Université de Montréal

Agent Abstraction in Multi-Agent Reinforcement Learning

par

Amin Memarian

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en Informatique

June 06, 2022

Université de Montréal

Faculté des arts et des sciences

Ce mémoire intitulé

Agent Abstraction in Multi-Agent Reinforcement Learning

présenté par

Amin Memarian

a été évalué par un jury composé des personnes suivantes :

Laurent Charlin
(président-rapporteur)

Irina Rish
(directeur de recherche)

Eugene Belilovsky
(membre du jury)

Résumé

Cette thèse est organisée en deux chapitres. Le premier chapitre sert d'introduction aux concepts et idées utilisés dans le deuxième chapitre (l'article).

Le premier chapitre est divisé en trois sections. Dans la première section, nous introduisons l'apprentissage par renforcement en tant que paradigme d'apprentissage automatique et montrons comment ses problèmes sont formalisés à l'aide de processus décisionnels de Markov. Nous formalisons les buts sous forme de rendements attendus et montrons comment les équations de Bellman utilisent la formulation récursive du rendement pour établir une relation entre les valeurs de deux états successifs sous la politique de l'agent. Après cela, nous soutenons que la résolution des équations d'optimalité de Bellman est insoluble et introduisons des algorithmes basés sur des valeurs tels que la programmation dynamique, les méthodes de Monte Carlo et les méthodes de différence temporelle qui se rapprochent de la solution optimale à l'aide de l'itération de politique généralisée. L'approximation de fonctions est ensuite proposée comme moyen de traiter les grands espaces d'états. Nous discutons également de la manière dont les méthodes basées sur les politiques optimisent directement la politique sans optimiser la fonction de valeur. Dans la deuxième section, nous introduisons les jeux de Markov comme une extension des processus décisionnels de Markov pour plusieurs agents. Nous couvrons les différents cadres formés par les différentes structures de récompense et donnons les dilemmes sociaux séquentiels comme exemple du cadre d'incitation mixte. En fin de compte, nous introduisons différentes structures d'information telles que l'apprentissage centralisé qui peuvent aider à faire face à la non-stationnarité induite par l'adversaire. Enfin, dans la troisième section, nous donnons un bref aperçu des types d'abstraction d'état et introduisons les métriques de bisimulation comme un concept inspiré de l'abstraction de non-pertinence du modèle qui mesure la similarité entre les états.

Dans le deuxième chapitre (l'article), nous approfondissons finalement l'abstraction d'agent en tant que métrique de bisimulation et dérivons un facteur de compression que nous pouvons appliquer à la diplomatie pour révéler l'agence supérieure sur les unités de joueur.

Mots clés: Apprentissage par renforcement multi-agents, abstraction d'état, abstraction d'agent

Abstract

This thesis is organized into two chapters. The first chapter serves as an introduction to the concepts and ideas used in the second chapter (the article).

The first chapter is divided into three sections. In the first section, we introduce *Reinforcement Learning* as a *Machine Learning* paradigm and show how its problems are formalized using *Markov Decision Processes*. We formalize *goals* as *expected returns* and show how the *Bellman equations* use the recursive formulation of return to establish a relation between the values of two successive *states* under the agent's *policy*. After that, we argue that solving the *Bellman optimality equations* is intractable and introduce *value-based* algorithms such as *Dynamic Programming*, *Monte Carlo* methods, and *Temporal Difference* methods that approximate the optimal solution using *Generalized Policy Iteration*. *Function approximation* is then proposed as a way of dealing with large state spaces. We also discuss how *policy-based* methods optimize the policy directly without optimizing the value function. In the second section, we introduce *Markov Games* as an extension of Markov Decision Processes for multiple agents. We cover the different settings formed by the different reward structures and give *Sequential Social Dilemmas* as an example of the *mixed-incentive* setting. In the end, we introduce different information structures such as *centralized learning* that can help deal with the *opponent-induced non-stationarity*. Finally, in the third section, we give a brief overview of *state abstraction types* and introduce *bisimulation metrics* as a concept inspired by *model-irrelevance abstraction* that measures the similarity between states.

In the second chapter (the article), we ultimately delve into *agent abstraction* as a bisimulation metric and derive a *compression factor* that we can apply to Diplomacy to reveal the higher agency over the player units.

Keywords: Multi Agent Reinforcement Learning, State Abstraction, Agent Abstraction

Contents

List of Figures

List of Abbreviations

AI	Artificial Intelligence
CAP	Credit Assignment Problem
DDPG	Deep Deterministic Policy Gradient
DP	Dynamic Programming
DQN	Deep Q-Network
GPI	Generalized Policy Improvement
MADDPG	Multi-Agent Deep Deterministic Policy Gradient
MARL	Multi-Agent Reinforcement Learning
MC	Monte Carlo
MDP	Markov Decision Process
MG	Markov Game

ML	Machine Learning
NE	Nash Equilibrium
POSG	Partially Observable Stochastic Game
RL	Reinforcement Learning
SD	Social Dilemma
SSD	Sequential Social Dilemma
TD	Temporal Difference

Acknowledgments

I would like to thank all my friends and collaborators who helped me throughout my academic journey. Special thanks to Maximilian, whom I worked with since the very beginning of my research and mentored me on many topics, and to Matthew for never hesitating to help when I was stuck.

I would like to thank Irina, to whom I am forever indebted, for her endless support.

Chapter 1

Introduction

This chapter aims to give readers a broad background so that the concepts and notions used in the article are easily understood. We first briefly introduce *Reinforcement Learning* (RL) in Section 1.1, and then explore the *Multi-Agent Reinforcement Learning* (MARL) setting in Section 1.2. Finally, in Section 1.3, we outline the state abstraction concepts relevant to the article. Reading the parts marked with * are encouraged, but not necessary.

1.1. Reinforcement Learning: Learning from Experience

Machine Learning (ML) is a subfield of *Artificial Intelligence* (AI) primarily concerned with using data to improve the performance on tasks at hand. There are usually two (prominent) paradigms of ML that come to mind:

- *Supervised Learning*: Data is labelled. Each sample in the training set is a pair consisting of a feature vector X and a label y. The goal is to find a mapping function from the inputs (feature vectors) to the outputs (labels) in a way that generalizes well to unseen samples.
- *Unsupervised Learning*: Data is not labelled. The goal is to find patterns and structures in a set of feature vectors $\{X\}$.

However, these two paradigms and their intermediary forms (such as *Semi-Supervised* and *Self-Supervised Learning*) do not completely cover the set of ML paradigms; Reinforcement Learning (RL) is another ML paradigm that studies the problem of an agent trying to achieve its goal inside an environment through interaction, and the methods that solve this problem. We use RL to refer to all three: the field of study, the problem facing the agent, and the solution methods [**38**]. It is worth noting that RL differs in three distinct ways from the other three paradigms:

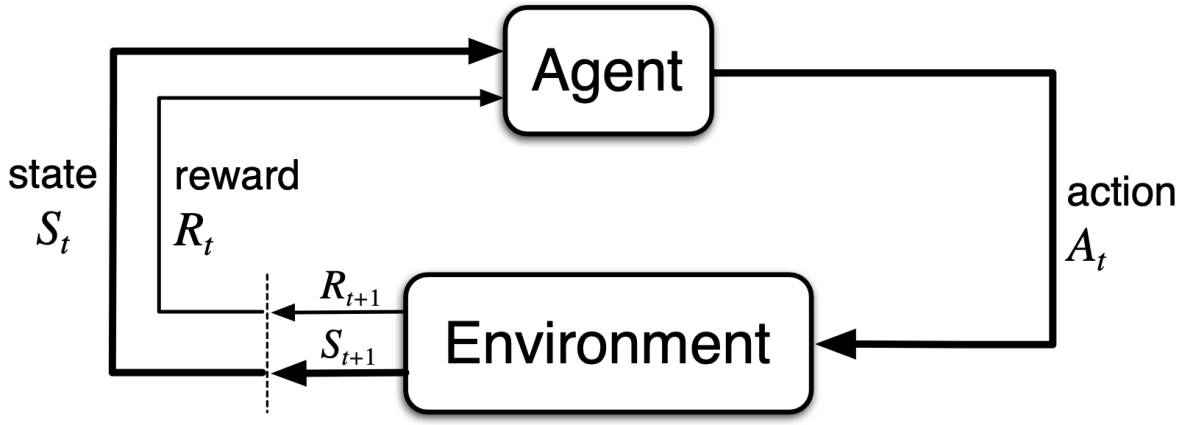


Fig. 1.1. Interaction of the agent with the environment in an MDP (taken from [**38**]); at the beginning the agent receives the start state, and at each time step after that, it takes an action and receives the next state and immediate reward from the environment.

- RL studies the problem of an agent trying to achieve its goal inside an environment through interaction in its entirety [**38**]. Assuming that "intelligence is the computational part of the ability to achieve goals in the world" (McCarthy, [**23, 36**]), other paradigms fall short of explicitly fitting into this definition.
- For an agent to achieve its goal, it must *exploit* the good experiences that it has gathered, and at the same time, it must *explore* the actions to gather better experiences. With only exploration or exploitation, failure is guaranteed. This is called the exploration-exploitation dilemma, and it is unique to RL [**38**].
- When the reward is temporally delayed, the agent must be able to assign a *credit* to the preceding actions that contributed to a successful outcome. This is called the *(tempral) credit assignment problem* (CAP) [**24**]. [1]

Throughout this section, we will use notations, figures, and equations from [**38, 5**], to define terms and convey ideas.

1.1.1. Formalizing RL Problems

A classic framework to formalize the sequential decision-making facing the agent is *Markov Decision Processes* (MDPs) [**31, 38**]. At each discrete *time step* $t \geq 0$, the agent gets the *state* of the environment S_t, takes an *action* A_t, and receives a numerical *reward* signal R_{t+1}, and the *next state* S_{t+1} from the environment. We will assume that MDPs are *finite* (an MDP in which $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{R}$ are finite), and that time steps are *discrete* unless explicitly mentioned otherwise. The sequence of the experienced states, actions, and rewards

[1] The credit assignment problem is also used in other domains, such as Deep Learning, to describe the problem of determining the contribution of certain components of the network to a specific outcome.

is called the *trajectory* τ, of the agent:

$$\tau_t \doteq (S_0, A_0, R_1, S_1, A_1, R_2, S_2, \cdots, S_t, A_t, R_t),$$

where $S_t \in \mathcal{S}$, $A_t \in \mathcal{A}(s)$, and $R_t \in \mathcal{R} \subset \mathbb{R}$. We define $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0,1]$, the *dynamics* of the MDP as:

$$p\left(s', r \mid s, a\right) \doteq \Pr\left\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\right\}, \tag{1.1.1}$$

such that:

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p\left(s', r \mid s, a\right) = 1, \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}(s). \tag{1.1.2}$$

p is a four argument function that could be used to compute all other functions of interest, such as the *state-transition probabilities*:

$$p\left(s' \mid s, a\right) \doteq \Pr\left\{S_t = s' \mid S_{t-1} = s, A_{t-1} = a\right\} = \sum_{r \in \mathcal{R}} p\left(s', r \mid s, a\right), \tag{1.1.3}$$

the *state-action reward*:

$$r(s, a) \doteq \mathbb{E}\left[R_t \mid S_{t-1} = s, A_{t-1} = a\right] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p\left(s', r \mid s, a\right), \tag{1.1.4}$$

and the *state-action-next-state reward*:

$$r\left(s, a, s'\right) \doteq \mathbb{E}\left[R_t \mid S_{t-1} = s, A_{t-1} = a, S_t = s'\right] = \sum_{r \in \mathcal{R}} r \frac{p\left(s', r \mid s, a\right)}{p\left(s' \mid s, a\right)}. \tag{1.1.5}$$

The states are said to have the *Markov property* if the current state includes all the information from past interactions that dictate the next state. In other words, states satisfy the Markovian assumption if the future is conditionally independent of the past, given the present:

$$\begin{aligned} &\Pr\left\{S_t = s', R_t = r \mid S_0, A_0, R_1, \ldots, R_{t-1}, S_{t-1} = s, A_{t-1} = a\right\} \\ &\quad = \Pr\left\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\right\} \\ &\quad = p\left(s', r \mid s, a\right). \end{aligned} \tag{1.1.6}$$

Given the description above, one can formally define an MDP as a 4-tuple $(\mathcal{S}, \mathcal{A}, p, r)$, where:

- $\mathcal{S}$ is the set of states or the *state space*. [2]
- $\mathcal{A}$ is the set of action or the a*ction space*. [3]
- $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ is the *state-transition function* $p\left(s' \mid s, a\right)$. [4]

[2] More precisely, $\mathcal{S}$, is the set of non-terminal states, while the set of all states (including terminal states) is denoted by $\mathcal{S}^+$; however, the distinction is often ignored, and they are used interchangeably.
[3] We will assume that $\mathcal{A} = \mathcal{A}(s)$ for any $s \in \mathcal{S}$.
[4] The state-transition function is also known as *(state) transition probability*, and is sometimes denoted as $p : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$, where Δ is a probability simplex.

