorithm. However, there are a few fundamental differences:

- We do not estimate the prior probability of the roles because the assignment is purely based on the similarities states have with roles.
- Role MDP update is not an m-step in EM because the weighted average is not necessarily the maximum likelihood estimator.
- Most importantly, the transition-to-roles as states' attributes are not stationary.

4.3 Assignment Update

4.3.1 Concentration

The approximation quality of assignment iteration depends on choosing the right concentration parameter c. If c is too large, even small differences between states' attributes can cause a large difference in their likelihood of the same roles. This is not an issue for bisimular states, but for other states, true similarities may be meaninglessly low, and barely propagate. If c is too small, assignment iteration cannot effectively invalidate the candidate true similarity between less similar states. As a result, the assignment matrix always stays near-uniform, no matter how many iterations there have been. A near-uniform assignment matrix doesn't approximate a hard assignment well and causes poor performance when approximate planning is used.

The update of role MDP is also sensitive to c. Ideally, every state has a widespread of true similarities, from high to low, with other states, thus truly similar states together "win over" a role, while letting other roles be "taken away" by less truly similar states. If c is too large, a state is likely to have low true similarity with all other states that are not equivalent to it, so their role MDP update cannot be controlled; if c is too small, every state has high true similarities with all other states, the update gradually causes all role to become the same.

Choosing the right c is difficult because the spread of true similarities depends on the structure of the original MDP. If states are generally similar to each other, a large c is desirable to distinguish the more similar from the less; otherwise, a smaller c is desirable to preserve true similarity. Without

knowing the details of the original MDP, it might require multiple trials of assignment iteration to find a good value for c.

4.3.2 Assignment update

To handle this difficulty, we consider an extension to assignment iteration known as *assignment update*. Instead of normalizing the likelihood to acquire the assignment, we update the assignment on top of the old assignment.

Notice a property of softmax function σ, for some $\alpha > 0$:

$$\sigma(\alpha \mathbf{z})_i = \frac{\exp(\alpha \mathbf{z}_i)}{\sum_j^m \exp(\alpha \mathbf{z}_j)} \propto \sigma(\mathbf{z})_i^\alpha. \tag{4.24}$$

That is, we can use a uniform distribution, and repetitively update it by multiplying it with the softmax of a vector and normalizing it, and the result is the same as directly calculating the softmax of the vector multiplied by the scalar.

The update of role assignment follows a similar fashion. The calculation for assignment weights of greater concentration is split into multiple steps of updating the old assignment by multiplying the likelihood acquired with a smaller concentration.

Note that unlike the update for the naive softmax, we allow the likelihood to be different at every update, calculated using the latest assignment. This update rule can be written as:

$$\mathbf{W}_{i,j}^{(t)} \leftarrow \frac{\mathbf{W}_{i,j}^{(t-1)} \mathbf{L}_{i,j}^{(t)}}{\sum_{k=1}^m \mathbf{W}_{i,k}^{(t-1)} \mathbf{L}_{i,k}^{(t)}}. \tag{4.25}$$

At every update, if an assignment weight's corresponding likelihood is greater than the weighted average of all likelihoods from the same state, the weight gets increased; if the likelihood is less than the weighted average, the weight gets decreased.

We can establish the connection between the assignment iteration method and the assignment update method in a probabilistic sense. The assignment iteration is the Bayesian inference of the conditional probability of roles given a state, using a uniform prior. The assignment update method may also be considered as finding the conditional probability, except that the prior is the state's previous role assignment. Such method that repetitively updates the prior using the posterior as more evidence comes in is known as "Bayesian belief updating".

The reason why assignment update is preferable over assignment iteration is that true similarity is preserved through both the likelihood and the old assignment, as opposed to only through likelihood. The concentration parameter used in assignment update is also named the "*step size*". It is a relatively small value, so the likelihood preserves much of the true similarity from the previous assignment weights. Meanwhile, as the update goes on, the overall concentration is increasing, leading to an assignment of high dominance.

Switching from assignment iteration to assignment update alone doesn't solve the concentration problem, because eventually true similarity diminishes from the assignment weights. However, it is possible to "flatten" the assignment weights by a small value ξ between updates. The update rule we use is defined as follows:

$$\mathbf{W}_{i,j}^{(t)} \leftarrow \left(\frac{\mathbf{W}_{i,j}^{(t-1)} \mathbf{L}_{i,j}^{(t)}}{\sum_{k=1}^{m} \mathbf{W}_{i,k}^{(t-1)} \mathbf{L}_{i,k}^{(t)}} + \xi \right) \times \frac{1}{1 + m\xi}. \tag{4.26}$$

Flattening changes the update little if the assignment is near-uniform, but has a more significant effect as the weight distribution becomes more skewed. It guarantees that every role is assigned to with at least a minimum weight and prevents a state to be assigned to a role completely.

How true similarity changes with the addition of flattening is hard to pin down mathematically, but our tests show that it finds significantly better assignments than the assignment iteration method in terms of the bisimulation metrics calculated between states whose most assigned role is the same. Additionally, although it is still required to specify the concentration/step size parameter c and the flattening factor ξ, the assignment update process appears more robust to the two values.

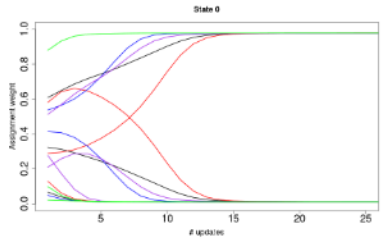
We conjecture that flattening helps spread true similarity in two main ways:

- For state pairs with similar assignment weights, flattening changes very little to their assignment weights' ratios; for state pairs with very different assignment weights, flattening reduces the differences. It directly adds to the values in $\Omega[\mathbf{W}]$. The smaller $\Omega[\mathbf{W}]_{i,j}$, the more likely it gets a greater increase, but the general order is preserved.

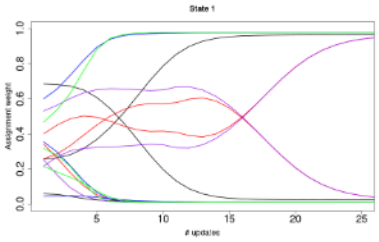
- Flattening adds a uniform weight distribution (0 concentration) to the weight distribution calculated using the old $\mathbf{W}$ (cumulative concentration). Since flattening affects each state differently, the effective concentration is also different for each state.

The usefulness of flattening may also be explained from the Bayesian belief updating's view. At the beginning of the process, the evidence (the states' attributes) and the likelihood (the exponentiated similarity score) are not accurate. Thus the prior may converge to a bad assignment. Flattening always prevents the prior from converging to a deterministic (hard) assignment, allowing the process to "correct" mistakes.

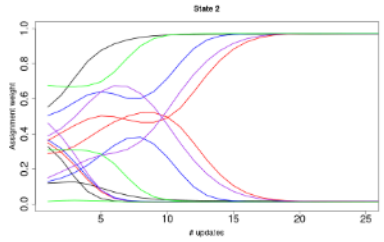
(a) An MDP with 2 actions and 5 states



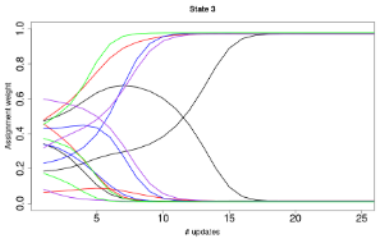
(b) State 0



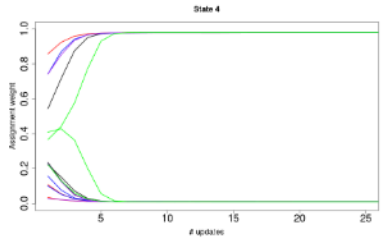
(c) State 1



(d) State 2



(e) State 3



(f) State 4

Figure 4.1: An example of a very simple 5-state MDP. Only two actions are possible: `move-left` and `move-right`. At state 0, `move-left` gives reward of 0.5 and terminates; at state 4, `move-right` gives reward of 1 and terminates; at any other states, it transitions to an adjacent state of the chosen direction and gives 0 reward. The plots shows how each state finds its weights of 3 roles with the assignment update method. The tests ran 5 times, with different randomly initialized $\bar{R}$ and $\bar{P}$. Each run is represented in a color. Hyper-parameters used are: $c = 0.01$, $\beta = 1$, $\xi = 0.01$. In this problem, all 5 tests found same solution where the partition of the states' most likely role is {0}{1,2,3}{4}.

Chapter 5

Reinforcement Learning with Role Assignments

The MDP minimization/state space abstraction methods discussed in previous chapters are based on the assumptions that 1) the original MDP is known, and 2) the state space is discrete, finite, and practically enumerable. However, both assumptions are often not true for real-world interesting problems.

In this chapter, we address these issues. In section 5.1, we extend the assignment update method to reinforcement learning settings, so that without knowing the full attributes of states, it works with samples collected by the agent's interaction with the environment. In section 5.2, we parameterize the assignment weights using deep neural networks.

5.1 Role Assignments on Unknown MDPs

5.1.1 Assignment update with experience

Recall that the softmax of state-role similarity score has a probabilistic interpretation. By treating the exponentiated score as likelihood, the main procedure of both the assignment iteration method and the assignment update method is the Bayesian inference of the roles, with the difference of which prior to use. Calculating the likelihood of a state belonging to a role requires the similarity score, which is the scaled sum of negative differences in rewards and transitions under all actions. Without knowing the outcomes of the state, this state-role likelihood cannot be calculated.