- $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathcal{R}$ is the *state-action-next-state reward function* $r\,(s, a, s')$. [5]

If the MDP is *discounted*, it would also contain an element called the *discount factor*, denoted by $\gamma \in [0,1)$, and the MDP would be a 5-tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$. Furthermore, if the *start state* s_0 is not fixed, the MDP would also contain a *start state distribution* $\rho_0 \in \Delta(\mathcal{S})$, from which s_0 is sampled ($s_0 \sim \rho_0(\cdot)$) [**5**]; the MDP would then be 5-tuple (if the MDP is not discounted) or a 6-tuple (if the MDP is discounted), and would be denoted by $(\mathcal{S}, \mathcal{A}, p, r, \rho_0)$ or $(\mathcal{S}, \mathcal{A}, p, r, \gamma, \rho_0)$ respectively. The MDP abstraction successfully reduces most but not all decision-making problems of interest to three signals, namely: states, actions, and rewards [**38**].

1.1.1.1. Formalizing Goals in RL. MDPs attempt to formalize the problem of an agent trying to achieve its *goal* by interacting with its environment. However, there is no explicit notion of *goal* in the definition of an MDP. *The reward hypothesis* establishes the relation between the reward signal and the goal of an agent:

"That all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward)." (Sutton, [**37, 38**]).

Formally, the goal of the agent is to maximize its *expected (discounted) return*; We define the discounted return as:

$$G_t \doteq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k, \tag{1.1.7}$$

where T is the terminal time step. For *continuing tasks* where the interaction infinitely goes on, T is infinite, and for *episodic tasks*, in which *terminal states* exist, T is finite. $\gamma \in [0,1]$ is a discount factor that determines how myopic or far-sighted the agent is in cumulating its rewards. We should note that if $\gamma = 1$, the infinite sum in the continuing setting would be infinite; therefore, we restrict γ to values less than one to use a unified notation for both settings. The discounted return could be recursively defined as:

$$\begin{aligned} G_t &= R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots \right) \\ &= R_{t+1} + \gamma G_{t+1}. \end{aligned} \tag{1.1.8}$$

1.1.2. The RL Solutions

Assuming that an agent is following a *policy* π, the goal of the agent is finding a π that maximizes the expected *state-value* or *action-value function*. The policy π is simply mapping from states to actions. π can be *deterministic* or *stochastic*; in the deterministic case, it is a function that takes the state and outputs the action to be taken:

$$a = \pi(s), \,^{6}$$

[5] r can also be the *state-action reward function* $r(s,a)$.
[6] If the policy is deterministic, it is often denoted by μ.

and in the stochastic case, it takes the state and the action of interest and outputs the probability of taking that action in that state. Without passing an action, the policy outputs a probability distribution from which the action can be sampled:

$$A \sim \pi(\cdot \mid s).$$

We can formally define v_π[7], the state-value function for policy π, in any state $s \in \mathcal{S}$, as:

$$v_\pi(s) \doteq \mathbb{E}_\pi \left[G_t \mid S_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right], \tag{1.1.9}$$

and q_π [8], the action-value function for policy π, in any state $s \in \mathcal{S}$, taking any action $a \in \mathcal{A}$, as:

$$q_\pi(s,a) \doteq \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right]. \tag{1.1.10}$$

The relationship between v_π, q_π, π, and p (the MDP dynamics) is established by the following equations:

$$q_\pi(s,a) = \sum_{s',r} p\left(s', r \mid s, a\right) \left[r + \gamma v_\pi\left(s'\right) \right], \tag{1.1.11}$$

$$v_\pi(s) = \sum_{a} \pi(a) q_\pi(s,a). \tag{1.1.12}$$

The relationship between the value of a state s, and its next state s', under policy π, is established by the *Bellman equation for* v_π:

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi \left[G_t \mid S_t = s \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} \mid S_t = s \right] \\ &= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p\left(s', r \mid s, a\right) \left[r + \gamma \mathbb{E}_\pi \left[G_{t+1} \mid S_{t+1} = s' \right] \right] \\ &= \sum_a \pi(a \mid s) \sum_{s',r} p\left(s', r \mid s, a\right) \left[r + \gamma v_\pi\left(s'\right) \right], \qquad \text{for all } s \in \mathcal{S}. \end{aligned} \tag{1.1.13}$$

Equation 1.1.13, is better understood by following the backup diagram in Figure 1.2. The *Bellman equation for* q_π, is written as:

$$\begin{aligned} q_\pi(s,a) &\doteq \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a \right] \\ &= \sum_{s',r} p\left(s', r \mid s, a\right) \left[r + \gamma \sum_{a'} \pi\left(a' \mid s'\right) q_\pi\left(s', a'\right) \right], \end{aligned} \tag{1.1.14}$$

and its backup diagram is depicted in Figure 1.3.

[7] v_π denotes the state-value for a single state $s \in \mathcal{S}$, while $V_\pi : \mathcal{S} \to \mathbb{R}$ denotes the state-value of all states as a vector of size $|\mathcal{S}|$.

[8] q_π denotes the action-value for a single state-action pair $s \in \mathcal{S}, a \in \mathcal{A}$, while $Q_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ denotes the action-value of all state-action pairs as a vector of size $|\mathcal{S}| \times |\mathcal{A}|$.

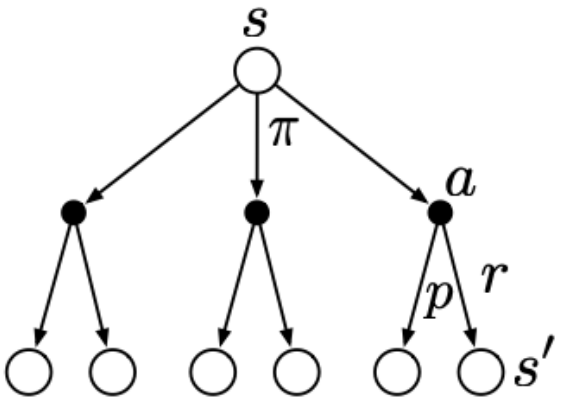


Fig. 1.2. The backup diagram for v_π (taken from [**38**]), where white circles are states, and black circles are state-action pairs. The Bellman equation calculates the state-value of s, by summing over all possible immediate rewards, plus discounted state-values of successor states, for all possible actions at state s, weighted by their probability. This diagram is called a backup diagram because it sends back information from the successor states to the current state.

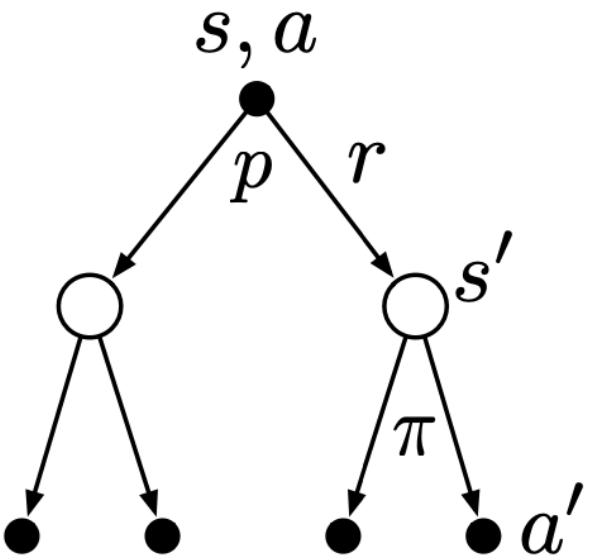


Fig. 1.3. The backup diagram for q_π (taken from [**38**]). The Bellman equation calculates the action-value of s,a, by summing over all immediate rewards, plus discounted action-values of successor state-action pairs, for all possible successor states, weighted by their probability.

1.1.2.1. v_* and q_*. The state-value function induces a non-strict partial ordering over the set of all policies Π, (a homogeneous relation with reflexive, symmetric, and transitive properties):

$$\pi \leq \pi' \Leftrightarrow v_\pi(s) \leq v_{\pi'}(s) \quad \text{for all } s \in \mathcal{S},$$

where $\pi, \pi' \in \Pi$ [**38, 34**]. We say that π' is *better* than π. The policy that is better than all other policies is called the *optimal policy* and is denoted by π_* [**38**]. The optimal policy always exists, but it may not be unique since the induced partial ordering is not strict [**38, 5**].

The *optimal value function* is defined as the value function of the optimal (maximizing) policy:

$$v_*(s) \doteq \max_{\pi \in \Pi} v_\pi(s), \quad \text{for all } s \in \mathcal{S}, \tag{1.1.15}$$

and the *optimal action-value function* is defined as the action-value function of the optimal (maximizing) policy:

$$q_*(s,a) \doteq \max_{\pi \in \Pi} q_\pi(s,a), \quad \text{for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A}. \tag{1.1.16}$$

The optimal policy π_*, could be written as:

$$\pi_* = \arg\max_{\pi \in \Pi} v_\pi(s), \quad \text{for all } s \in \mathcal{S}, \tag{1.1.17}$$

$$\pi_* = \arg\max_{\pi \in \Pi} q_\pi(s,a), \quad \text{for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A}. \tag{1.1.18}$$

The *Bellman optimality equation for* v_*, is therefore expressed as:

$$\begin{aligned} v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\ &= \max_a \mathbb{E}_{\pi_*}\left[G_t \mid S_t = s, A_t = a\right] \\ &= \max_a \mathbb{E}_{\pi_*}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a\right] \\ &= \max_a \mathbb{E}\left[R_{t+1} + \gamma v_*\left(S_{t+1}\right) \mid S_t = s, A_t = a\right] \\ &= \max_a \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma v_*\left(s'\right)\right], \end{aligned} \tag{1.1.19}$$

and the *Bellman optimality equation for* q_* is:

$$\begin{aligned} q_*(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*\left(S_{t+1}, a'\right) \mid S_t = s, A_t = a\right] \\ &= \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma \max_{a'} q_*\left(s', a'\right)\right]. \end{aligned} \tag{1.1.20}$$

The relationship between q_*, and v_* is established by:

$$q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma v_*\left(S_{t+1}\right) \mid S_t = s, A_t = a\right]. \tag{1.1.21}$$

The backup diagram for Equation 1.1.19, and Equation 1.1.20 is shown in Figure 1.4.

There is a unique solution to the Bellman optimality equation for v_* or q_* [**38, 7**]; however, finding the exact solution to a system of $|\mathcal{S}|$ (for each $s \in \mathcal{S}$), or $|\mathcal{S}| \times |\mathcal{A}|$ (for each $s \in \mathcal{S}, a \in \mathcal{A}$) non-linear equations is not practical since:

- States should satisfy the Markovian assumption.
- It requires full knowledge of the dynamics of the environment p.

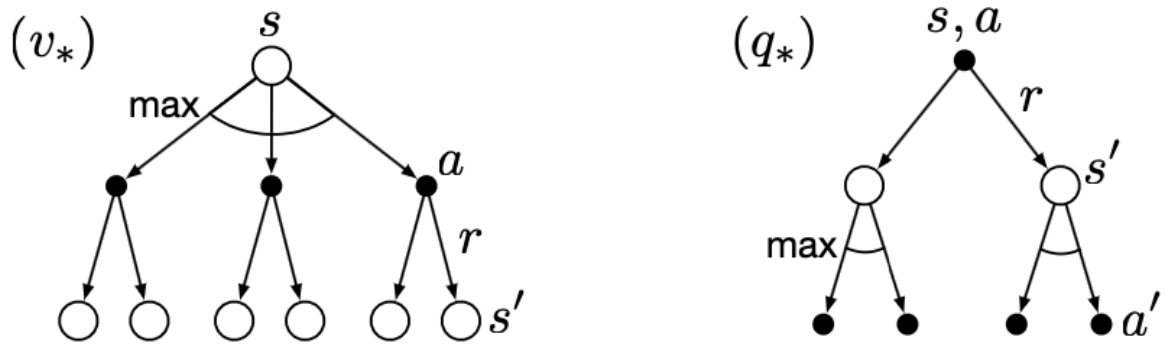


Fig. 1.4. The backup diagrams for v_*, and q_* (taken from [**38**]). The only difference between the Bellman optimality equations' backup diagrams and the Bellman equations' backup diagrams is the substitution of expectation over actions with the max over actions.

- It is computationally expensive, especially if the state space or the state-action space is huge.

In spite of that, deriving the optimal policy from either v_* or q_* is quite efficient. Any *greedy policy* with regard to v_* or q_* is optimal since it encompasses the expected discounted cumulative reward that could be sought from all possible successor states or state-action pairs. While deriving the optimal policy from v_*, requires the knowledge of transition probability (dynamics) for each action given the state, q_* does not require this knowledge at the expense of using more memory for storing the values of all state-action pairs [**38**].

Consequently, finding a good enough approximation of the v_* or q_* with low computational and/or memory cost would be ideal. We will cover these methods briefly in the next section.

1.1.3. An Overview of RL Algorithms

There are many ways to classify RL algorithms. We can categorize them using the following criteria: *tabular* versus *function approximation* methods, *model-free* versus *model-based* methods, or *value-based* versus *policy-based* methods. We will understand what these terms mean as we go through an inexhaustive list of different algorithms. This section is divided into four subsections, namely: *Dynamic Programming*, *Monte Carlo*, *Temporal Difference*, and *Deep Reinforcement Learning*. The first three subsections are basically a summarization of the corresponding sections in [**38**]; therefore, only sentences containing notable facts are cited to prevent overcitation.

1.1.3.1. *Dynamic Programming*. Dynamic Programming (DP) algorithms are algorithms that require a perfect model of the environment (it is model-based). DP algorithms work by turning the Bellman equations into *update rules* with which the optimal state-value function

could be approximated. To understand how DP works, we need to understand two distinct procedures: *policy evaluation* or *prediction*, and *policy improvement* or *control*. Computing the value of states under an arbitrary policy π is called policy evaluation. *Iterative policy evaluation*, starts from an arbitrary value function v_0, and uses the Bellman equation 1.1.13 to successively approximate the solution to a system of $|\mathcal{S}|$ linear equations in $|\mathcal{S}|$ unknowns (one for each $s \in \mathcal{S}$):

$$\begin{aligned} v_{k+1}(s) &\doteq \mathbb{E}_\pi \left[R_{t+1} + \gamma v_k \left(S_{t+1} \right) \mid S_t = s \right] \\ &= \sum_a \pi(a \mid s) \sum_{s',r} p \left(s', r \mid s, a \right) \left[r + \gamma v_k \left(s' \right) \right]; \end{aligned} \tag{1.1.22}$$

each update to the whole state space is called a *state sweep*. Using the converged state-values, we then can compute action-values by using Equation 1.1.11. Having the action-values allows us to use the *policy improvement theorem*, which states that for any pair of deterministic policies $\pi, \pi' \in \Pi$, if $q_\pi(s, \pi'(s)) \geq v_\pi(s)$ for all $s \in \mathcal{S}$, then $v_{\pi'}(s) \geq v_\pi(s)$ for all $s \in \mathcal{S}$. In other words, finding a policy π' which is at least as good as π, only requires greedily choosing the maximizing action of the action-value function under π:

$$\begin{aligned} \pi'(s) &\doteq \arg\max_a q_\pi(s, a) \\ &= \arg\max_a \mathbb{E} \left[R_{t+1} + \gamma v_\pi \left(S_{t+1} \right) \mid S_t = s, A_t = a \right] \\ &= \arg\max_a \sum_{s',r} p \left(s', r \mid s, a \right) \left[r + \gamma v_\pi \left(s' \right) \right]. \end{aligned} \tag{1.1.23}$$

From 1.1.23, we can derive the following equation:

$$\begin{aligned} v_{\pi'}(s) &= \max_a q_\pi(s, a) \\ &= \max_a \mathbb{E} \left[R_{t+1} + \gamma v_\pi \left(S_{t+1} \right) \mid S_t = s, A_t = a \right] \\ &= \max_a \sum_{s',r} p \left(s', r \mid s, a \right) \left[r + \gamma v_\pi \left(s' \right) \right]. \end{aligned} \tag{1.1.24}$$

The process of improving the policy described in Equation 1.1.23 is called policy improvement. Policies found in the process are either strictly better or equal to the current policy. In the latter case, the policy is optimal since replacing v_π with $v_{\pi'}$ in Equation 1.1.24, turns it into the Bellman optimality equation 1.1.19. The policy improvement theorem and its process also hold for stochastic policies [**38**].

Combining these two processes gives us an algorithm for finding the optimal policy, called *policy iteration*. Given an arbitrary policy π_0, we use policy evaluation to obtain the state-value function v_{π_0} and action-value function q_{π_0}, and use policy improvement to find a better policy π_1. Iteration is continued until convergence:

$$\pi_0 \overset{\text{Evaluate}}{\longrightarrow} v_{\pi_0} \overset{\text{Improve}}{\longrightarrow} \pi_1 \overset{\text{Evaluate}}{\longrightarrow} v_{\pi_1} \overset{\text{Improve}}{\longrightarrow} \pi_2 \overset{\text{Evaluate}}{\longrightarrow} \cdots \overset{\text{Improve}}{\longrightarrow} \pi_* \overset{\text{Evaluate}}{\longrightarrow} v_*.$$

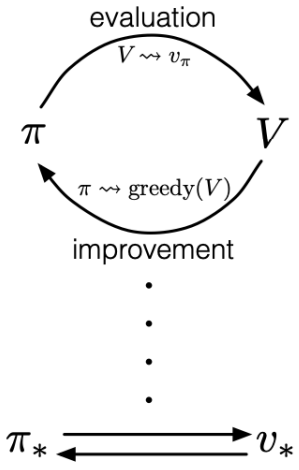


Fig. 1.5. Generalized Policy Improvement (taken from [**38**]); policy evaluation and policy improvement processes work together to attain convergence to the optimal state-value function, and optimal policy.

.

Value iteration optimizes the policy evaluation phase in the policy iteration while preserving the convergence guarantee [**38**]. It could also be seen as the update rule version of the Bellman optimality equation 1.1.19:

$$\begin{aligned} v_{k+1}(s) &\doteq \max_a \mathbb{E}\left[R_{t+1} + \gamma v_k\left(S_{t+1}\right) \mid S_t = s, A_t = a\right] \\ &= \max_a \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma v_k\left(s'\right)\right]. \end{aligned} \tag{1.1.25}$$

The policy iteration and value iteration algorithms *bootstrap*, meaning they use previous estimates to compute the next ones.

Generalized Policy Improvement or GPI [Figure 1.5] is the procedure used to describe most RL algorithms. It is generalized because it is not necessary for each process to complete before the other begins (as we have seen in value iteration). Independently, policy improvement makes the state values invalid, and policy evaluation makes the policy stale, but when these two processes work together and interact, convergence to the optimal value and policy is attained [**38**]. Convergence of the DP algorithms presented in this section, to the optimal value function and policy, is guaranteed [**38, 5**]. GPI algorithms are value-based methods, meaning the policy is derived from a value function.

DP methods introduced in this section have polynomial time complexity in terms of $|\mathcal{S}|$, and $|\mathcal{A}|$ (for a fixed γ); value iteration, which is more time-efficient than policy iteration, has a time complexity of $\mathcal{O}(|\mathcal{S}|^2 \times |\mathcal{A}|)$ [**38, 5**]. Although this is much more efficient than direct search methods, it is still very inefficient for large state spaces.

1.1.3.2. *Monte Carlo*. Monte Carlo (MC) methods do not require the complete knowledge of the dynamics of the environment p (they are model-free methods). Explicitly expressing p is not always possible, but it can be easily sampled. MC methods can learn from interacting with the sample episodes generated by p. They also do not bootstrap, and therefore they are less susceptible to issues caused by ignoring the Markovian assumption.

Monte Carlo prediction estimates $v_\pi(s)$ by averaging the returns that follow the visitation of any s in a set of sampled episodes. There are two variations of this algorithm: *first visit MC* and *every visit MC*. As their names suggest, first visit MC averages over returns followed by the first occurrence of the state in the episodes, while every visit MC averages over returns followed by every visit of the state in the episodes. While both methods converge to $v_\pi(s)$ as the number of visits goes to infinity, they have slightly different properties [**38**]. Monte Carlo methods are an unbiased estimate of $v_\pi(s)$ with a variance of $1/n$, in which n is the number of averaged returns [**38**]. The same procedure could be used to get estimates of the action-values; however, many state-action pairs will never be sampled if the policy is deterministic. One way to solve this is to change the start state; this is called *exploring starts*, and doing it for an infinite number of episodes ensures the visitation of all state-action pairs. A more practical alternative for a finite number of episodes is to use a stochastic policy with a non-zero probability for choosing different actions at each state. However, we are only covering the theoretical grounds for MC and not the practical version, which relaxes the assumption of infinite episode sampling and exploring starts.

Monte Carlo prediction interacts with *Monte Carlo control* in the way described by GPI to approximate the optimal value function and policy. In the policy improvement phase, a better policy π_{k+1} is estimated based on q_{π_k}; therefore, MC methods are value-based methods.

1.1.3.3. *Temporal Difference*. Temporal Difference (TD) methods are central to reinforcement learning. They could be seen as an intermediate class between MC methods and DP methods. Like DP, they bootstrap, and like MC, they use samples of the environment dynamics π.

TD prediction updates its estimate V of v_π, for S_t, by the following update rule:

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right], \tag{1.1.26}$$

where $\alpha \in [0,1]$, is the step-size. This is called *TD(0)* or *one-step TD* since the sample update uses a one-step look-ahead. TD(0) is a particular case of *n-step TD* [Figure 1.6]. *Constant-α MC* could be seen as an ∞-step TD [Figure 1.6], and its sample update could be written as:

$$V(S_t) \leftarrow V(S_t) + \alpha \left[G_t - V(S_t)\right]. \tag{1.1.27}$$

$R_{t+1}+\gamma V(S_{t+1})$ in 1.1.26, and G_t in 1.1.27 are called *TD target*, and *MC target* respectively. $\delta_t \doteq R_{t+1}+\gamma V(S_{t+1})-V(S_t)$ is called the *TD error*, thus we can rewrite 1.1.26 as:

$$V(S_t) \leftarrow V(S_t)+\alpha\delta_t. \tag{1.1.28}$$

We can also write the sample update rule for action-values as:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t)+\alpha\left[R_{t+1}+\gamma Q(S_{t+1},A_{t+1})-Q(S_t,A_t)\right], \tag{1.1.29}$$

where $R_{t+1}+\gamma Q(S_{t+1},A_{t+1})$ is the TD target. The TD error in this case is:

$$\delta_t \doteq R_{t+1}+\gamma Q(S_{t+1},A_{t+1})-Q(S_t,A_t),$$

and the sample update rule for action-values could be rewritten as:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t)+\alpha\delta_t. \tag{1.1.30}$$

Like MC, TD(0) does not require the model of the environment (they are model-free), but unlike it, it does not need to wait until the end of the episode to perform updates. This is a considerable advantage, especially when episodes are long or the task is continuing. While MC tries to estimate v_π by estimating $\mathbb{E}_\pi[G_t \mid S_t=s]$, TD(0) tries to do it by estimating $\mathbb{E}_\pi[R_{t+1}+\gamma v_\pi(S_{t+1}) \mid S_t=s]$ [**38**]. One-step TD's estimate is an estimate, because, as in MC, we use samples to estimate the true expectation ($\mathbb{E}_\pi$), and as in DP, we use estimates of the successor state-values ($v_\pi(S_{t+1})$) to perform the updates [**38**].

TD prediction works with *TD control* within the GPI algorithm to obtain the optimal value function and policy. Therefore, TD methods are value-based. We cover two cases of TD(0) control: *SARSA*, which is an *on-policy* control, and *Q-learning*, which is an *off-policy* one. SARSA is on-policy because the actions we take are the actions we use to update the action-values. Q-learning is off-policy because the action used to update the action-values is always the maximizing action, regardless of the actual action taken.