However, note that the state-role likelihood can be calculated by separately exponentiating the negative distances of each action, and then taking the product. The action-specific distances can be sampled through the agent's interaction with the environment. Therefore, we use the agent's experience of taking an action as evidence to perform the Bayesian update. A piece of experience is a tuple: $x_t = (s_t, a_t, r_t, s_{t+1})$. Similar to the state-role

likelihood, we define the likelihood of the experience belonging to role $\bar{s}$ as

$$\mathcal{L}(\bar{s}|x_t) = \exp\big(-c\beta|r_t - \bar{R}(\bar{s},a)|\big)\exp\Big(-c\delta_{\mathrm{TV}}\big(w(s_{t+1},\cdot),\bar{P}(\cdot,\bar{s},a)\big)\Big). \tag{5.1}$$

Role MDP update also works well with experience. Given a piece of experience, we can estimate both 1) which role the experience belongs to, and 2) the state's attributes under the chosen action. With both estimators, we adjust all roles slightly. Let $w^{(t)}(s,\bar{s})$ denote the assignment weight for the experienced state at the t-th role update to role $\bar{s}$, then every attribute of $\bar{s}$ is updated as:

$$\theta_{\bar{s}}^{(t+1)} \leftarrow \theta_{\bar{s}}^{(t)} + \alpha_{\bar{s}}^{(t)} w^{(t)}(s,\bar{s})(\theta_s^{(t)} - \theta_{\bar{s}}^{(t)}). \tag{5.2}$$

The magnitude of the update is controlled by the product of a learning rate $\alpha_{\bar{s}}^{(t)}$ and the estimated assignment weight. The learning rate $\alpha_{\bar{s}}^{(t)}$ is the inverse of the total weight that has contributed to the role attributes during the most recent K role updates:

$$\alpha_{\bar{s}}^{(t)} = \Big(\sum_{k=0}^{K} w^{(t-k)}(s,\bar{s})\Big)^{-1}. \tag{5.3}$$

Therefore it is different for every role and is different at every update. The adjustment may also be seen as incremental averaging, but with a fixed count.

A naive way is to update the assignment and the role MDP every time the agent interacts with the environment, using the latest piece of experience. However, there is a correlation within the state sequence, which may cause the role assignment to over-commit to the most recent episode. Therefore, we employ experience replay [Lin92] for both the assignment update and the role MDP update.

Algorithm 3 shows the framework of role assignment using experience.

5.1.2 Value-guided exploration

The offline MDP role assignment treats all state-action pairs equally. When experience is used as samples, bias is introduced according to the policy the agent follows. Using biased samples, the assignment update no longer assigns states to roles according to the state-role distances.

However, the bias can be used to our advantage in finding a more exploitable assignment. In Section 3.3, we have shown that the role assignment

Algorithm 3: State space abstraction with experience

```
initialization
    W ← uniform;
    P̄ ← random;
    R̄ ← random;
    buffer is a queue of a limited size;
end
begin
    s ← current state;
    while Termination condition not met do
        /* agent-environment interaction */
        choose a random action a according to the action-values of
         the state's role;
        take action a;
        add to buffer: (s, s', r, a);
        if update assignment then
            sample experiences from buffer;
            update assignment with samples;
        end
        if update role MDP then
            sample experiences from buffer;
            update the role MDP with samples;
            update optimal policy and values of the role MDP;
        end
        /* next time step */
        s ← s'
    end
end
```

that gives the smallest maximum state-role difference is not necessarily the best performing one possible. On the contrary, for a role MDP to produce a well-performing lifted optimal policy, smaller reward and transition differences are required for actions that give higher predicted values, and larger differences can be pardoned for actions that give lower predicted values (Section 3.3.3).

To collect samples that are biased according to the action-values, we employ an exploration policy that selects actions which give higher values in the role MDP with higher probability, and vice versa. With a reasonable number of roles, the role MDP can be solved efficiently with dynamic programming. Additionally, since every time the role MDP is updated by only a small step, solving the role MPD does not have to start from scratch, but a few value iterations on top of the old solution is often enough.

In section 2.3.2, we discussed the trade-off between exploration and exploitation. A GLIE policy gives sufficient exploration and converges to a greedy policy. However, it is still unclear whether there is a GLIE policy applicable for role assignment. The main issue here is that if the policy becomes deterministically greedy, the assignment update and role update processes may give a very different result from a stochastic policy.

In this work, we use separate policies for exploration and exploitation. A Boltzmann (softmax) policy (see Section 2.3.2) is used for exploration while a lifted optimal policy is used for evaluation.

The simple example shown in Figure 5.1 makes a comparison of experience-based role assignment between random exploration and value-guided exploration.

5.1.3 Value-oriented role assignment

There is another use for the role MDP solution. Recall that the performance loss of a lifted optimal policy may occur from two places: 1) the difference between a state's optimal value in the original MDP and its role's optimal value in the role MDP, and 2) the difference between a role's optimal value in the role MDP and the value of the lifted policy. Both differences can be bounded by the assignment temporal difference (Theorem 3.13 and Theorem 3.15).

To minimize the assignment temporal difference, we have so far focused only on minimizing the reward and transition differences between states and their assigned roles because they can be used to bound the assignment temporal difference (Theorem 3.4). However in practice, it may be difficult to find every state a role that is similar enough to achieve a tight assignment

temporal difference bound. For example, an outlier state's reward and transition may be very different from all roles, but the occurrence of the state is not significant enough to win over a dedicated role. In such cases, assigning the state to even the most likely role may give it a very poor performance. On the other hand, even if the differences are large, it is still possible for a state-action pair to have a small assignment temporal difference with its assigned role.

With the solution of the role MDP, the assignment temporal difference can directly be estimated with samples. Given a piece of experience $x_t = (s_t, a_t, r_t, s_{t+1})$, we can estimate the assignment temporal difference between s_t and a role $\bar{s}$ under action a_t as

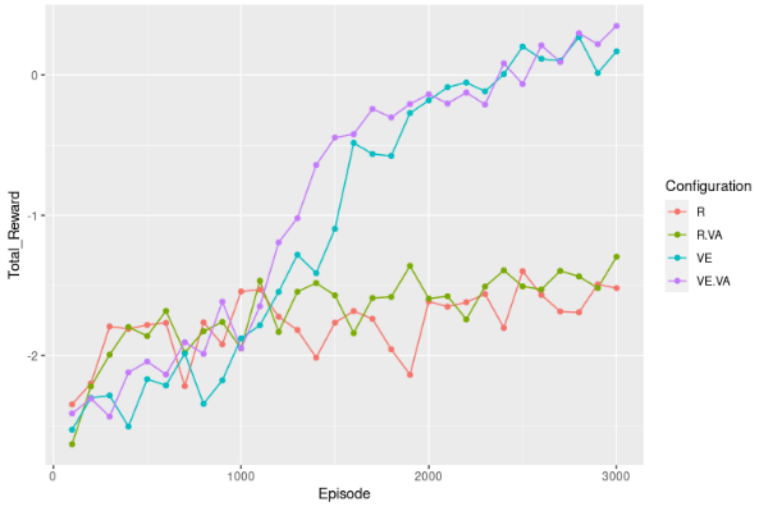
$$\begin{aligned}\mathrm{ATD}(\bar{s}, s_t, a_t) \approx \Big| r_t - \bar{R}(\bar{s}, a_t) + \\ \gamma\Big(\sum_{\bar{s}' \in \bar{S}} w(s_{t+1}, \bar{s}') V_{\bar{\pi}^\star}(\bar{s}') - \sum_{\bar{s}'' \in \bar{S}} \bar{P}(\bar{s}'', \bar{s}, a_t) V_{\bar{\pi}^\star}(\bar{s}'')\Big)\Big|. \end{aligned} \tag{5.4}$$

In addition to small reward and transition differences, the process also updates a state's assignment towards the role that gives small assignment temporal difference. Since reward difference is already part of the experience-role likelihood, the difference between the next state's role value and the next role's value, which we refer to as *estimated action-value difference* (EAVD), is included:

$$\begin{aligned}\mathcal{L}(\bar{s}|x_t) = &\exp(-c\beta_r |r_t - \bar{R}(\bar{s}, a)|) \\ &\times \exp\Big(- c\delta_{\mathrm{TV}}\Big(w(s_{t+1}, \cdot), \bar{P}(\cdot, \bar{s}, a)\Big)\Big) \\ &\times \exp\Big(- c\beta_v \gamma \Big| \sum_{\bar{s}' \in \bar{S}} w(s_{t+1}, \bar{s}') V_{\bar{\pi}^\star}(\bar{s}') - \sum_{\bar{s}'' \in \bar{S}} \bar{P}(\bar{s}'', \bar{s}, a_t) V_{\bar{\pi}^\star}(\bar{s}'')\Big|\Big). \end{aligned} \tag{5.5}$$

Similar to the reward difference, the importance of EAVD is scaled with a non-negative coefficient β_v.

EAVD is closely related to the transition difference. If there is a role that has a similar transition with a state, EAVD must also be small. However, if there is no such role available, EAVD no longer agrees with the transition difference, preventing the state to be assigned to roles that give very different values. Note that the use of EAVD is only helpful when the optimal policy is being lifted at every time step because the future roles are not predicted. When the model is also used for mixed open-loop and closed-loop control (see section 2.3.5), the benefit of having EAVD should not be expected.



Performance after 3,000 episodes				
	R	R-VA	VE	VE-VA
Mean	-1.52	-1.30	0.17	0.35
Standard deviation	1.86	1.86	1.18	1.05

Figure 5.1: Experience-based role assignment example. The agent spawns at random in one of the grid cells and can move to adjacent ones. Landing on any cell gives a -0.1 reward. From the left-most and right-most cell of the middle row, moving left and moving right gives -0.5 and 1.0 rewards respectively and terminates the environment. Tests were run in four different settings: R is random exploration; R-VA is random exploration with value-oriented assignment; VE is value-guided exploration; VE-VA is random exploration with value-oriented assignment. Each configuration is run 100 times. The plot shows the average performance every 100 episodes.

5.2 Role Assignments with Neural Networks

5.2.1 Neural network assigner

When the state space is too large or continuous, maintaining the assignment weights in a tabular form is no longer feasible. The states are often represented as vectors of variables. Instead of assigning roles to individual states, parametrized methods are used to approximate role assignments.

Let $\mathbf{x} \in \mathbb{R}^d$ be a d-dimensional vector representing a state. The role assignment approximator is a function that takes $\mathbf{x}$ as input, and outputs the assignment weights as a vector: $\mathbf{w} = F_\theta(\mathbf{x})$. In this work, we focus on using artificial neural networks as assigners.

Similar to the neural networks used in supervised learning and in other reinforcement learning tasks, the choice for intermediate layers should depend on the nature of the state space. A softmax layer is used to output a weight distribution over all roles. The targets used for training are updated assignment weights, evaluated using the assignment network itself. KL-divergence is used as the loss function.

5.2.2 Dealing with catastrophic forgetting

Catastrophic forgetting [Fre99, KPR+17, KMA+18] is the phenomenon that a neural network abruptly forgets previously learned information when learning new information. It is usually not an issue in supervised learning because a neural network is typically used for only one task and the training batches are independent and identically distributed (i.i.d) samples from the input space.

In reinforcement learning, however, the collected training data depend on the exploration policy, and learning the policy and value function for the recently experienced states may cause the forgetting of the ones experienced earlier. We note that learning the role assignment is more prone to catastrophic forgetting than other reinforcement learning algorithms because not only the neural network, but also the role MDP itself depend on the distribution of the training data.

In the last section, we mention that the experience replay [Lin92] method can be used to de-correlate the training data. Experience replay has also shown value in preventing catastrophic forgetting [RKP19, IC18, RAS+19]. In our implementation, both the experience buffer and experience sample batch are typically larger than the ones used in DQN.

Another method we employ is checkpoints. During learning, the assigner and the role MDP that give historically best performance are temporally

saved. Later after some learning, if the evaluation indicates the performance has significantly worsened (when the performance drop exceeds a certain threshold), both the assigner and the role MDP are restored.

5.3 Experiments

5.3.1 Experiment environments

The test environments we use are developed under the OpenAI Gym framework [BCP+16].

PuddleWorld

PuddleWorld is a problem environment presented in [BM95, Sut96]. An agent is tasked to navigate to the target area at the top-right corner while avoiding the two puddle zones. The problem has a continuous state space consisting of 2-dimensional real-numbered coordinates and a discrete action space of moving to four directions. The agent may choose one direction to move, but the result of the action has a small Gaussian noise. Before reaching the target zone, a small cost is given at each step. Additionally, a large penalty is given if the agent is on the puddle zone. Reaching the target zone gives a positive reward. For this work, we adopt the open-source implementation "PuddleWorld-v0" from [Ima18]. The role assignment on its 2-dimensional state space can easily be visualized. See Figure 5.2(a).

CartPole

CartPole is a problem first discussed in [BSA83] and has since become a standard benchmark environment for optimal control and reinforcement learning algorithms. In this problem, a pole is attached to a cart, which can move left or right on a frictionless track. The agent learns to control the cart so that the pole doesn't fall over while maintaining the cart within a small distance from the center. A state is defined by a vector of four real numbers representing:

1. position of the cart,
2. angle of the pole,
3. cart velocity, and
4. rate of change of the pole angle.

The controller can take one of the two actions, `left` and `right`, which applies a small force to the cart accordingly. The agent receives a positive reward at every time step, as long as the pole is no more than 15 degrees from being vertical, and the cart is no more than 2.4 units away from the center. The environment we use is "CartPole-v1", which has a maximum length of 500 steps for each episode.

Catcher

In the Catcher environment, the agent moves a paddle left and right to catch falling fruits. The control of the paddle is similar to that of the cart in CartPole problem, except that an idle action is also available, and the velocity of the paddle diminishes unless a force is applied. Successfully catching the fruit gives a positive reward while letting a fruit drop gives a negative reward and reduces a life. A state is defined by four variables:

1. player's x position,
2. player's velocity,
3. fruit's x position, and
4. fruit's y position.

The test environment is powered by PyGame Learning Environment (PLE) [Tas16] and the implementation "Catcher-v0" is adopted from [Sob20]. We modified the termination condition so that each episode has at most 5000 time steps, and the agent has only one life every episode as opposed to 3 in the default setting.

5.3.2 Experiments

Exploratory analysis with PuddleWorld

Since a state in PuddleWorld is represented as a two-dimensional coordinate, it is very easy to visualize how the abstract knowledge formed by the learning agent looks like. After 50,000 time steps (25 episodes), we take 100×100 coordinates evenly from the space, and colour them by their most likely roles. The results are shown in Figure 5.2.

PuddleWorld is a very easy task for most of the reinforcement learning algorithms. This is no exception for role assignment. With an 0.3-greedy exploration policy (0.3 probability to choose the action that is estimated by the role MDP to have the highest value, and 0.7 probability to choose

an action at random) and $\beta = 0.02$, an exploitable partition can easily be found. The partition is different each time the experiment is run, and here we show one of the examples in in Figure 5.2(b).

In this example, under any action, every role has a relatively high probability to transition to itself, but may also have a smaller, yet still significant probability to transition to another role. The agent can use this partition and the outcomes of the roles to reason: the red role marks the target zone with a high reward, and the dandelion role has a relatively high value because it has a chance to reach the red zone with the action `move-up`; the policy chooses the action `move-right` for the blue role because it has a chance to reach the dandelion role, etc.

Our method has a larger number of hyperparameters compared to model-free and other model-based methods (see Appendix E for the full list of hyperparameters). Apart from the ones that are already commonly used in other reinforcement learning methods, we also need to specify the ones that are related to the role assignment and the role MDP. Choosing the number of roles m is an important task that requires further investigation. Fortunately for our tests, it is not difficult. With some human understanding of the problem, we may be able to come up with an estimation of how many situations the problem can be described with. Even if human knowledge is absent, our tests show that the performance is not sensitive to m as long as it surpasses a certain number based on the environment. We have observed that, using a relatively generous budget of roles, there will be some of the roles that are simply not assigned to with a notable weight by any state. For this, we use $m = 10$. We are also interested in how different exploration strategies and β affect the role assignment. The results are shown in subfigures in 5.2.

By changing the exploration policy to a uniform probability for all actions, it often ends up in an unexploitable partition. One example is shown in Figure 5.2(c). The solution to the role MDP gives the yellow role `move-down` action because green is of higher value. Meanwhile, it also gives green role `move-up` action because it may reach the target zone with a high reward. Lifting such a policy, the agent may be stuck in a loop between green and yellow roles. On the other hand, when the exploration policy is too greedy ($\epsilon = 0.8$), the agent fails to explore the state space (Figure 5.2(d)).

Since we want the role assignment to be based on the similarity of both reward and transition, it is important to scale the reward differences to a comparable level with transition differences (as total variation distances). Using a too-small β often causes trivial assignments while using a too large β often causes the transition difference to be ignored. Examples are shown

in Figure 5.2(e) and 5.2(f).

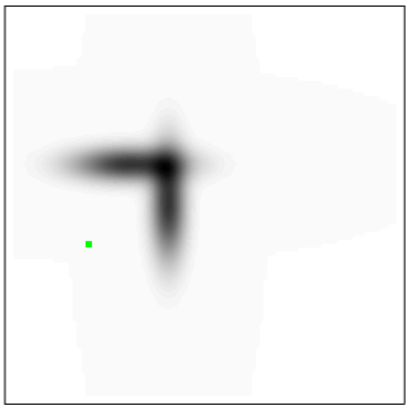
Performance tests with CartPole and Catcher

For reinforcement learning problems, the performance of a policy is typically defined as the expected sum of rewards that can be collected within an episode following that policy. The performance of role assignment in CartPole and Catcher is tested as follows.

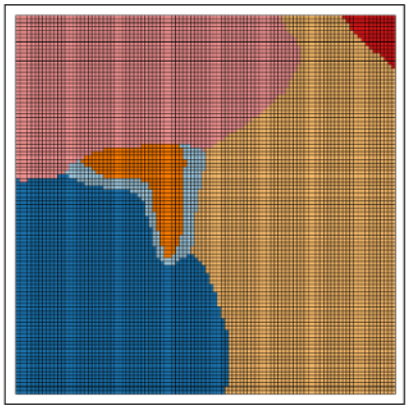
Each test consists of 100 training sessions. Each training session consists of 100 training episodes. At the start of a training session, the assigner and the role MDP are randomly initialized. Training takes place between time steps, and uses only the experience collected so far within the current training session. After every episode, the performance is evaluated 10 times with randomly initialized environments. See Appendix D for the details. Without training additional models, we also tested the usefulness of using the role MDP for mixed open-loop and closed-loop control.

In the toy problem from the last section, the inclusion of EAVD gives very little improvement. In CartPole and Catcher however, a small value of β_v allows the agent to learn an assignment faster and better. Without using EAVD, the agent fails to see any performance improvement in Catcher. On the other hand, not using EAVD has its advantage when the role MDP is used for mixed open-loop and closed-loop control. See Figure 5.5 and 5.6.

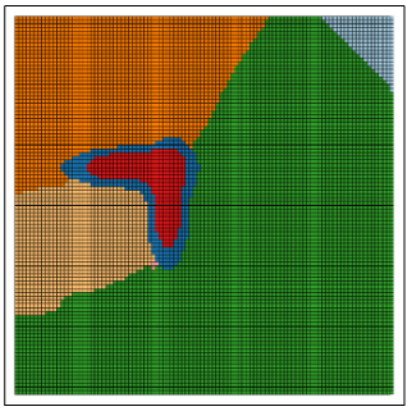
Role assignment can be seen as an abstract-model-based method. We compare its sample efficiency with DQN, Actor-Critic and MBPO algorithms. The results in CartPole and Catcher show that the sample efficiency of role assignment is much better than model-free methods, and has comparable sample efficiency to MBPO. See Figure 5.7. The hyperparameters used for the tests are listed in Appendix F.



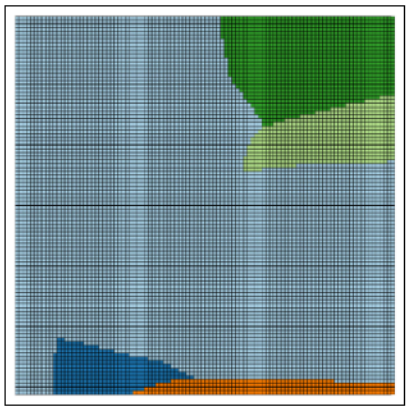
(a) PuddleWorld-v0 environment



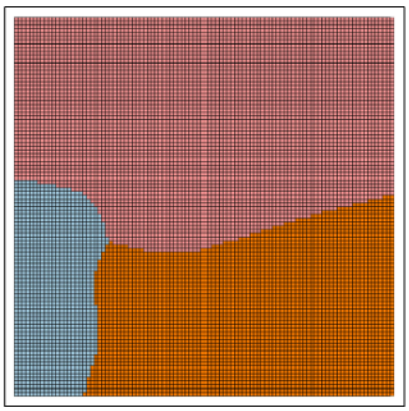
(b) 0.3-greedy exploration policy, $\beta = 0.02$.