SARSA*. SARSA is an on-policy TD control algorithm that is named after the tuple of elements used in Equation 1.1.29, $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$. Deriving an on-policy control algorithm using the SARSA prediction (Equation 1.1.29) is fairly simple; we only have to make π more and more greedy with respect to q_π, until convergence to the optimal policy. SARSA's convergence to the optimal action-value function and policy is guaranteed if all state-action pairs are visited infinitely many times, and the learning policy becomes greedy in limit [**38, 35**].

Q-learning*. Q-learning is an off-policy TD control algorithm whose update rule is:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t)+\alpha\left[R_{t+1}+\gamma\max_a Q(S_{t+1},a)-Q(S_t,A_t)\right] \tag{1.1.31}$$

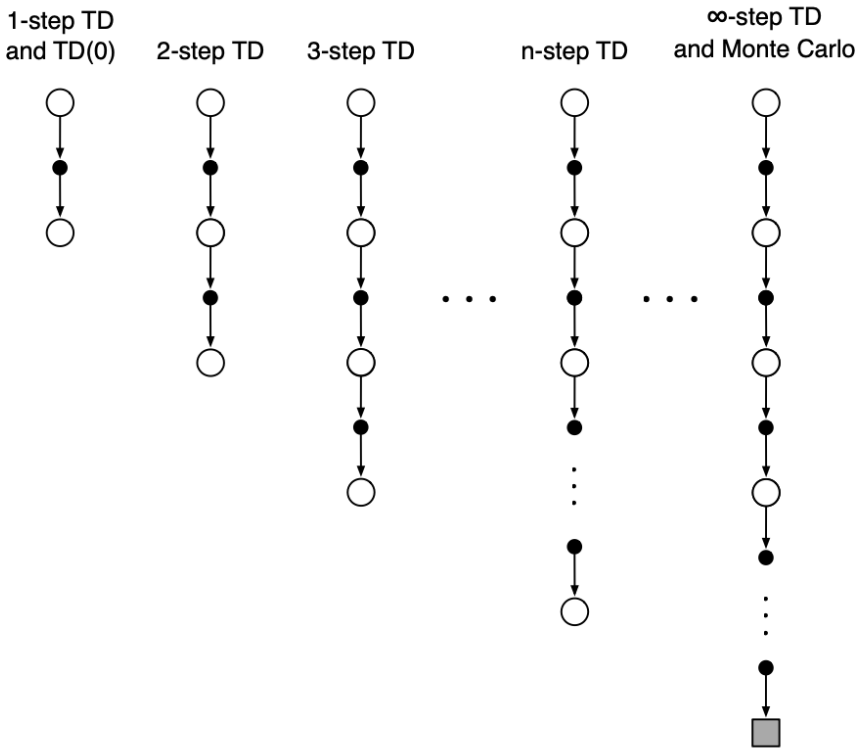


Fig. 1.6. N-step TD backup diagrams, Monte Carlo could be seen as a ∞-step TD method (taken from [**38**]).

In Q-learning, the *behaviour policy* does not affect the estimates but dictates which action-state pairs are visited. Therefore, the update rule approximates q_* directly. Q-learning is guaranteed to converge under certain conditions on the learning rates, if the values of all state-action pairs are continually updated [**38, 41**].

1.1.3.4. Deep Reinforcement Learning. All the methods discussed so far are tabular methods [Figure 1.7]–the policy and its corresponding action-value function are all stored in a table or array. However, as the size of the state-action space increases, memory, time, and data required for such algorithms become intractable. We can use function approximation and parameterize the action-value function or the policy to deal with this issue. This helps to generalize to states that the agent has not seen before.

So far, we have only explored *action-value methods* (also known as value-based methods); methods in which the policy is derived from the action-value function. However, there is another class of algorithms in which a parameterized policy is directly optimized, these algorithms are called *policy gradient methods* (also known as policy-based methods). If a parameterized value function is also learned along with the parameterized policy, the method is called *actor-critic*.

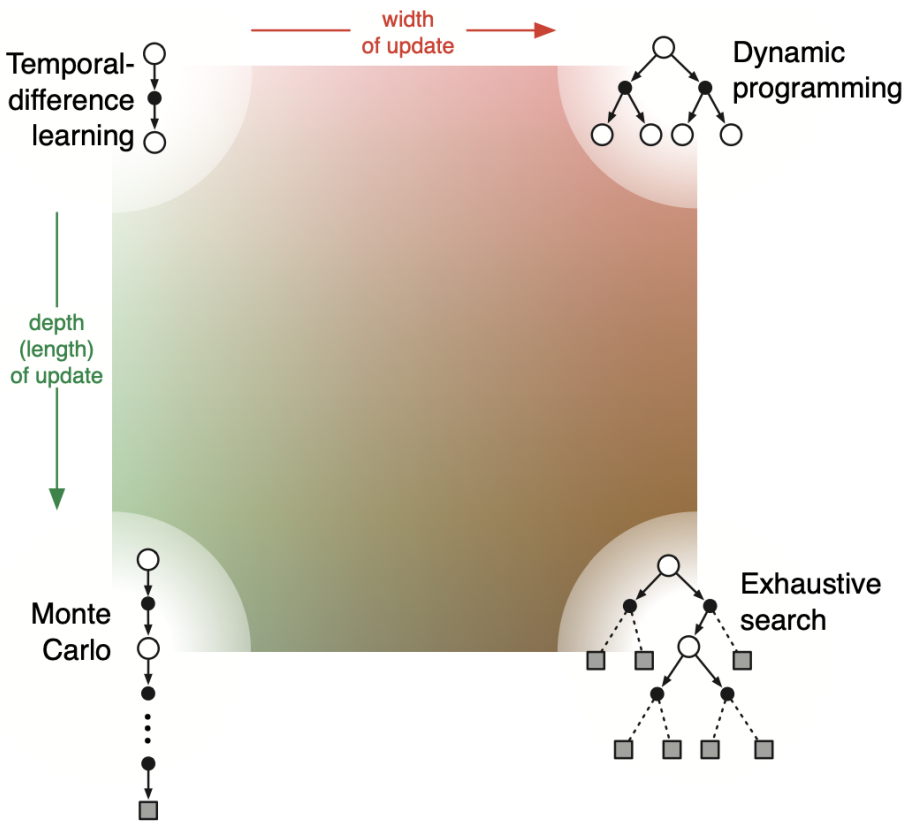


Fig. 1.7. Classification of the tabular methods based on the width and the length of their updates (taken from [**38**]).

.

Deep Q-Network*. We start off by introducing Deep Q-Network (DQN) [**25**], a model-free value-based off-policy method that learns a parameterized action-value function. DQN's objective could be written as:

$$\text{minimize}_{\phi}\, \mathbb{E}_{(S_t,A_t,R_{t+1},S_{t+1})\sim D}\left[\left(R_{t+1}+\gamma \max_{a\in\mathcal{A}} Q\left(S_{t+1},a;\phi^{-}\right)-Q\left(S_t,A_t;\phi\right)\right)^2\right]. \quad (1.1.32)$$

Function approximation combined with bootstrapping and off-policy training creates a situation prone to instability and divergence, called *the deadly triad* [**38**]. Therefore, two tricks are used in 1.1.32 to stabilize learning. The first trick is separating the *target network* (with parameters ϕ^{-}) from the *current network* (with parameters ϕ) [**25**]. The parameters of the target network are frozen and are updated periodically to match the parameters of the current network [**25, 32**]. The second trick is the use of the *experience replay* method, in which experiences are randomly sampled from a *replay memory* (D), and fed to the network [**25**]. This increases the data efficiency of the algorithm, since a single experience could be used several times [**25**]. It also reduces the variance of updates by breaking the strong correlation between consecutive experiences and makes learning smoother in case of

a change in the maximizing action [**25, 42**].

Deep Deterministic Policy Gradient*. Now we turn to a policy gradient method called Deep Deterministic Policy Gradient (DDPG). All policy gradient methods work by performing *gradient ascent* on the parameters of a policy. If we denote the parameterized policy by $\pi(a \mid s; \theta)$, then the update rule for θ is:

$$\theta_{t+1} = \theta_t + \alpha \widehat{\nabla J(\theta_t)} \tag{1.1.33}$$

where $\widehat{\nabla J(\theta_t)}$ is the estimate of the gradient of *performance measure function* $J(\theta)$ at time step t. Usually, $J(\theta)$ is the true value of the starting state under the parameterized policy π_θ:

$$J(\theta) \doteq v_{\pi_\theta}(s_0), \tag{1.1.34}$$

where s_0 is the starting state of the episode. We will only discuss the episodic setting, and we assume that s_0 is the same across different episodes.

Changing policy parameters changes both the distribution of states and the action selection. The policy gradient theorem associates the gradient of the performance measure function with the gradient of the parameterized policy while leaving out the *state distribution*:

$$\begin{aligned}
\nabla J(\theta) &\propto \sum_s d^{\pi_\theta}(s) \sum_a q_{\pi_\theta}(s,a) \nabla \pi(a \mid s; \theta) \\
&= \mathbb{E}_{S_t \sim d^{\pi_\theta}} \left[\sum_a q_{\pi_\theta}(S_t, a) \nabla \pi(a \mid S_t; \theta) \right] \\
&= \mathbb{E}_{S_t \sim d^{\pi_\theta}} \left[\sum_a \pi(a \mid S_t; \theta) q_{\pi_\theta}(S_t, a) \frac{\nabla \pi(a \mid S_t; \theta)}{\pi(a \mid S_t; \theta)} \right] \\
&= \mathbb{E}_{S_t \sim d^{\pi_\theta}} \mathbb{E}_{A_t \sim \pi_\theta(\cdot \mid S_t)} \left[q_{\pi_\theta}(S_t, A_t) \nabla \ln \pi(A_t \mid S_t; \theta) \right],
\end{aligned} \tag{1.1.35}$$

where d^{π_θ} is the on-policy state distribution under π_θ. With a conventional abuse of notation, we can write:

$$\nabla J(\theta) \propto \mathbb{E}_{\pi_\theta} \left[Q_{\pi_\theta}(S_t, A_t) \nabla \ln \pi(A_t \mid S_t, \theta) \right]. \tag{1.1.36}$$

This is very useful since we would have needed the analytical form of the state distribution otherwise, which is typically unknown [**38**].

Using the background above, we will now briefly cover DDPG [**17**], which is only applicable to continuous action spaces. DDPG is an actor-critic method, meaning that it learns both a value function with parameters ϕ, and a policy with parameters θ [**17**]. If the policy is deterministic, it is often denoted by μ instead of π. Like DQN, DDPG uses a replay memory D, and a separate target network with parameters ϕ^- [**17, 26**]; hence, the objective of the DDPG's action-value network could be written as:

$$\text{minimize}_\phi \, \mathbb{E}_{(S_t, A_t, R_{t+1}, S_{t+1}) \sim D} \left[\left(R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q\left(S_{t+1}, a; \phi^-\right) - Q\left(S_t, A_t; \phi\right) \right)^2 \right]. \tag{1.1.37}$$

However, since the action space is continuous, calculating the maximizing action in 1.1.37 is computationally intractable [**17, 26**]. To solve this issue, DDPG uses a separate parameterized policy μ_{θ^-}, which approximates the maximizing action of Q_{ϕ^-} [**18, 26**]. Therefore, the TD target in the objective of the action-value network (1.1.37) could be rewritten as:

$$R_{t+1} + \gamma Q\left(S_{t+1}, \mu_{\theta^-}(S_{t+1}); \phi^-\right). \tag{1.1.38}$$

The current parameterized policy μ_θ uses the deterministic policy gradient theorem (1.1.40) for continuous action spaces [**2, 17**], to maximize its parameters:

$$\text{maximize}_\theta \underset{S\sim\mathcal{D}}{\mathrm{E}}\left[Q_\phi\left(S, \mu_\theta(S)\right)\right], \tag{1.1.39}$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{S\sim\mathcal{D}}\left[\nabla_\theta \mu_\theta(S) \nabla_a Q_\phi(S, a = \mu_\theta(S))\right]. \tag{1.1.40}$$

Note that in 1.1.40, the expectation is with respect to $S \sim \mathcal{D}$ since we are using a replay memory whose state samples are generated by the state distribution under the parameterized deterministic policy d^{μ_θ}.