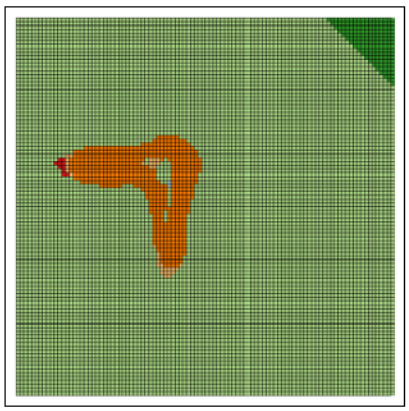
(c) 0-greedy (uniform) exploration policy.



(d) 0.8-greedy exploration policy.



(e) $\beta = 0.002$.



(f) $\beta = 0.2$.

Figure 5.2: Assignment on Puddle World environment. Colors represent the most likely roles of the coordinates.

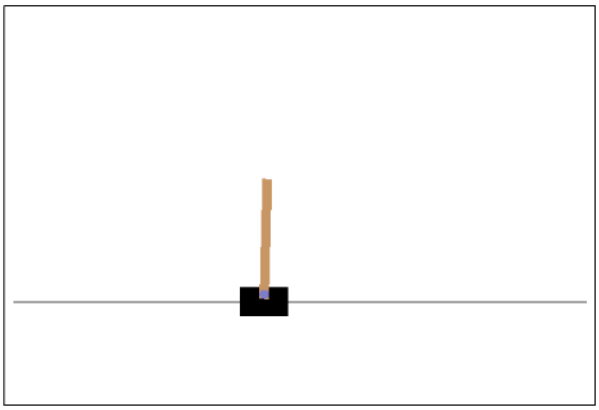


Figure 5.3: CartPole-v1 Environment

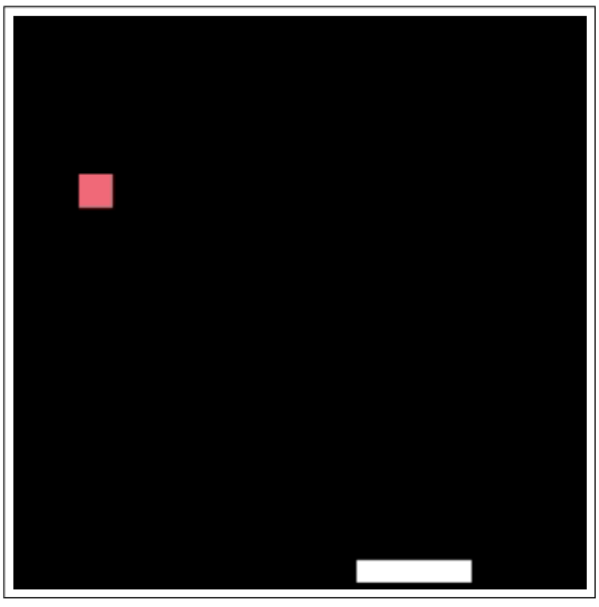


Figure 5.4: Catcher-v0 Environment

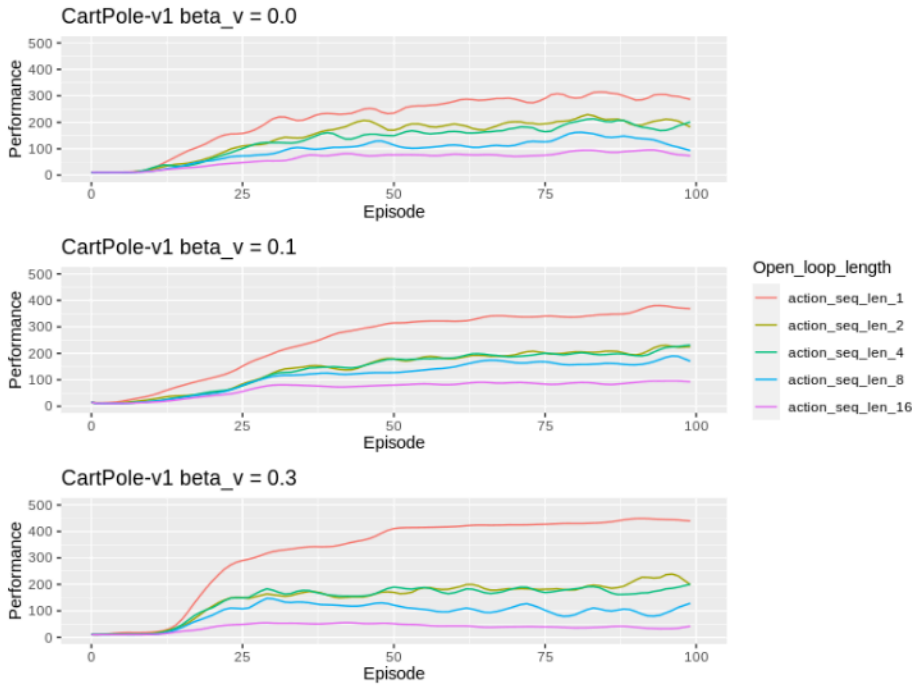


Figure 5.5: Performance evaluation with different values of β_v in CartPole-v1 environment.

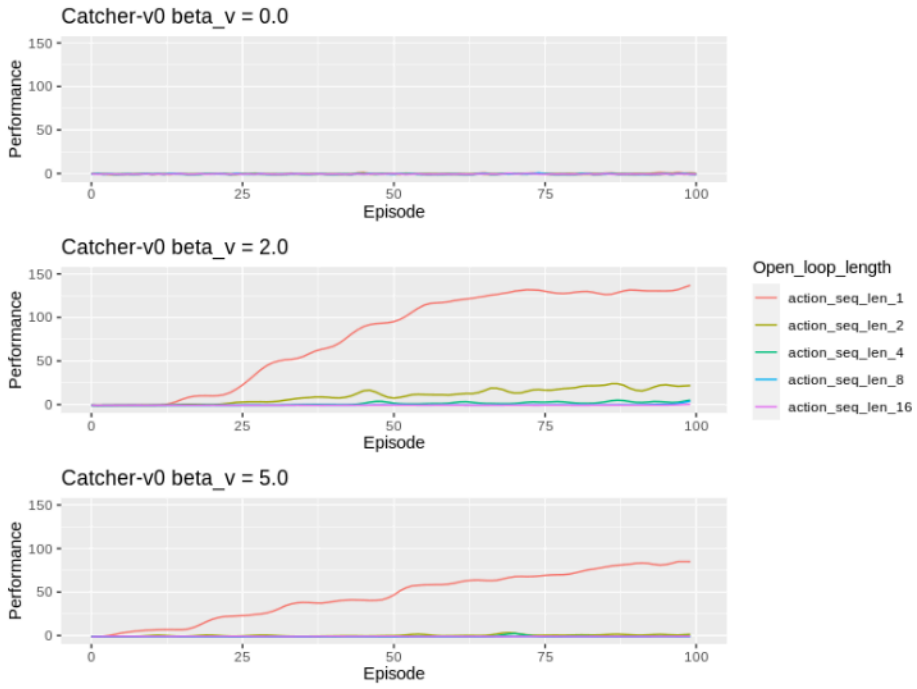


Figure 5.6: Performance evaluation with different values of β_v in Catcher-v0 environment.

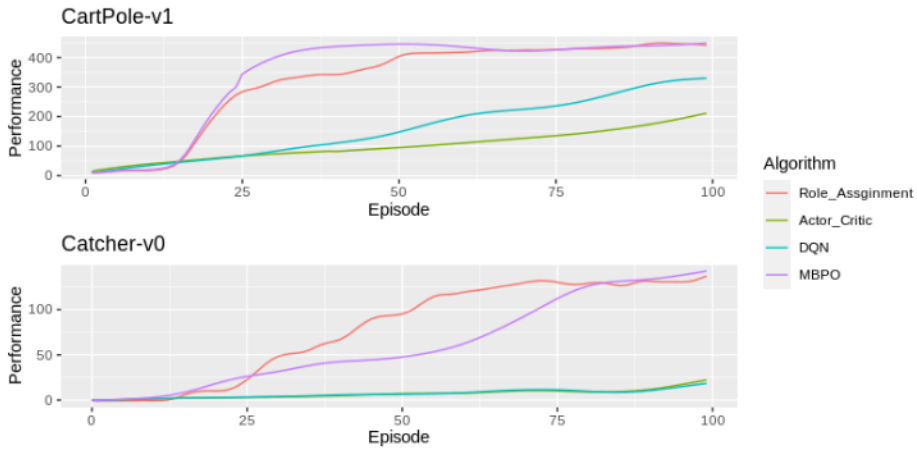


Figure 5.7: Benchmark with other RL algorithms. The sample efficiency of MDP role assignment is significantly better than model-free methods, and is comparable to MBPO.

Chapter 6

Final Remarks

6.1 Conclusions

In this thesis, we use role assignments on MDPs to build abstract models for agents to plan on. It is an attempt to contribute to the work that aims at empowering an learning AI to form its own abstract understanding of the problem and use it for reasoning. Without defining the rules with human knowledge, the role MDP extracts an abstract rule system. When a neural network is used as the assigner, the abstraction is also fundamentally different from that of autoencoders because the role of a state is defined by the states it transitions to so only the task-specific representations are needed. In the context of reinforcement learning, role assignment is a very unconventional approach because neither a policy nor a value function is learned, and action selection is through reasoning with roles.

Unlike bisimulation metrics and other methods that are based on state-state similarities, our algorithms are based on the similarities between states and roles. This greatly improves memory and time efficiency, and makes it possible to be applied on unknown MDPs. How exactly they find an assignment is not easily understood because transition-to-roles changes over time. Our work on the true similarity reveals part of the underlying mechanisms.

Despite being unconventional, our tests have shown that role assignment has great potential for solving real problems. It may outperform conventional RL algorithms in many tasks. Compared to model-free methods, it also has better sample efficiency and explainability.

6.2 Future Work

The methods discussed in this thesis have primarily focused on MDPs with discrete action spaces. Adaptation to continuous action spaces will be a valuable extension. There are two directions for this: 1) using a continuous-action role MDP, or 2) discretization and abstraction of actions.

Compared to other reinforcement learning algorithms, role assignment

methods presented in this work use more hyperparameters. Tuning them can be challenging. It may be possible to reduce the number of hyperparameters with more in-depth knowledge of how they work. Another promising direction is to have the learning agent dynamically adjust them during training. For example, the number of roles to be used may change on-demand, and concentration varies according to the cascading shape of true similarity.

In this work, the distance between a state and a role is calculated as a linear combination of the absolute difference between the rewards, and the total variation difference between the transitions (4.1). We chose this definition because of its relation to bisimulation metrics and the performance bound. We note the possibilities of using other ways to measure the similarity between a state and a role. For example, the reward difference can be calculated as a squared sum, which changes the likelihood from Laplace distribution to Gaussian, and true similarity can still be derived, in a slightly different form.

Bibliography

[Abb15] Pieter Abbeel. Optimal control for linear dynamical systems and quadratic cost. *UC Berkley EECS*, 2015. → pages 13

[Abe19] David Abel. A theory of state abstraction for reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 9876–9877, 2019. → pages 18, 19

[AF18] Masataro Asai and Alex Fukunaga. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. → pages 4

[AFDM17] Alexander A. Alemi, Ian Fischer, Joshua V. Dillon, and Kevin Murphy. Deep variational information bottleneck. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. → pages 4

[AML18] Kavosh Asadi, Dipendra Misra, and Michael Littman. Lipschitz continuity in model-based reinforcement learning. In *International Conference on Machine Learning*, pages 264–273. PMLR, 2018. → pages 13

[BBC57] R. Bellman, R.E. Bellman, and Rand Corporation. *Dynamic Programming*. Rand Corporation research study. Princeton University Press, 1957. → pages 5, 6, 8

[BBV04] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004. → pages 47

[BCP+16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba.

Openai gym. *arXiv preprint arXiv:1606.01540*, 2016. → pages 70

[BD94] Craig Boutilier and Richard Dearden. Using abstractions for decision-theoretic planning with time constraints. In *AAAI*, pages 1016–1022, 1994. → pages 18, 19

[BDF92] Vladimir Batagelj, Patrick Doreian, and Anuška Ferligoj. An optimizational approach to regular equivalence. *Social networks*, 14(1-2):121–135, 1992. → pages 17

[BE89] Stephen P Borgatti and Martin G Everett. The class of all regular equivalences: Algebraic structure and computation. *Social networks*, 11(1):65–88, 1989. → pages 15

[BE92] Stephen P Borgatti and Martin G Everett. Notions of position in social network analysis. *Sociological methodology*, pages 1–35, 1992. → pages 14, 31

[BH79] J Bretagnolle and C Huber. Estimation des densités: risque minimax. *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete*, 47(2):119–137, 1979. → pages 54

[BIF+21] David Benrimoh, Sonia Israel, Robert Fratila, Caitrin Armstrong, Kelly Perlman, Ariel Rosenfeld, and Adam Kapelner. Ml and ai safety, effectiveness and explainability in healthcare. *Frontiers in big Data*, 4, 2021. → pages 4

[BL10] Ulrik Brandes and Jürgen Lerner. Structural similarity: Spectral methods for relaxed blockmodeling. *Journal of classification*, 27(3):279–306, 2010. → pages 17

[BM95] Justin Boyan and Andrew W Moore. Generalization in reinforcement learning: Safely approximating the value function. *Advances in neural information processing systems*, pages 369–376, 1995. → pages 70

[Bra86] Valentino Braitenberg. *Vehicles: Experiments in synthetic psychology*. MIT press, 1986. → pages 3

[Bro86] Rodney Brooks. A robust layered control system for a mobile robot. *IEEE journal on robotics and automation*, 2(1):14–23, 1986. → pages 3

[BSA83] Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 1983. → pages 70

[Buc05] Bruce G Buchanan. A (very) brief history of artificial intelligence. *Ai Magazine*, 26(4):53–53, 2005. → pages 3

[Bur76] Ronald S Burt. Positions in networks. *Social forces*, 55(1):93–122, 1976. → pages 15

[Cas98] Anthony Rocco Cassandra. *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, USA, 1998. AAI9830418. → pages 45

[DBF05] Patrick Doreian, Vladimir Batagelj, and Anuska Ferligoj. *Generalized blockmodeling*. Number 25. Cambridge university press, 2005. → pages 17

[DDSGA13] Manlio De Domenico, Albert Solé, Sergio Gómez, and Alex Arenas. Random walks on multiplex networks. *arXiv preprint arXiv:1306.0519*, 2013. → pages 37

[d'E63] Francois d'Epenoux. A probabilistic production and inventory problem. *Management Science*, 10(1):98–108, 1963. → pages 6

[EB91] Martin G Everett and Steve Borgatti. Role colouring a graph. *Mathematical Social Sciences*, 21(2):183–188, 1991. → pages 31

[EB94] Martin G Everett and Stephen P Borgatti. Regular equivalence: General theory. *Journal of mathematical sociology*, 19(1):29–52, 1994. → pages 16

[Eve85] Martin G Everett. Role similarity and complexity in social networks. *Social Networks*, 7(4):353–359, 1985. → pages 15

[FDB11] Anuska Ferligoj, Patrick Doreian, and Vladimir Batagelj. Positions and roles. *The SAGE handbook of social network analysis*, pages 434–446, 2011. → pages 17

[Fer07] Norman Francis Ferns. *State-similarity metrics for continuous Markov decision processes*. PhD thesis, McGill University, 2007. → pages 2, 30, 35, 48

[FPP04] Norm Ferns, Prakash Panangaden, and Doina Precup. Metrics for finite markov decision processes. In *UAI*, volume 4, pages 162–169, 2004. → pages 2, 18, 26, 28, 30, 35, 48

[FPP05] Norm Ferns, Prakash Panangaden, and Doina Precup. Metrics for markov decision processes with infinite state spaces. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, UAI'05, page 201–208, Arlington, Virginia, USA, 2005. AUAI Press. → pages 18, 28, 30, 48

[FPP11] Norm Ferns, Prakash Panangaden, and Doina Precup. Bisimulation metrics for continuous markov decision processes. *SIAM Journal on Computing*, 40(6):1662–1714, 2011. → pages 18, 28, 30, 48

[Fre99] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999. → pages 69

[GDG03] Robert Givan, Thomas Dean, and Matthew Greig. Equivalence notions and model minimization in markov decision processes. *Artificial Intelligence*, 147(1-2):163–223, 2003. → pages 1, 18, 20, 22, 23, 24, 25, 28

[GGR19] Lauren Gordon, Teodor Grantcharov, and Frank Rudzicz. Explainable artificial intelligence for safe intraoperative decision support. *JAMA surgery*, 154(11):1064–1065, 2019. → pages 4

[GKPV03] Carlos Guestrin, Daphne Koller, Ronald Parr, and Shobha Venkataraman. Efficient solution algorithms for factored mdps. *Journal of Artificial Intelligence Research*, 19:399–468, 2003. → pages 25

[GMW08] Alyssa Glass, Deborah L McGuinness, and Michael Wolverton. Toward establishing trust in adaptive agents. In *Proceedings of the 13th international conference on Intelligent user interfaces*, pages 227–236, 2008. → pages 4

[GW92] Fausto Giunchiglia and Toby Walsh. A theory of abstraction. *Artificial intelligence*, 57(2-3):323–389, 1992. → pages 18, 19

[HBZ96] Eric Hansen, Andrew Barto, and Shlomo Zilberstein. Reinforcement learning for mixed open-loop and closed-loop con-

trol. *Advances in Neural Information Processing Systems*, 9, 1996. → pages 14

[How60] R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA, 1960. → pages 6

[HR05] Robert A Hanneman and Mark Riddle. *Introduction to social network methods*. University of California Riverside, 2005. → pages 16, 17

[IC18] David Isele and Akansel Cosgun. Selective experience replay for lifelong learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. → pages 69

[Ima18] Ehsan Imani. Ehsanei/gym-puddle: Puddle world environment for openai gym. `https://github.com/EhsanEI/gym-puddle`, 2018. → pages 70

[Jan20] Michael Janner. Model-based reinforcement learning: Theory and practice. *BAIR Blog*, 2020. → pages 12

[JFZL19] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. *When to Trust Your Model: Model-Based Policy Optimization*. Curran Associates Inc., Red Hook, NY, USA, 2019. → pages 13

[JJN22] Mehdi Jafarnia-Jahromi, Rahul Jain, and Ashutosh Nayyar. Online learning for unknown partially observable mdps. In Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera, editors, *International Conference on Artificial Intelligence and Statistics, AISTATS 2022, 28-30 March 2022, Virtual Event*, volume 151 of *Proceedings of Machine Learning Research*, pages 1712–1732. PMLR, 2022. → pages 8

[KAA21] Mete Kemertas and Tristan Aumentado-Armstrong. Towards robust bisimulation metric learning. *Advances in Neural Information Processing Systems*, 34:4764–4777, 2021. → pages 48

[KAB+14] Mikko Kivelä, Alex Arenas, Marc Barthelemy, James P Gleeson, Yamir Moreno, and Mason A Porter. Multilayer networks. *Journal of complex networks*, 2(3):203–271, 2014. → pages 31

[KBM+20] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H. Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model based reinforcement learning for atari. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. → pages 13

[Kem76] John Kemeny. *Finite Markov chains*. Springer-Verlag, New York, 1976. → pages 23

[KK99] Michael Kearns and Daphne Koller. Efficient reinforcement learning in factored mdps. In *IJCAI*, volume 16, pages 740–747, 1999. → pages 25

[KLC95] LP Kaebling, ML Littman, and AR Cassandra. Learning policies for partially observable environments: Scaling up. In *Proceedings of the Twelfth International Conference on Machine Learning*, 1995. → pages 45