1.2. Multi-Agent Reinforcement Learning (MARL)

MDP is a framework that can formalize most problems of interest for single-agent RL [Figure 1.8(a)]. To formalize the MARL problem, we introduce an extension of MDPs called *Markov Games* (MGs) [Figure 1.8(b)], along with a less restrictive framework called Extensive-Form Games [Figure 1.8(c)]. This section is substantially a summarization of [**44**].

1.2.1. Markov Games

Markov Games (also known as Stochastic Games) [**44, 19, 11**], are defined by a 6-tuple $\left(\mathcal{N}, \mathcal{S}, \{\mathcal{A}^i\}_{i\in\mathcal{N}}, p, \{r^i\}_{i\in\mathcal{N}}, \gamma\right)$:

- $\mathcal{N} = \{1, \cdots, N\}$ is the set of agents in the environment ($N > 1$).
- $\mathcal{S}$ is the state space of all agents.
- $\mathcal{A} \doteq \mathcal{A}^1 \times \cdots \times \mathcal{A}^N$ where $\mathcal{A}^i$ is the action space of agent i ($1 \leq i \leq N$).
- $p : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition probability function (also denoted by T) from any $s \in \mathcal{S}$ using any $\boldsymbol{a} \in \mathcal{A}$ to any $s' \in \mathcal{S}$, where $\boldsymbol{a}$ is the *joint action* of all agents.
- $r^i : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ the reward function of the ith agent that returns a scalar value for any transition tuple (s,$\boldsymbol{a}$,s').
- $\gamma \in [0,1)$ is the discount factor.

At each time step, agent i uses its policy $\pi^i : \mathcal{S} \to \Delta\left(\mathcal{A}^i\right)$ to take action $A_t^i \sim \pi^i\left(\cdot \mid S_t\right)$, and to maximize its return by accumulating reward $r^i(S_t,A_t,S_{t+1})$. Therefore, we can define

the state value function of the ith agent $V^i : \mathcal{S} \to \mathbb{R}$ as:

$$V^i_{\pi^i,\pi^{-i}}(S) \doteq \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r^i\left(S_t, A_t, S_{t+1}\right) \mid A^i_t \sim \pi^i\left(\cdot \mid S_t\right), S_0 = s\right] \tag{1.2.1}$$

where $-i$ is the indices of agents in the set $\mathcal{N}$ other than i. V^i is a function of the joint policy $\boldsymbol{\pi} : \mathcal{S} \to \Delta(\mathcal{A})$, defined as:

$$\boldsymbol{\pi}(\boldsymbol{a} \mid s) \doteq \prod_{i \in \mathcal{N}} \pi^i\left(a^i \mid s\right). \tag{1.2.2}$$

Equation 1.2.1 tells us that the value of a state for agent i depends on the policies of all other agents as well as its own. This *opponent induced non-stationarity* is what makes MARL extremely challenging [**44**].

The solution to a Markov Game is a *Nash Equilibrium* (NE) $\boldsymbol{\pi}_* = \left(\pi^1_*, \cdots, \pi^N_*\right)$, such that:

$$V^i_{\pi^i_*,\pi^{-i}_*}(S) \geq V^i_{\pi^i,\pi^{-i}_*}(S) \quad \text{for any } i \in \mathcal{N}, \pi^i, \tag{1.2.3}$$

where π^i_* is called the *best-response* of π^{-i}_* [**44, 6, 11**]. If NE exists, MARL algorithms should ideally converge to it.

1.2.1.1. Reward Structure. Three settings can be defined based on the reward structure of the environment:

- The *fully cooperative* setting where agents collaborate to maximize a common return [**44**].
- The *fully competitive* setting where the sum of returns is zero [**44**].
- The *mixed-incentive* setting in which return is a general-sum, and (subsets of) agents can either cooperate or compete [**44, 13, 20**].
  - *Social Dilemma* (SD) is a subset of problems with mixed-incentive structure in which the interest of the individual is conflicted with the interest of the collective [**15**]. A famous social dilemma frequently studied in game theory is the *Prisoner's Dilemma* [**29, 15**], in which two players decide to defect (the individually optimal action), while cooperating yields the optimal collective payoff.
  - *Sequential Social Dilemmas* (SSDs) are temporally extended social dilemmas in which cooperativeness is a property of the policy of self-interested, independent agents (as opposed to their atomic action in social dilemmas) [**15**]. Agents are motivated to act greedily in the short term, but all of them will suffer if none cooperate in the long term [**15, 30**]. Diplomacy is a complex social dilemma game with mixed-incentive structure where agents should learn to cooperate with other agents to win [**28**].

In the fully cooperative setting, agents usually share their reward function:

$$r^1 = r^2 = \cdots = r^N = r,$$

this enables the use of single agent RL methods since the state-value function, and action-value function of all agents would be similar [**44, 40**]. In the fully competitive setting, the return of all agents sums up to zero:

$$\sum_{i \in \mathcal{N}} r^i\left(s, \boldsymbol{a}, s'\right) = 0 \quad \text{for any } \left(s, \boldsymbol{a}, s'\right).$$

In the mixed setting, there is no restriction on the sum of returns $\sum_{i \in \mathcal{N}} r^i\left(s, \boldsymbol{a}, s'\right)$, the fully cooperative setting and the fully competitive setting could be seen as special cases of the mixed setting.

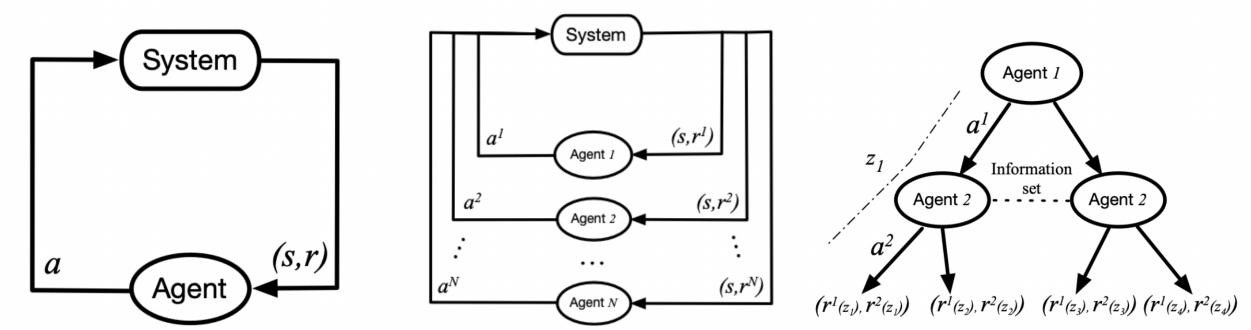
1.2.2. Extensive-Form Games*

Markov Games assume that agents have *perfect information* of the game at time step t; meaning that they have access to the *fully observable state* of the environment S_t, and the joint action A_t [**44**].[9] Extensive-Form Games allows for *imperfect information* or *partial observability* of the environment [Figure 1.8(c)] [**44, 27, 33**], and is defined by an 8-tuple $\left(\mathcal{N} \cup \{c\}, \mathcal{H}, \mathcal{Z}, \mathcal{A}, \{r^i\}_{i \in \mathcal{N}}, \tau, \pi^c, \mathcal{S}\right)$, where:

- $\mathcal{N} = \{1, \cdots, N\}$ is the set of agents in the environment ($N > 1$), and c is a special agent with a fixed stochastic policy representing the randomness of the world or the environment.
- $\mathcal{H}$ is the set of all possible histories (sequence of actions taken from the start).
- $\mathcal{Z} \subseteq \mathcal{H}$ is the subset of all terminal histories.
- $\mathcal{A}$ is the set of possible actions; at a given history h, it determines the set of actions that can be taken from there $\mathcal{A}(h) = \{a \mid ha \in \mathcal{H}\}$.
- $r^i : \mathcal{Z} \to \mathbb{R}$ assigns a scalar reward to each terminal history for agent i ($i \in \mathcal{N}$).
- $\tau : \mathcal{H} \to \mathcal{N} \cup \{c\}$ is the identification function that returns of the identity of the agent taking action at each history.
- π^c is policy of the chance agent from which is samples its action $a \sim \pi^c(\cdot \mid h)$.
- $\mathcal{S}$ is a partitioning of $\mathcal{H}$ to information states $s \in \mathcal{S}$, such that $\forall h, h' \in s$, $\tau(h) = \tau\left(h'\right) = \tau(s)$, and $\mathcal{A}(h) = \mathcal{A}\left(h'\right) = \mathcal{A}\left(s\right)$.

Extensive-Form Games can deal with imperfect information due to the fact that all histories in the same information state are indistinguishable to the agent [**44**]. Behavioural policy of agent i is defined as $\pi^i : \mathcal{S}^i \to \Delta(\mathcal{A}(s))$ where $\mathcal{S}^i = \{s \in \mathcal{S} : \tau(s) = i\}$. The joint policy is denoted as $\boldsymbol{\pi} = \left(\pi^1, \ldots, \pi^N\right)$, and the expected reward of agent i as $r^i(\boldsymbol{\pi}) =$

[9] Partially Observable Stochastic Games (POSGs) are a variation of Markov Games that can deal with partial observability. We briefly introduce POSGs in 1.2.3.2.



(a) Markov decision process (b) Markov game (c) Extensive-form game

Fig. 1.8. (a) MDP, describes the sequential decision-making problem facing a single agent. (b) Markov Game is used to formalize the decision-making problem for N agents; all agents receive the same state s and their reward r^i from the environment after taking a joint action $(a^1, a^2, \ldots, a^N)$. (c) Extensive-Form Games can formalize the problems with imperfect information, the two agents take turn taking their actions; since agent 2 is not aware of agent 1's action, the horizontal dashed line represents an information set. The diagonal dashed line is a terminal history z_1 at the end of which the agents are rewarded by $r^i(z_1)$. This figure is taken from [**44**].

.

$\sum_{z\in\mathcal{Z}} \eta_{\boldsymbol{\pi}}(z)\cdot r^i(z)$ where $\eta_{\boldsymbol{\pi}}(z)$ is the probability of reaching the terminal history z under the joint policy $\boldsymbol{\pi}$. In general, for any history h, $\eta_{\boldsymbol{\pi}}(h)$ is defined as:

$$\eta_{\boldsymbol{\pi}}(h) = \prod_{h':h'a\sqsubseteq h} \boldsymbol{\pi}^{\tau(h')}\left(a \mid I\left(h'\right)\right) = \prod_{i\in\mathcal{N}\cup\{c\}} \prod_{h':h'a\sqsubseteq h,\tau(h')=i} \pi^i\left(a \mid I\left(h'\right)\right), \tag{1.2.4}$$

where $h' : h'a \sqsubseteq h$ denotes all histories $h'a$ from which we can arrive at h following a sequence of actions. $I : \mathcal{H} \to \mathcal{S}$ is a function that maps histories to their corresponding information state:

$$I(h) = s \quad \text{if } h \in s.$$

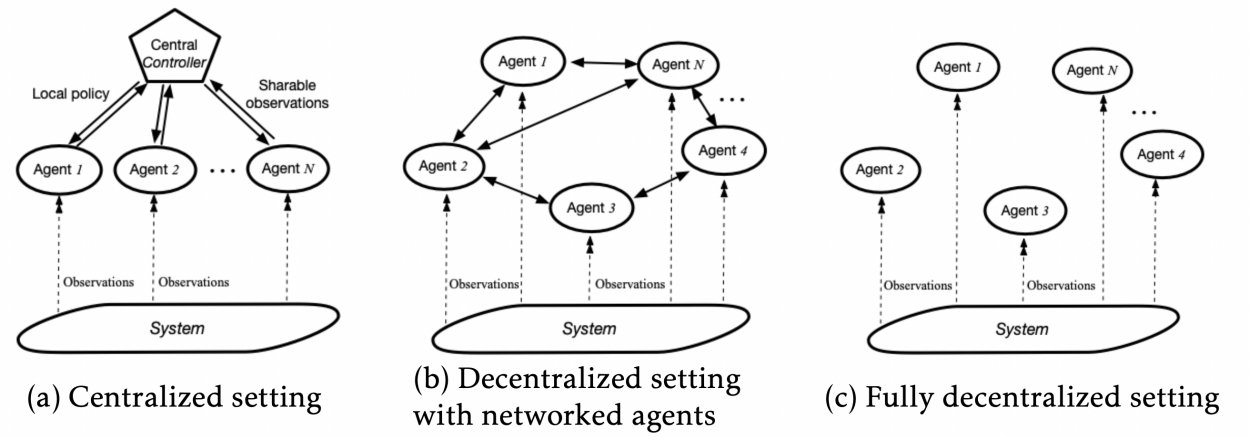
The ϵ-Nash Equilibrium for this formalism is then defined as $\boldsymbol{\pi}^* = \left(\pi_*^1, \cdots, \pi_*^N\right)$ such that:

$$r^i\left(\pi_*^i, \pi_*^{-i}\right) \geq r^i\left(\pi^i, \pi_*^{-i}\right) - \epsilon \quad \text{for any } i \in \mathcal{N}, \pi^i, \tag{1.2.5}$$

if $\epsilon = 0$, the equilibrium is a Nash Equilibrium [**44, 33**].