[KLC98] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998. → pages 7

[KMA+18] Ronald Kemker, Marc McClure, Angelina Abitino, Tyler Hayes, and Christopher Kanan. Measuring catastrophic forgetting in neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. → pages 69

[Kol82] Gina Kolata. How can computers get common sense? two of the founders of the field of artificial intelligence disagree on how to make a thinking machine. *Science*, 217(4566):1237–1238, 1982. → pages 3

[KP14] Volodymyr Kuleshov and Doina Precup. Algorithms for multi-armed bandit problems. *arXiv preprint arXiv:1402.6028*, 2014. → pages 10

[KPR+17] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan,

John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017. → pages 69

[LDK95] Michael L. Littman, Thomas L. Dean, and Leslie Pack Kaelbling. On the complexity of solving markov decision problems. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, UAI'95, page 394–402, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. → pages 6, 7

[Ler05] Jürgen Lerner. Role assignments. In *Network analysis*, pages 216–252. Springer, 2005. → pages 14, 16, 31

[LHN06] Elizabeth A Leicht, Petter Holme, and Mark EJ Newman. Vertex similarity in networks. *Physical Review E*, 73(2):026120, 2006. → pages 17

[Lin92] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992. → pages 64, 69

[LP17] David A Levin and Yuval Peres. *Markov chains and mixing times*, volume 107. American Mathematical Soc., 2017. → pages 27

[LW71] Francois Lorrain and Harrison C White. Structural equivalence of individuals in social networks. *The Journal of mathematical sociology*, 1(1):49–80, 1971. → pages 15

[LWL06] Lihong Li, Thomas J Walsh, and Michael L Littman. Towards a unified theory of state abstraction for mdps. *ISAIM*, 4:5, 2006. → pages 18, 19, 20, 35

[MHC99] Omid Madani, Steve Hanks, and Anne Condon. On the undecidability of probabilistic planning and infinite-horizon partially observable markov decision problems. In *AAAI/IAAI*, pages 541–548, 1999. → pages 8, 45

[MKS+15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al.

Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015. → pages 3

[Mon82] George E Monahan. State of the art—a survey of partially observable markov decision processes: theory, models, and algorithms. *Management science*, 28(1):1–16, 1982. → pages 14

[MPKK99] Nicolas Meuleau, Leonid Peshkin, Kee-Eung Kim, and Leslie Pack Kaelbling. Learning finite-state controllers for partially observable environments. In Kathryn B. Laskey and Henri Prade, editors, *UAI '99: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, Stockholm, Sweden, July 30 - August 1, 1999*, pages 427–436. Morgan Kaufmann, 1999. → pages 8

[Nil09] Nils J Nilsson. *The quest for artificial intelligence*. Cambridge University Press, 2009. → pages 3

[NKFL18] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018. → pages 13

[NS76] Allen Newell and Herbert A Simon. Computer science as empirical inquiry: symbols and search. *Communications of the ACM*, 19(3):113–126, 1976. → pages 3

[Pre00] Doina Precup. *Temporal abstraction in reinforcement learning*. University of Massachusetts Amherst, 2000. → pages 14

[PT87] Christos H Papadimitriou and John N Tsitsiklis. The complexity of markov decision processes. *Mathematics of operations research*, 12(3):441–450, 1987. → pages 6, 45

[RAS+19] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. *Advances in Neural Information Processing Systems*, 32, 2019. → pages 69

[RB02] Balaraman Ravindran and Andrew G Barto. Model minimization in hierarchical reinforcement learning. In *International*

Symposium on Abstraction, Reformulation, and Approximation, pages 196–211. Springer, 2002. → pages 1, 18, 23

[RB04] Balaraman Ravindran and Andrew G Barto. Approximate homomorphisms: A framework for non-exact minimization in markov decision processes. *In International Conference on Knowledge Based Computer Systems*, 2004. → pages 1, 18, 26, 27, 28, 35, 48

[Ric05] Arthur George Richards. *Robust constrained model predictive control*. PhD thesis, Massachusetts Institute of Technology, 2005. → pages 13

[RKP19] Mohammad Rostami, Soheil Kolouri, and Praveen K. Pilly. Complementary learning for overcoming catastrophic forgetting using experience replay. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, IJCAI'19, page 3339–3345. AAAI Press, 2019. → pages 69

[RN10] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition, 2010. → pages 3

[RPPCD08] Stéphane Ross, Joelle Pineau, Sébastien Paquet, and Brahim Chaib-Draa. Online planning algorithms for pomdps. *Journal of Artificial Intelligence Research*, 32:663–704, 2008. → pages 47

[RW07] Jörg Reichardt and Douglas R White. Role models for complex networks. *The European Physical Journal B*, 60(2):217–224, 2007. → pages 17

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018. → pages 6, 10, 12

[SDL07] Alexander L Strehl, Carlos Diuk, and Michael L Littman. Efficient structure learning in factored-state mdps. In *AAAI*, volume 7, pages 645–650, 2007. → pages 25

[Sea80] John R Searle. Minds, brains, and programs. *Behavioral and brain sciences*, 3(3):417–424, 1980. → pages 3

[SHM+16] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser,

Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016. → pages 3

[SMSM00] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 2000. → pages 10

[Sob20] Luis Sobrecueva. Lusob/gym-ple: Ple as a gym environment. `https://github.com/lusob/gym-ple`, 2020. → pages 71

[SS09] Jonathan Sorg and Satinder Singh. Transfer via soft homomorphisms. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 741–748, 2009. → pages 46, 94

[SSS+17] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017. → pages 3

[Sut91] Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991. → pages 13

[Sut96] Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in neural information processing systems*, pages 1038–1044, 1996. → pages 70

[Tas16] Norman Tasfi. Pygame learning environment. `https://github.com/ntasfi/PyGame-Learning-Environment`, 2016. → pages 71

[TE20] Nicolás I Tapia and Pablo A Estévez. On the information plane of autoencoders. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2020. → pages 4

[TET12] Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory

optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4906–4913. IEEE, 2012. → pages 13

[TM13] Hui Tan and Shaohui Ma. Learning partially observable markov decision model with em algorithm. In *2013 7th International Conference on Application of Information and Communication Technologies*, pages 1–4. IEEE, 2013. → pages 8

[TPB99] Naftali Tishby, Fernando C. Pereira, and William Bialek. The information bottleneck method. In *Proc. of the 37-th Annual Allerton Conference on Communication, Control and Computing*, pages 368–377, 1999. → pages 4

[TPP08] Jonathan Taylor, Doina Precup, and Prakash Panagaden. Bounding performance loss in approximate mdp homomorphisms. *Advances in Neural Information Processing Systems*, 21:1649–1656, 2008. → pages 18, 24, 26, 48

[WBC+19] Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. Benchmarking model-based reinforcement learning. *arXiv preprint arXiv:1907.02057*, 2019. → pages 12

[WF+94] Stanley Wasserman, Katherine Faust, et al. *Social network analysis: Methods and applications*. Cambridge university press, 1994. → pages 14

[Whi78] Ward Whitt. Approximations of dynamic programs, i. *Mathematics of Operations Research*, 3(3):231–243, 1978. → pages 18, 26, 27

[Win88] Christopher Winship. Thoughts about roles and relations: An old document revisited. *Social Networks*, 1988. → pages 15

[WR83] Douglas R White and Karl P Reitz. Graph and semigroup homomorphisms on networks of relations. *Social Networks*, 5(2):193–234, 1983. → pages 15

[WSBR15] Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. *Advances in neural information processing systems*, 28, 2015. → pages 4

Appendices

Appendix A

Equivalence of Exact Soft Role Assignments

From a starting state s_0, given an action sequence $\mathcal{A} = \{a_0, a_1, ..., a_k\}$, the expected reward at each time step is the same as the weighted average of the state's roles' expected reward.

Proof. Let a row vector $\mathbf{w}_0^\intercal$ be the assignment weights for the initial state s_0. Let the probability distribution of all states at time t be represented as a row vector $\mathbf{s}_t^\intercal$, and the roles' probability distribution be represented as $\mathbf{w}_t^\intercal$. If the two distributions satisfy $\mathbf{s}_t^\intercal \mathbf{W} = \mathbf{w}_t^\intercal$, then after taking some action a_t,

1. the expected reward is $\mathbf{s}_t^\intercal \mathbf{r}^{(a_t)}$, and the expected reward from the roles is $\mathbf{w}_t^\intercal \bar{\mathbf{r}}^{(a_t)} = \mathbf{s}_t^\intercal \mathbf{W} \bar{\mathbf{r}}^{(a_t)}$, thus the two expected rewards are the same;

2. the state distribution becomes $\mathbf{s}_{t+1}^\intercal = \mathbf{s}_t^\intercal \mathbf{P}^{(a_t)}$ and the role distribution becomes $\mathbf{w}_{t+1}^\intercal = \mathbf{w}_t^\intercal \bar{\mathbf{P}}^{(a_t)} = \mathbf{s}_t^\intercal \mathbf{W} \bar{\mathbf{P}}^{(a_t)}$; the two distributions will also satisfy $\mathbf{s}_{t+1}^\intercal \mathbf{W} = \mathbf{w}_{t+1}$ based on our exact soft role assignment definition.

For the base case, we know that at the beginning, $\mathbf{s}_0^\intercal \mathbf{W} = \mathbf{w}_0^\intercal$ simply because $\mathbf{w}_0$ is the column in $\mathbf{W}$ that corresponds to beginning state s_0. □

Appendix B

Q-MDP Approximation Performance

Lemma B.1 (Q-MDP total advantage bound (Lemma 2 in [SS09]))**.** *Let $D(s)$ be the dominance of s, i.e., the largest assignment weight to a role of state s. We have:*

$$\sum_{\bar{s}\in\bar{S}} w(s,\bar{s})V_{\bar{\pi}^\star}(\bar{s}) - V_{\bar{\pi}^\star_{Q\text{-}MDP}}(s) \leq (1-D(s))\delta_{\bar{A}^\star}$$

where $\delta_{\bar{A}^\star}$ is the largest advantage in the role MDP under an optimal policy:

$$\delta_{\bar{A}^\star} = \max_{a_1,a_2\in A,\bar{s}\in\bar{S}} |Q_{\bar{\pi}^\star}(\bar{s},a_1) - Q_{\bar{\pi}^\star}(\bar{s},a_2)|$$

.

The difference between the value of an optimal policy and the value of a lifted optimal policy can be bounded with two parts:

$$\begin{aligned} & V_{\pi^\star}(s) - V_{\bar{\pi}^\star_{\text{Q-MDP}}}(s) \\ = \; & \underbrace{V_{\pi^\star}(s) - V_{\bar{\pi}^\star_{\text{Q-MDP}}}(s)}_{\text{Part I}} + \underbrace{V_{\bar{\pi}^\star_{\text{Q-MDP}}}(s) - V_{\tilde{\pi}^\star_{\text{Q-MDP}}}(s)}_{\text{Part II}} \end{aligned} \tag{B.1}$$

The bounds of both parts involve a special action-value function $Q_{\hat{\pi}^\star_{\text{Q-MDP}}}$ that combines the immediate reward from an action and the next state's role value:

$$Q_{\hat{\pi}^\star_{\text{Q-MDP}}}(s,a) = R(s,a) + \gamma \sum_{s'} P(s',s,a) V_{\bar{\pi}^\star_{\text{Q-MDP}}}(s') \tag{B.2}$$

and the maximum difference between $Q_{\hat{\pi}^\star_{\text{Q-MDP}}}(s,a)$ and $Q_{\bar{\pi}^\star_{\text{Q-MDP}}}(s,a)$:

$$\begin{aligned} K(Q_{\bar{\pi}^\star_{\text{Q-MDP}}}) = & \max_{a\in A, s\in S} |Q_{\hat{\pi}^\star_{\text{Q-MDP}}}(s,a) - Q_{\bar{\pi}^\star_{\text{Q-MDP}}}(s,a)| \\ = & \max_{a\in A, s\in S} |R(s,a) - \sum_{\bar{s}\in\bar{S}} w(s,\bar{s})\bar{R}(\bar{s},a) \\ & + \gamma \sum_{s'} P(s',s,a) V_{\bar{\pi}^\star_{\text{Q-MDP}}}(s') \\ & - \gamma \sum_{\bar{s}\in\bar{S}} w(s,\bar{s}) \sum_{\bar{s}'\in\bar{S}} \bar{P}(\bar{s}',\bar{s},a) V_{\bar{\pi}^\star}(\bar{s}')| \end{aligned} \tag{B.3}$$

Similar to what we did in section 3.3, it is also useful to define $K(V_{\bar{\pi}^\star_{\text{Q-MDP}}})$ as a special case of $K(Q_{\tilde{\pi}^\star_{\text{Q-MDP}}})$ with the actions being the ones selected by the lifted optimal policy:

$$K(V_{\tilde{\pi}^\star_{\text{Q-MDP}}}) = \max_{s\in S} |Q_{\hat{\pi}^\star_{\text{Q-MDP}}}(s, \tilde{\pi}^\star_{\text{Q-MDP}}(s)) - V_{\bar{\pi}^\star_{\text{Q-MDP}}}(s)| \tag{B.4}$$

and $K(Q^\dagger_{\tilde{\pi}^\star_{\text{Q-MDP}}})$ as a modified version of $K(Q_{\tilde{\pi}^\star_{\text{Q-MDP}}})$ with an advantage subtracted:

$$\begin{aligned} K(Q^\dagger_{\tilde{\pi}^\star_{\text{Q-MDP}}}) = \max_{s\in S, a\in A} [& |Q_{\hat{\pi}^\star_{\text{Q-MDP}}}(s,a) - Q_{\bar{\pi}^\star_{\text{Q-MDP}}}(s,a)| \\ & - (V_{\bar{\pi}^\star_{\text{Q-MDP}}}(s) - Q_{\bar{\pi}^\star_{\text{Q-MDP}}}(s,a))] \end{aligned} \tag{B.5}$$

For Part I in B.1:

Theorem B.2 (Value difference between optimal policy and Q-MDP approximation)**.**

$$V_{\pi^\star} - V_{\bar{\pi}^\star_{Q\text{-}MDP}} \leq \frac{1}{1-\gamma} K(Q^\dagger_{\tilde{\pi}^\star_{Q\text{-}MDP}}) \tag{B.6}$$

Proof. From a state s, let $a^\star = \pi^\star(s)$ be the action picked by an optimal policy, and $\tilde{a}^\star = \pi^\star_{\text{Q-MDP}}(s)$ be the action picked by a Q-MDP policy. Then,

$$\begin{aligned} & V_{\pi^\star}(s) - V_{\bar{\pi}^\star_{\text{Q-MDP}}}(s) \\ = \; & Q_{\pi^\star}(s,a^\star) - Q_{\bar{\pi}^\star_{\text{Q-MDP}}}(s,\tilde{a}^\star) \\ = \; & Q_{\pi^\star}(s,a^\star) - Q_{\hat{\pi}^\star}(s,a^\star) \\ & + Q_{\hat{\pi}^\star}(s,a^\star) - Q_{\bar{\pi}^\star_{\text{Q-MDP}}}(s,a^\star) \\ & + Q_{\bar{\pi}^\star_{\text{Q-MDP}}}(s,a^\star) - Q_{\bar{\pi}^\star_{\text{Q-MDP}}}(s,\tilde{a}^\star) \\ \leq \; & \gamma \sum_{s'\in S} P(s',s,a^\star)(V_{\pi^\star}(s') - V_{\bar{\pi}^\star_{\text{Q-MDP}}}(s')) + K(Q^\dagger_{\tilde{\pi}^\star_{\text{Q-MDP}}}) \end{aligned}$$

□

For Part II in B.1:

Theorem B.3 (Value difference between Q-MDP approximation and lifted Q-MDP policy)**.**

$$V_{\bar{\pi}^\star_{Q\text{-}MDP}}(s) - V_{\tilde{\pi}^\star_{Q\text{-}MDP}}(s) \leq \frac{1}{1-\gamma} K(V_{\tilde{\pi}^\star_{Q\text{-}MDP}}) \leq \frac{1}{1-\gamma} K(Q^\dagger_{\tilde{\pi}^\star_{Q\text{-}MDP}}) \quad \text{(B.7)}$$

Proof. The second inequality is obvious. We consider the first. From a state s, let $\tilde{a}^\star = \bar{\pi}^\star_{\text{Q-MDP}}(s)$ be the action picked by a Q-MDP policy. Then,

$$\begin{aligned}
& V_{\bar{\pi}^\star_{\text{Q-MDP}}}(s) - V_{\tilde{\pi}^\star_{\text{Q-MDP}}}(s) \\
= \;& Q_{\bar{\pi}^\star_{\text{Q-MDP}}}(s, \tilde{a}^\star) - Q_{\hat{\pi}^\star_{\text{Q-MDP}}}(s, \tilde{a}^\star) \\
& + Q_{\hat{\pi}^\star_{\text{Q-MDP}}}(s, \tilde{a}^\star) - Q_{\tilde{\pi}^\star_{\text{Q-MDP}}}(s, \tilde{a}^\star) \\
\leq \;& K(V_{\tilde{\pi}^\star_{\text{Q-MDP}}}) + \gamma \sum_{s'} P(s', s, \tilde{a}^\star)(V_{\bar{\pi}^\star_{\text{Q-MDP}}}(s') - V_{\tilde{\pi}^\star_{\text{Q-MDP}}}(s'))
\end{aligned}$$

$$\begin{aligned}
& V_{\bar{\pi}^\star_{\text{Q-MDP}}}(s) - V_{\tilde{\pi}^\star_{\text{Q-MDP}}}(s) \\
= \;& \sum_{\bar{s} \in \bar{S}} w(s, \bar{s}) Q_{\bar{\pi}^\star}(\bar{s}, \tilde{a}^\star) - Q_{\tilde{\pi}^\star_{\text{Q-MDP}}}(s, \tilde{a}^\star) \\
= \;& \sum_{\bar{s} \in \bar{S}} w(s, \bar{s}) \bar{R}(\bar{s}, \tilde{a}^\star) - R(s, \tilde{a}^\star) \\
& + \gamma \sum_{\bar{s} \in \bar{S}} w(s, \bar{s}) \sum_{\bar{s}' \in \bar{S}} \bar{P}(\bar{s}', \bar{s}, \tilde{a}^\star) V_{\bar{\pi}^\star}(\bar{s}') - \gamma \sum_{s' \in S} P(s', s, \tilde{a}^\star) V_{\tilde{\pi}^\star_{\text{Q-MDP}}}(s') \\
= \;& \sum_{\bar{s} \in \bar{S}} w(s, \bar{s}) \bar{R}(\bar{s}, \tilde{a}^\star) - R(s, \tilde{a}^\star) \\
& + \gamma \sum_{\bar{s} \in \bar{S}} w(s, \bar{s}) \sum_{\bar{s}' \in \bar{S}} \bar{P}(\bar{s}', \bar{s}, \tilde{a}^\star) V_{\bar{\pi}^\star}(\bar{s}') - \gamma \sum_{s' \in S} P(s', s, \tilde{a}^\star) V_{\bar{\pi}^\star_{\text{Q-MDP}}}(s') \\
& + \gamma \sum_{s' \in S} P(s', s, \tilde{a}^\star) V_{\bar{\pi}^\star_{\text{Q-MDP}}}(s') - \gamma \sum_{s' \in S} P(s', s, \tilde{a}^\star) V_{\tilde{\pi}^\star_{\text{Q-MDP}}}(s') \\
\leq \;& K_{V_{\tilde{\pi}^\star_{\text{Q-MDP}}}} + \gamma \sum_{s' \in S} P(s', s, \tilde{a}^\star)(V_{\bar{\pi}^\star_{\text{Q-MDP}}}(s') - V_{\tilde{\pi}^\star_{\text{Q-MDP}}}(s'))
\end{aligned}$$

□

For a state-action pair (s, a), we can expand the difference between $Q_{\hat{\pi}^\star_{\text{Q-MDP}}}(s,a)$ and $Q_{\bar{\pi}^\star_{\text{Q-MDP}}}(s,a)$:

$$\begin{aligned} & |Q_{\hat{\pi}^\star_{\text{Q-MDP}}}(s,a) - Q_{\bar{\pi}^\star_{\text{Q-MDP}}}(s,a)| \\ = \; & |R(s,a) - \sum_{\bar{s}\in\bar{S}} w(s,\bar{s})\bar{R}(\bar{s},a) \\ & + \gamma \sum_{s'} P(s',s,a) V_{\bar{\pi}^\star_{\text{Q-MDP}}}(s') - \gamma \sum_{\bar{s}\in\bar{S}} w(s,\bar{s}) \sum_{\bar{s}'\in\bar{S}} \bar{P}(\bar{s}',\bar{s},a) V_{\bar{\pi}^\star}(\bar{s}')| \end{aligned} \tag{B.8}$$

which helps us bound $K(Q_{\bar{\pi}^\star_{\text{Q-MDP}}})$:

Theorem B.4.

$$K(Q_{\bar{\pi}^\star_{\text{Q-MDP}}}) \leq K_R + \gamma((1-\mathcal{D})\delta_{\bar{A}^\star} + \delta_{V_{\bar{\pi}^\star}} \frac{K_P}{2}) \tag{B.9}$$

where $\mathcal{D} = \max_{s\in S} D(s)$.