The perfect information game could be seen as a special case of imperfect information Extensive-Form Game where, for any $s \in \mathcal{S}$, $|s| = 1$ [**44**].

Since in Markov Games, the actions of other agents are not known, we can define all possible outcomes of that time step as an information state and formalize MGs as a special case of Extensive-Form Games where joint actions are the histories [**44**].



(a) Centralized setting

(b) Decentralized setting with networked agents

(c) Fully decentralized setting

Fig. 1.9. (a) The centralized setting, where a central controller gets the local observations of N agents and learns a local policy for each of them. (b) The decentralized setting in which agents can communicate certain local information with their neighbours. (c) The fully decentralized setting where there is neither a central controller nor a communication channel among agents, in some fully decentralized settings agents are allowed to receive some global information such as the joint action of other agents along with their local observations. This figure is taken from [**44**].

1.2.3. Information and Observability

We can describe different information sharing structures with three settings [Figure 1.9]:

- The *fully decentralized setting* where agents do not share any information. This is a realistic setting and is suited for many practical applications. However, the lack of access to policy, rewards, or observations of other agents exacerbates the non-stationary caused by their presence [**44**]. An extreme and special case of this setting is the independent learning scheme, where access to only the local observation and reward is granted [**44, 39**]. A less strict case of decentralized learning is the control sharing information structure, where the local observations contain some global information, such as the joint action of other agents [**44, 22**].
- The *decentralized setting with networked agents* allows the exchange of information over some communication channel between neighbours [**44**]. This setting is mostly used in cooperative tasks, and its convergence analysis is easier than the fully decentralized setting [**44, 14**].
- The *centralized setting* where a central controller has access to local information of all agents and can design their policies [**44, 12**]. A special and commonly used case of this setting is the *centralized-learning decentralized-execution* scheme, where observations and actions of all agents are sent to a central controller and the policies of the agents are designed based on the aggregated information [**44, 12, 21**]. This

alleviated the non-stationary due to partial observability [**21**]. At execution time, there is no controller and no communication channel between agents.

The following two methods illustrate two of these settings, namely: the centralized setting, and the centralized-learning decentralized-execution setting.

1.2.3.1. *Parameter Sharing.* tParameter sharing is a centralized training scheme which most effective in fully cooperative settings with *homogeneous agents*, (agents that share their reward function); however, it could also be used in other settings such as competitive settings with *self-play*. In this method, all agents use the same network for optimizing their value function and/or policy. Since all agents have access to the parameters of the network of other agents, this scheme is categorized as a centralized method. For example, the authors of [**28**] use self-play to train the agents on Diplomacy where one agent controls one power and the other control the remaining powers; the networks are initialized by training on human data with supervised learning.

1.2.3.2. *Centralized-Learning Decentralized-Execution: MADDPG*.* Multi-Agent Deep Deterministic Policy Gradient (MADDPG) is a centralized-training decentralized-execution method [Figure 1.10], that does not impose any restrictions on the communication between agents, and could be applied to cooperative, competitive or mixed settings [**21**].

MADDPG uses the Markov Games framework with partial observability (commonly known as Partially Observable Stochastic Games or POSGs), where we have an *observation space* $\mathcal{O}$, in addition to the state space $\mathcal{S}$. Each agent receives an observation correlated with the state using the function $\mathbf{o}^i : \mathcal{S} \mapsto \mathcal{O}^i$. The parameterized policy of agent i, is then denoted by $\mu_{\theta^i} : \mathcal{O}^i \mapsto \Delta(\mathcal{A}^i)$. We also assume that the reward function of agent i, $r^i : \mathcal{S} \times \mathcal{A}^i \mapsto \mathbb{R}$ is conditioned on its own action space and not the joint action space of other agents to make the scenario suited for decentralized execution. There are several challenges for extending DDPG to the multi-agent setting:

- Policy gradient methods usually have high variance, in the described setting (conditioning the reward on the local action), the variance is even higher due to the underlying dependence of the reward on the joint action of other agents [**21**].
  - We can resolve this issue by using a centralized critic for each agent i, denoted by $Q^i_\phi \equiv Q^i_{\phi^i,\mu_{\theta^i}} \equiv Q^i_{\phi^i,\mu^i}$ that takes the joint actions of all agents $\boldsymbol{a}_t = (a^1_t, \ldots, a^N_t)$ and some *state information* $\mathbf{X}_t$ at time step t. ϕ^i denotes the parameters of the centralized critic for agent i, and θ^i denotes the policy parameters of agent i. In the minimal case, $\mathbf{X}_t = \left(O^1_t, \ldots, O^N_t\right) \equiv \boldsymbol{O}_t$, but it usually contains some additional data [**21**].
- The replay memory D filled with transition/experience tuples of $(S_t, A_t, R_{t+1}, S_{t+1})$ is not useful, since change in policy of only one agent can change the transition probability from (S_t, A_t) to S_{t+1} [**21**].

– Since the knowledge of the joint actions of the agents makes the distribution stationary despite the changes in policies of agents $\left(p\left(s_{t+1} \mid s, a_1, \ldots, a_N, \pi_1, \ldots, \pi_N\right) = p\left(s_{t+1} \mid s, a_1, \ldots, a_N\right)\right)$, we can resolve this issue by filling the replay memory $\mathcal{D}$ with tuples of $\left(\mathbf{X}_t, \mathbf{X}_{t+1}, a_t^1, \ldots, a_t^N, R_{t+1}^1, \ldots, R_{t+1}^N\right) \equiv \left(\mathbf{X}_t, \mathbf{X}_{t+1}, \boldsymbol{a}_t, \boldsymbol{R}_{t+1}\right)$ [**21**].

Given the information above, we can write the gradient of deterministic policy $\mu_{\theta^i} \equiv \mu^i$ for agent i as:

$$\nabla_{\theta^i} J\left(\theta^i\right) = \mathbb{E}_{\mathbf{X}_t, \boldsymbol{a}_t^{-i} \sim \mathcal{D}}\left[\nabla_{\theta^i} \mu^i\left(O_t^i\right) \nabla_{a_t^i} Q_\phi^i\left(\mathbf{X}_t, \left(\boldsymbol{a}_t^{-i}, a_t^i = \mu^i\left(O_t^i\right)\right)\right)\right], \tag{1.2.6}$$

where $\boldsymbol{a}_t^{-1}$ is the joint action of all agents excluding agent i. The loss function of agent i's centralized critic is written as:

$$\mathcal{L}\left(\phi^i\right) = \mathop{\mathbb{E}}_{(\mathbf{X}_t, \boldsymbol{a}_t, \boldsymbol{R}_{t+1}, \mathbf{X}_{t+1}) \sim \mathcal{D}}\left[\left(\left(R_{t+1}^i + \gamma Q_{\phi^-}^i\left(\mathbf{X}_{t+1}, \boldsymbol{\mu}^-(\boldsymbol{O}_{t+1})\right)\right) - Q_\phi^i\left(\mathbf{X}_t, \boldsymbol{a}_t\right)\right)^2\right], \tag{1.2.7}$$

where $\boldsymbol{\mu}^- = (\mu_{\theta^{1,-}}, \mu_{\theta^{2,-}}, \ldots, \mu_{\theta^{N,-}})$ is the joint target policy with frozen parameters $\boldsymbol{\theta}^- = (\theta^{1,-}, \ldots, \theta^{N,-})$, that are periodically updated to match the current policy parameters $\boldsymbol{\theta} = (\theta^1, \ldots, \theta^N)$. In practice, agent i does not have access to the joint target policy; therefore, it should store an approximation of agent j's policy $(j \neq i)$, $\hat{\mu}_{\phi_j^i} = \hat{\mu}_j^i$ [**21**]. Once the agent has obtained the approximations, the TD target in the centralized critic's loss function (Equation 1.2.7) could be rewritten as:

$$R_{t+1}^i + \gamma Q_{\boldsymbol{\phi}^-}^i\left(\mathbf{X}_{t+1}, \hat{\boldsymbol{\mu}}^{i,-}(\boldsymbol{O}_{t+1})\right), \tag{1.2.8}$$

where $\hat{\boldsymbol{\mu}}^{i,-} \equiv \left(\hat{\boldsymbol{\mu}}_{-i}^{i,-}, \mu^{i,-}\right)$ is the approximate joint policy with frozen parameters for all agents excluding agent i, along with the true policy of agent i.

MADDPG uses other techniques such as *ensemble policies* to improve learning [**21**]. However, they are beyond the scope of our interest. One critique of MADDPG is that it only works well in the environments that are specifically designed for this algorithm, and fails to perform well in other arbitrary environments [**1**].

1.3. State Abstraction

In this section, we will briefly cover different types of *state abstraction* and their properties. To acquaint readers with the ideas we borrowed from in the paper to define *agent abstraction*, we then discuss a metric called *bisimulation metric*, which given a pair of states, measures how behaviourally similar they are. We use [**3, 16, 9, 10**] as our sources for this section.[10]

[10]In this section, with abuse of notation, we use V and Q instead of v and q to match the definitions in the state abstraction literature.

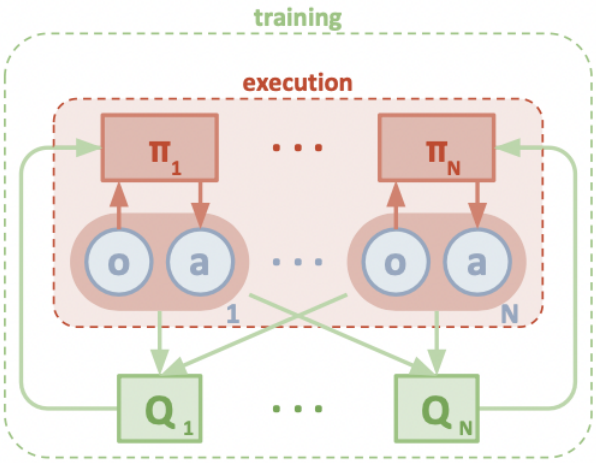


Fig. 1.10. The centralized-training decentralized-execution scheme used in MADDPG (taken from [**12**]). Each agent has a separate centralized critic that has access to the joint observation-action pairs of all agents. Each agent's actor uses the corresponding centralized critic to optimize its policy at training time. Agents do not have access to their centralized critic nor the joint observation-action pairs at execution time.

First, we need to define state abstraction: Let $M = (\mathcal{S}, \mathcal{A}, p, r, \gamma, \rho_0)$ be the ground truth MDP; state abstraction is a function $\phi : \mathcal{S} \to \mathcal{S}_\phi$ that maps every $s \in \mathcal{S}$ to a $s_\phi \in \mathcal{S}_\phi$. $\mathcal{S}_\phi$ is called the *abstract state space* and s_ϕ is an *abstract state* in that space [**3, 16**].

To define the *abstract MDP* M_ϕ, we also need to define the *abstract reward and transition functions*. In order to do so, we first introduce the *weighting function* w as an element of the abstract MDP, using which the abstract reward function and the abstract transition function are defined. $w : \mathcal{S} \to [0,1]$ is a function such that for each abstract state, the sum of w for all *ground truth states* that are mapped to that abstract state is equal to 1:

$$\forall_{s_\phi \in \mathcal{S}_\phi} : \sum_{s \in S_\phi} w(s) = 1 \quad \text{where } s \in s_\phi \equiv s \in \{\bar{s} \in \mathcal{S} : \phi(\bar{s}) = s_\phi\},$$

in other words, the weighting function measures the amount of contribution of s to $\phi(s)$ [**4, 16**]. The abstract reward function $r_\phi : \mathcal{S}_\phi \times \mathcal{A} \times \mathcal{S} \to \mathcal{R}$ is then defined as:

$$r_\phi\left(s_\phi, a, s'_\phi\right) = \sum_{s \in s_\phi} \sum_{s' \in s'_\phi} r\left(s, a, s'\right) w(s). \tag{1.3.1}$$

And the abstract transition function $p_\phi : \mathcal{S}_\phi \times \mathcal{A} \to \Delta\left(\mathcal{S}_\phi\right)$ as:

$$p_\phi\left(s'_\phi \mid s_\phi, a\right) = \sum_{s \in s_\phi} \sum_{s' \in s'_\phi} p\left(s' \mid s, a\right) w(s). \tag{1.3.2}$$

The abstract reward and transition functions are the weighted sums of rewards and transition probabilities of the ground MDP states that map to the same abstract state [**3**]. (M, ϕ, w)

induces an abstract MDP $M_\phi = \left(\mathcal{S}_\phi, \mathcal{A}, r_\phi, p_\phi, \gamma, \rho_0^\phi\right)$. The optimal policy for M_ϕ is:

$$\pi_{\phi,\star} = \arg\max_{\pi_\phi \in \Pi_\phi} \mathbb{E}_{s_0 \sim \rho_0} \left[v_{\pi_\phi} \left(\phi \left(s_0 \right) \right) \right]. \tag{1.3.3}$$

The goal of abstraction is to find a good policy in the abstract MDP M_ϕ that could perform well in the ground MDP M [**4, 16**]. We can formalize this objective as minimizing the *value loss*:

$$\min_{\pi_\phi \in \Pi_\phi} \mathbb{E}_{s_0 \sim \rho_0} \left[v_* \left(s_0 \right) - v_{\pi_\phi} \left(s_0 \right) \right]. \tag{1.3.4}$$

1.3.1. Abstraction Types

A *state abstraction type* is a set of functions $\Phi_{pred} \subseteq \Phi_{\text{all}}$, where $pred : \mathcal{S} \times \mathcal{S} \to \{0,1\}$ is a predicate on state pairs of a ground MDP M, such that:

$$\phi_{pred} \left(s_1 \right) = \phi_{pred} \left(s_2 \right) \Longrightarrow p \left(s_1, s_2 \right) \quad \text{for any } \phi_{pred} \in \Phi_{pred}, \tag{1.3.5}$$

in other words, the predicate must be true for any state pairs abstracted by ϕ_{pred} [**4**]. Authors of [**16**] discuss five types of abstraction:

- *Model-irrelevance abstraction*, also known as *bisimulation*, denoted by ϕ_{model} is an abstraction in which two states are aggregated if the rewards obtained from those states for all actions, and the transition probabilities from those states for all actions *and* all abstract states are equal:

$$\begin{aligned} \phi_{\text{model}} \left(s_1 \right) = \phi_{\text{model}} \left(s_2 \right) \Longrightarrow r \left(s_1, a \right) = r \left(s_2, a \right) \\ \& \\ \sum_{s' \in s_\phi} p \left(s' \mid s_1, a \right) = \sum_{s' \in s_\phi} p \left(s' \mid s_2, a \right), \end{aligned} \tag{1.3.6}$$

  for all $a \in \mathcal{A}, s_\phi \in \mathcal{S}$.

- *Q_π-irrelevance abstraction*, denoted by ϕ_{Q_π} is an abstraction in which two states are aggregated if their action-value functions for those states and all actions are equal under any policy:

$$\phi_{Q_\pi} \left(s_1 \right) = \phi_{Q_\pi} \left(s_2 \right) \Longrightarrow Q_\pi \left(s_1, a \right) = Q_\pi \left(s_2, a \right) \quad \text{for any } a \in \mathcal{A}, \pi \in \Pi.$$

- *Q_*-irrelevance abstraction*, denoted by ϕ_{Q_*} is an abstraction in which two states are aggregated if their action-value functions for those states and all actions are equal under the optimal policy:

$$\phi_{Q_*} \left(s_1 \right) = \phi_{Q_*} \left(s_2 \right) \Longrightarrow Q_* \left(s_1, a \right) = Q_* \left(s_2, a \right) \quad \text{for any } a \in \mathcal{A}.$$

- *a_*-irrelevance abstraction*, denoted by ϕ_{a_*} is an abstraction in which two states are aggregated if the action that maximizes the optimal action-value function of those states is the same *and* the optimal action-value of those states given the maximizing

action is the same:

$$\begin{aligned}\phi_{a_*}(s_1) = \phi_{a_*}(s_2) \Longrightarrow Q_*(s_1, a_*) &= \max_a Q_*(s_1, a) \\ &= \max_a Q_*(s_2, a) = Q_*(s_2, a_*).\end{aligned} \tag{1.3.7}$$

- π_*-*irrelevance abstraction*, denoted by ϕ_{π_*} is an abstraction in which two states are aggregated if the action that the optimal policy chooses in those states is the same:

$$\phi_{\pi_*}(s_1) = \phi_{\pi_*}(s_2) \Longrightarrow \pi_*(s_1) = \pi_*(s_2). \tag{1.3.8}$$

All of the mentioned abstraction types are *exact abstractions*, since the operation that satisfies the predicate is equality.

Authors of [**16**], also introduce two important theorems regarding the exact abstraction types mentioned above:

- For any MDP M, $\Phi_0 \succeq \Phi_{\text{model}} \succeq \Phi_{Q_\pi} \succeq \Phi_{Q_*} \succeq \Phi_{a_*} \succeq \Phi_{\pi_*}$, where Φ_0 denotes no abstraction, and $\Phi_i \succeq \Phi_j$ if:

$$\forall \phi_i \in \Phi_i : \phi_i \in \Phi_i \Longrightarrow \phi_i \in \Phi_j, \tag{1.3.9}$$

  instances of Φ_i are said to be *finer* than that of Φ_j. This simply means that any pair of states abstracted by instances of Φ_i must be also abstracted by instances of Φ_j [**4**].
- For Φ_{model} , Φ_{Q_π}, Φ_{Q_*}, and Φ_{a_*} the optimal policy in the abstract MDP, $\pi_{\phi,\star}$ is also optimal in the ground MDP. However, Φ_{π_*} does not guarantee such optimality.

1.3.1.1. *Bisimulation Metrics.* Model-irrelevance abstraction or bisimulation, abstracts states with similar reward and transition probabilities. Bisimulation metric is a concept inspired by bisimulation that measures how behaviourally similar states are [**24, 9, 10, 8**]. The abstraction function in bisimulation (1.3.6), is essentially an *equivalence relation* on the state space of the ground MDP $M = (\mathcal{S}, \mathcal{A}, p, r, \gamma)$, which partitions it into groups of equivalent states [**9, 10**]. However, this exact partitioning, is of little practical use; therefore, it is relaxed in bisimulation metrics by introducing a *pseudo-metric* ($\mathcal{S}$,d) in which $\mathcal{S}$ is the state space of the ground MDP, and $d : \mathcal{S} \times \mathcal{S} \mapsto \mathbb{R}_{\geq 0}$ is a *distance function* between two states [**24, 9, 10**]. One choice of distance function for transition probability functions is the k^{th} *Wasserstein distance*, defined as:

$$W_p(p_i, p_j; d) = \left(\inf_{\gamma' \in \Gamma(p_i, p_j)} \int_{\mathcal{S} \times \mathcal{S}} d(s_i, s_j)^p \ \mathrm{d}\gamma'(s_i, s_j) \right)^{1/k}, \tag{1.3.10}$$

in which $\Gamma(p_i, p_j)$ denotes all *couplings* of p_i and p_j. A famous lemma in [**9**], states that if d is a bisimulation metric, then:

$$d(s_i, s_j) = 0 \Leftrightarrow \forall a \in \mathcal{A}. \Big(r(s_i,a) = r(s_j,a) \text{ and } W_1\big(p(\cdot \mid s_i,a), p(\cdot \mid s_j,a); d\big) = 0 \Big). \tag{1.3.11}$$

for any s_i,$s_j \in \mathcal{S}$. Furthermore, a desirable bisimulation metric should preserve the optimal value of abstract states, for example:

$$|V_*(s_i) - V_*(s_j)| \leq \max_{a \in \mathcal{A}} \left(|r(s_i,a) - r(s_j,a)| + \gamma \left| \sum_{s' \in S} \left(p(s' \mid s_i,a) - p(s' \mid s_i,a) \right) V_*(s') \right| \right),$$

inspired the example above, and using the Wasserstein distance, bisumilation metric is then defined as:

$$d(s_i, s_j) = \max_{a \in \mathcal{A}} (1-c) \cdot |r(s_i,a) - r(s_j,a)| + c \cdot W_1\Big(p(\cdot \mid s_i,a), p(\cdot \mid s_j,a); d\Big), \tag{1.3.12}$$

where $c \in [0,1)$ is a constant [**24, 9**]. A natural choice for c in discounted MDPs is the discount factor γ [**9**]. c simply determines the relative importance of reward distance in comparison to the transition probability distance [**24**]. Other formulations of the definition allow for two separate positive constants c_R and c_P where $c_R + c_P \leq 1$; however, for introductory purposes we stick with this formulation where they sum up to 1.

An important theorem in [**9**], shows that the difference between the values of states in the abstract MDP M_ϕ and ground MDP M is bounded:

$$|V_*(s) - V_*(\phi(s))| \leq \frac{2\epsilon}{(1-\gamma)(1-c)} \quad , \tag{1.3.13}$$

where ϵ is the neighbourhood of aggregation by the bisimulation mapping ϕ. This is an important result, since the goal of abstraction is finding policies in the abstract MDP that can perform well in the ground MDP.

References

[1] Algorithms — Ray 1.12.1.

[2] Deterministic policy gradient algorithms | Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32.

[3] David Abel. A Theory of Abstraction in Reinforcement Learning. March 2022.

[4] David Abel. A theory of abstraction in reinforcement learning, 2022.

[5] Alekh Agarwal, Nan Jiang, Sham M Kakade, and Wen Sun. Reinforcement Learning: Theory and Algorithms. page 205.

[6] T. Basar and G. J. Olsder. *Dynamic noncooperative game theory: second edition.* Classics in Applied Mathematics. SIAM, 1999.

[7] Richard Bellman. *Dynamic Programming.* Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

[8] Pablo Samuel Castro. Scalable methods for computing state similarity in deterministic markov decision processes. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(06):10069–10076, Apr 2020.

[9] Norm Ferns, Prakash Panangaden, and Doina Precup. Bisimulation metrics for continuous markov decision processes. *SIAM J. Comput.*, 40(6):1662–1714, dec 2011.

[10] Norman Ferns, Prakash Panangaden, and Doina Precup. Metrics for finite markov decision processes, 2012.

[11] J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer, 1997.

[12] Eric A. Hansen, Daniel S. Bernstein, and Shlomo Zilberstein. Dynamic programming for partially observable stochastic games. In *Proceedings of the 19th National Conference on Artifical Intelligence*, AAAI'04, page 709–715. AAAI Press, 2004.

[13] Junling Hu and Michael P. Wellman. Nash q-learning for general-sum stochastic games. *J. Mach. Learn. Res.*, 4(null):1039–1069, dec 2003.

[14] Soummya Kar, José M. F. Moura, and H. Vincent Poor. QD-learning: A collaborative distributed strategy for multi-agent reinforcement learning through consensus + innovations. *IEEE Transactions on Signal Processing*, 61(7):1848–1862, Apr 2013.

[15] Joel Z. Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent Reinforcement Learning in Sequential Social Dilemmas. February 2017.

[16] Lihong Li, Thomas J. Walsh, and Michael L. Littman. Towards a unified theory of state abstraction for mdps. In *In Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*, pages 531–539, 2006.

[17] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2015.

[18] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. Technical Report arXiv:1509.02971, arXiv, July 2019. arXiv:1509.02971 [cs, stat] type: article.

[19] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on International Conference on Machine Learning*, ICML'94, 1994.

[20] Michael L. Littman. Friend-or-foe q-learning in general-sum games. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, page 322–328, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[21] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6382–6393, 2017.

[22] Aditya Mahajan. Optimal decentralized control of coupled subsystems with control sharing. *IEEE Conference on Decision and Control and European Control Conference*, Dec 2011.

[23] John McCarthy. WHAT IS ARTIFICIAL INTELLIGENCE?

[24] Marvin Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30, 1961.

[25] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. Technical Report arXiv:1312.5602, arXiv, December 2013. arXiv:1312.5602 [cs] type: article.