Proof.

$$\begin{aligned} & K(Q_{\bar{\pi}^\star_{\text{Q-MDP}}}) \leq \max_{a\in A, s\in S} |R(s,a) - \sum_{\bar{s}\in\bar{S}} w(s,\bar{s})\bar{R}(\bar{s},a)| \\ & + \max_{a\in A, s\in S} |\gamma \sum_{s'} P(s',s,a) V_{\bar{\pi}^\star_{\text{Q-MDP}}}(s') - \gamma \sum_{\bar{s}\in\bar{S}} w(s,\bar{s}) \sum_{\bar{s}'\in\bar{S}} \bar{P}(\bar{s}',\bar{s},a) V_{\bar{\pi}^\star}(\bar{s}')| \end{aligned} \tag{B.10}$$

The difference between actual rewards and estimated rewards is obviously bounded by K_R. We consider the value difference between next states:

$$\begin{aligned} & |\sum_{s'} P(s',s,a) V_{\bar{\pi}^\star_{\text{Q-MDP}}}(s') - \sum_{\bar{s}\in\bar{S}} w(s,\bar{s}) \sum_{\bar{s}'\in\bar{S}} \bar{P}(\bar{s}',\bar{s},a) V_{\bar{\pi}^\star}(\bar{s}')| \\ = \; & |\sum_{s'} P(s',s,a) V_{\bar{\pi}^\star_{\text{Q-MDP}}}(s') - \sum_{s'} P(s',s,a) \sum_{\bar{s}'\in\bar{S}} w(s',\bar{s}') V_{\bar{\pi}^\star}(\bar{s}') \\ & + \sum_{s'} P(s',s,a) \sum_{\bar{s}\in\bar{S}} w(s',\bar{s}') V_{\bar{\pi}^\star}(\bar{s}') - \sum_{\bar{s}'\in\bar{S}} w(s,\bar{s}) \sum_{\bar{s}'\in\bar{S}} \bar{P}(\bar{s}',\bar{s},a) V_{\bar{\pi}^\star}(\bar{s}')| \\ \leq \; & (1-D(s))\delta_{\bar{A}^\star} + \delta_{V_{\bar{\pi}^\star}} \frac{K_P}{2} \end{aligned} \tag{B.11}$$

□

Appendix C

Lower Bound on the Quotient of Transition-to-Roles

Suppose $\mathbf{p}^{(1)}$, $\mathbf{p}^{(2)}$, and $\mathbf{b}$ are 3 positive vector of same length n. $\Omega[\mathbf{L}]$ is a n by n matrix of lower bounds on pairs of elements in $\mathbf{b}$:

$$\frac{\mathbf{b}_y}{\mathbf{b}_x} \geq \Omega[\mathbf{L}]_{x,y} \tag{C.1}$$

We want to find out a lower bound that is as large as possible for

$$\begin{aligned} &\frac{\mathbf{p}^{(1)\intercal}\mathbf{b}}{\mathbf{p}^{(2)\intercal}\mathbf{b}} \\ =&\frac{\sum_{i=1}^{n}\mathbf{p}_i^{(1)}\mathbf{b}_i}{\sum_{i=1}^{n}\mathbf{p}_i^{(2)}\mathbf{b}_i} \end{aligned} \tag{C.2}$$

We split each $\mathbf{b}_i$ in the numerator using non-negative variables $\beta_{i,j}$, such that $\mathbf{b}_i = \sum_{j=1}^{n}\beta_{i,j}\mathbf{b}_i$. In other words, $\sum_{j=1}^{n}\beta_{i,j} = 1$ for all i.

$$\begin{aligned} =&\frac{\sum_{i=1}^{n}\mathbf{p}_i^{(1)}\sum_{j=1}^{n}\beta_{i,j}\mathbf{b}_i}{\sum_{i=1}^{n}\mathbf{p}_i^{(2)}\mathbf{b}_i} \\ \geq&\frac{\sum_{i=1}^{n}\mathbf{p}_i^{(1)}\sum_{j=1}^{n}\beta_{i,j}\Omega[\mathbf{L}]_{j,i}\mathbf{b}_j}{\sum_{i=1}^{n}\mathbf{p}_i^{(2)}\mathbf{b}_i} \\ =&\frac{\sum_{j=1}^{n}\mathbf{b}_j\sum_{i=1}^{n}\beta_{i,j}\Omega[\mathbf{L}]_{j,i}\mathbf{p}_i^{(1)}}{\sum_{i=1}^{n}\mathbf{p}_i^{(2)}\mathbf{b}_i} \end{aligned} \tag{C.3}$$

We have the freedom to choose the values of β, so long they are non-negative and $\beta_{i,j}$ of same i sum up to 1, and this inequality always holds. The greatest lower bound we can derive is the largest quotient, which can

be found by solving the linear program of ω and β:

$$\begin{aligned}
\text{maximize:} \quad & \omega \\
\text{subject to:} \quad & \text{for all } i, \quad \sum_{j=1}^{n} \beta_{i,j} = 1 \\
& \text{for all } (i,j), \quad \beta_{i,j} \geq 0 \\
& \sum_{i=1}^{n} \beta_{i,1} \Omega[\mathbf{L}]_{1,i} \mathbf{p}_i^{(1)} = \omega \mathbf{p}_1^{(2)} \\
& \sum_{i=1}^{n} \beta_{i,2} \Omega[\mathbf{L}]_{2,i} \mathbf{p}_i^{(1)} = \omega \mathbf{p}_2^{(2)} \\
& \quad \vdots \\
& \sum_{i=1}^{n} \beta_{i,n} \Omega[\mathbf{L}]_{n,i} \mathbf{p}_i^{(1)} = \omega \mathbf{p}_n^{(2)}
\end{aligned} \tag{C.4}$$

The actual solution for this linear program isn't particularly interesting, but from here we can see that the solution ω can be expressed as linear combinations of elements of $\Omega[\mathbf{L}]$, with non-negative coefficients. This guarantees that $\Omega[\mathbf{L}]$ doesn't contribute negatively to ω.

Appendix D

Performance Evaluation

```
1 def test_performance(role_assignment, n_tests=10,
    open_loop_length=1):
2     result = 0
3     for i_test in range(n_tests):
4         result += test_performance_one_episode(role_assignment,
     open_loop_length)
5     result = result / n_tests
6     return result
7
8 def test_performance_one_episode(role_assignment,
    open_loop_length = 1):
9     env = gym.make('CartPole-v1')
10    state = env.reset()
11    total_reward = 0
12    for i_loop in range(int(max_episode_length/open_loop_length
    )):
13        weights = role_assignment.assigner.assign(state)
14        action_sequence = role_assignment.role_mdp.
    get_action_sequence(weights, open_loop_length)
15        for i_in_loop in range(open_loop_length):
16            action = action_sequence[i_in_loop]
17            next_state, reward, is_terminal, info = env.step(
    action)
18            total_reward += reward
19            if is_terminal:
20                break
21        if is_terminal:
22            break
23        else:
24            state = next_state
25    env.close()
26    return total_reward
```

Appendix E

List of Hyperparameters

`n_roles`	Number of roles.
`concentration`	The step size of each assignment update. How concentrated the likelihood to roles. See section 4.1.
`beta`	Coefficient applied to reward difference. Since the transition difference is a total variation distance (in a range of $[0, 1]$), we use `beta` to scale reward difference to a similar range. See section 4.1.
`beta_v`	Coefficient applied to estimated action-value difference (EAVD). Should be 0 if the role MDP is used also for mixed open-loop and closed-loop control. See subsection 5.1.3.
`flatten_parameter`	A small value to prevent over committing assignment weights to a single role. See section 4.3.
`role_update_count`	The roles' attributes are updated as running averages of a number of most recent updates. This parameter controls how many recent updates are used. See subsection 5.1.1.
`assignment_update_interval`	How frequent we update the assigner, between time steps.
`assignment_update_batch_size`	The size of samples we use to update the assigner.

`role_mdp_update_interval`	How frequent we update the role MDP, between time steps.
`role_mdp_update_batch_size`	The size of samples we use to update the role MDP.
`gamma`	Discount factor, used for calculating value and policy in the role MDP.
`max_episode_length`	How many time steps an episode can have at max.
`n_episode`	Number of episodes used for training.
The neural network	The architecture or the neural network and the optimizer used for training are also part of the hyperparameters.

Appendix F

Hyperparameters Used in Experiments

	CartPole	Catcher	PuddleWorld
n_roles	10	10	10
concentration	0.1	1	3
beta	1	2	0.02
beta_v	0.3	2	0.001
flatten_parameter	0.0000001	0.001	0.005
role_update_count	1000	1000	1000
exp_size	50000	500000	500000
assignment_update_interval	1	10	10
assignment_update_batch_size	1024	25600	1280
role_mdp_update_interval	1	100	50
role_mdp_update_batch_size	1024	25600	1280
gamma	0.99	0.99	0.99