[26] OpenAI. Deep Deterministic Policy Gradient — Spinning Up documentation.

[27] Martin J. Osborne and Ariel Rubinstein. *A course in game theory*. The MIT Press, Cambridge, USA, 1994. electronic edition.

[28] Philip Paquette, Yuchen Lu, Seton Steven Bocco, Max Smith, Satya O-G, Jonathan K Kummerfeld, Joelle Pineau, Satinder Singh, and Aaron C Courville. No-press diplomacy: Modeling multi-agent gameplay. *Advances in Neural Information Processing Systems*, 32, 2019.

[29] William Poundstone. *Prisoner's dilemma*. Doubleday, New York, 1992. OCLC: 23383657.

[30] Victor Purvis. Self-interest and the Common Good. *Br Med J*, 1(5697):692–692, March 1970. Publisher: British Medical Journal Publishing Group Section: Correspondence.

[31] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., USA, 1st edition, 1994.

[32] Daniel Seita. Understanding Prioritized Experience Replay.

[33] Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, USA, 2008.

[34] Dan A. Simovici and Chaabane Djeraba. *Mathematical Tools for Data Mining: Set Theory, Partial Orders, Combinatorics*. Springer Science & Business Media, August 2008.

[35] Satinder Singh, Tommi Jaakkola, Michael L. Littman, and Csaba Szepesvári. Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms. *Machine Learning*, 38(3):287–308, March 2000.

[36] Richard S. Sutton. The Definition of Intelligence.

[37] Richard S. Sutton. The reward hypothesis.

[38] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction*. Adaptive computation and machine learning series. The MIT Press, Cambridge, Massachusetts, second edition edition, 2018.

[39] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *In Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337. Morgan Kaufmann, 1993.

[40] J. K. Terry, Nathaniel Grammel, Sanghyun Son, and Benjamin Black. Parameter Sharing For Heterogeneous Agents in Multi-Agent Reinforcement Learning. Technical Report arXiv:2005.13625, arXiv, January 2022. arXiv:2005.13625 [cs, stat] type: article.

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

[42] Lilian Weng. A (Long) Peek into Reinforcement Learning, February 2018. Section: posts.

[43] Amy Zhang, Rowan McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction, 2020.

[44] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms. *arXiv:1911.10635 [cs, stat]*, April 2021. arXiv: 1911.10635.

First Article.

Summarizing Societies: Agent Abstraction in Multi-Agent Reinforcement Learning

by

Amin Memarian[1], Maximilian Puelma Touzel[2],
Matthew Riemer[3], Rupali Bhati[4], and Irina Rish[5]

([1]) Université de Montréal, Mila
([2]) Université de Montréal, Mila
([3]) Université de Montréal, IBM Research, Mila
([4]) Université Laval, Mila
([5]) Université de Montréal, Mila

This article was submitted in From Cells to Societies: Collective Learning across Scales ICLR 2022 Workshop; the text is slightly modified to accommodate the committee members' request for clarification.

The main contributions of Amin Memarian for this articles are presented.

I contributed to the ideation and conceptualization of the project along with Maximilian and Matthew. Most of the writing was done by Maximilian, while I created the codebase for conducting the experiments, and Matthew came up with the bisimulation formulations.

Rupali helped me with processing the dataset, and Irina was the PI. The authors were ordered based on the amount of time and effort put into the project.

RÉSUMÉ. Les agents ne peuvent pas donner un sens aux sociétés à plusieurs agents en tenant compte directement des identités d'agents à petite échelle et de bas niveau, mais doivent plutôt reconnaître les identités collectives émergentes. Ici, nous faisons un premier pas vers un cadre pour reconnaître cette structure dans de grands groupes d'agents de bas niveau afin qu'ils puissent être modélisés comme un nombre beaucoup plus petit d'agents de haut niveau - un processus que nous appelons l'abstraction d'agent. Nous illustrons ce processus en étendant les métriques de bisimulation pour l'abstraction d'état dans l'apprentissage par renforcement au contexte de l'apprentissage par renforcement multi-agents et analysons une abstraction simple, bien que grossière, basée sur des actions conjointes expérimentées. Il traite la non-stationnarité due à d'autres agents d'apprentissage en améliorant le regret minimax par un facteur intuitif. Pour tester si ce facteur de compression fournit un signal pour une agence de niveau supérieur, nous l'avons appliqué à un grand ensemble de données de jeu humain du jeu de dilemme social populaire Diplomacy. Nous constatons qu'il est fortement corrélé avec le degré d'abstraction de la vérité au sol des unités de bas niveau dans les joueurs humains.
Mots clés : Abstraction d'état, Abstraction d'action, Abstraction d'agent, Apprentissage par renforcement multi-agents

ABSTRACT. Agents cannot make sense of many-agent societies through direct consideration of small-scale, low-level agent identities, but instead must recognize emergent collective identities. Here, we take a first step towards a framework for recognizing this structure in large groups of low-level agents so that they can be modeled as a much smaller number of high-level agents—a process that we call agent abstraction. We illustrate this process by extending bisimulation metrics for state abstraction in reinforcement learning to the setting of multi-agent reinforcement learning and analyze a straightforward, if crude, abstraction based on experienced joint actions. It addresses non-stationarity due to other learning agents by improving minimax regret by an intuitive factor. To test if this compression factor provides a signal for higher-level agency, we applied it to a large dataset of human play of the popular social dilemma game Diplomacy. We find that it correlates strongly with the degree of ground-truth abstraction of low-level units into the human players.
Keywords: State Abstraction, Action Abstraction, Agent Abstraction, Multi Agent Reinforcement Learning

1. Introduction

Much of the complexity in life arises from the way that individuals organize into collective behaviours. This becomes evermore the case when we acknowledge that what we often think of as *individuals* are really abstract entities comprised of many smaller entities that can be separately viewed as agents [**9**]. When tackling this complexity, it often becomes useful to exploit the coherence in that behaviour by abstracting the space of joint actions that those individuals can take. We provide the following working definition:

Definition 1 (Agent Abstraction)**.** *An approximate clustering of part or all of the action space of two or more other agents in the environment, performed by either another interacting agent or an outside observer for the benefit of its own learning and/or planning.*

What about the utility of such abstractions? For example, it would seem so obviously advantageous to abstract cells of a human into a whole, given how many there are, how well-separated they are from the outside-human environment, and how completely dependent they have evolved to become on the inside-human environment [**16**]. However, there are collectives of simple and complex organisms for which the abstraction is more tenuous insofar as its utility is less certain. The uncertainty about the utility grows when considering abstractions for groups of agents that are not so obviously acting collectively and reminds us that the utility of agent abstraction arises from the strength of the collective behaviour and how a behaving agent can make use of that knowledge in maximizing the value of its actions [**6**]. So, how can we measure the strength of collective behaviour, and how can we tie abstracted representations of this behaviour to formal utility in multi-agent reinforcement learning (MARL)?

Given the overhead inherent in identifying proper abstractions, is building this capability into artificial intelligence even advantageous? In this paper, we show formally that, yes, agent abstractions can help each agent navigate the learning and planning process in the face of the non-stationarity in the environment arising from the presence of other learning agents, a key challenge to efficient reinforcement learning in multi-agent settings.

This is perhaps not surprising, given that abstraction is a well-studied concept in reinforcement learning and there is a vast literature on state and temporal abstraction in single agent settings. So, in the MARL setting, where other agents can be viewed as part of the environment, there is a natural extension of these ideas to abstracting the actions of other agents. Here, we begin paving that extension, bringing us a step closer to a good agent abstraction metric that can be deployed by agents in MARL settings.

We make the following theoretical contributions:

- We formulate agent abstraction as a special case of well-established bisimulation metrics and present a simple, but limited strategy to obtain one based on unique joint actions (Section 2.1).
- We define a compression measure inspired by a connection that we reveal between this abstraction and an improvement factor in the standard minimax regret bound for a RL agent (Section 2.2).
- We reduce a two-level MARL system to a single, low-level version that serves to test a compression measure's ability to reveal higher-level agency from the joint actions of low-level agents (Section 3.1).

Finally, in Section 3.2, we applied our reduction scheme to the game *Diplomacy* for which we obtained access to a large dataset of human-played games [**14**]. We show that despite its obvious limitations, the abstraction strategy we present gives a compression factor that correlates strongly with true player-groupings of unit agents controlled by individual human players. This suggests that more sophisticated metrics of the kind we outline that make better use of the action space structure could serve in forming useful agent abstractions.

2. Agent Abstraction for Behaving in Multi-Agent Environments

In MARL, the environment transition dynamics and reward function do not just depend on the environment state and actions from a single agent, but rather the joint space of actions of all agents acting in the environment. To be concrete, in an environment with N agents the environment transitions dynamics can be expressed by $T(\boldsymbol{s}'|\boldsymbol{s},a^1,\ldots,a^N)$ with state $\boldsymbol{s}\in\mathcal{S}$, next state $\boldsymbol{s}'\in\mathcal{S}$, and an action for each agent $a^i\in\mathcal{A}^i\ \forall i\in\{1,\ldots,N\}\equiv\boldsymbol{N}$. Each agent i has their own reward function $R^i(\boldsymbol{s},a^1,\ldots,a^N)\in\mathbb{R}$ and policy $\pi^i(a^i|\boldsymbol{s})$ for generating actions. Whereas in single agent RL a stationary model of the environment can be learned as only a function of an agent's own behavior, in MARL an attempt to do this has an implicit dependence on the potentially changing policies of other agents. Without loss of generality, we will conduct our analysis from the perspective from an arbitrary agent 1: the transitions for this agent are given by $T(\boldsymbol{s}'|\boldsymbol{s},a^1)=\sum_{a^2\in\mathcal{A}^2,\ldots,a^N\in\mathcal{A}^N}[\pi^2(a^2|\boldsymbol{s})\times\cdots\times\pi^N(a^N|\boldsymbol{s})\times T(\boldsymbol{s}'|\boldsymbol{s},a^1,a^2,\ldots,a^N)]$. As such, even in decentralized and model-free settings, it is necessary for agents to predict the actions of other agents in order to stabilize learning [**11, 20, 12**]. This stability is then achieved by approximating an action value function $Q^{\boldsymbol{\pi}}(\boldsymbol{s},a^1,\ldots,a^N)$ over the joint policy space $\boldsymbol{\pi}=(\pi^1,\ldots,\pi^N)$ of all N agents.

Even in the best case scenario where all actions are observed and all policies are known ahead of time, a single agent can naively view this as a single agent RL problem with a state space augmented by the action space of other agents, $\mathcal{S}_1^+=\mathcal{S}\times\mathcal{A}^{-\mathbf{1}}$ where $\mathcal{A}^{-\mathbf{1}}=\mathcal{A}^2\times\cdots\times\mathcal{A}^N$. Without exploiting the structure in the state and action spaces, a well-known result in the RL literature [**13**] is that an agent cannot achieve minimax regret (*i.e.* best in worst-case) better than

$$\Omega\left(\sqrt{HT|\mathcal{S}_1^+||\mathcal{A}^1|}\right)=\Omega\left(\sqrt{HT|\mathcal{S}||\mathcal{A}^{-\mathbf{1}}||\mathcal{A}^1|}\right)=\Omega\left(\sqrt{HT|\mathcal{S}|\left(\prod_{i=2}^{N}|\mathcal{A}^i|\right)|\mathcal{A}^1|}\right)\ , \qquad (2.1)$$

where H is the episode horizon length (or the minimum diameter for continuing problems [**7**]), T is the number of steps in the environment, and Ω is standard notation for asymptotic lower-bound scaling behaviour. However, such structure often exists so that, *e.g.*, leveraging an abstract state space of reduced size can help significantly by reducing the $|\mathcal{S}|$ factor

in Equation 2.1. There is already a vast literature on constructing such abstractions [**3, 10, 19, 4, 5, 1, 24**]. In this work, we will focus instead on reducing the potentially much larger contribution for many agent settings from $\mathcal{A}^{-\mathbf{1}}$ by formulating abstractions on this action space of other agents in the environment.[11]

2.1. Agent Abstraction as a Bisimulation Metric

Our view of agent abstraction can be seen as a special case of bisimulation-based state abstraction metrics following the results of [**3**]. The factored state view of agent abstraction presented previously can indeed be seen as a special case of a general MDP over the augmented state space $\mathcal{S}_1^+$ from the perspective of an arbitrary agent 1. This leads to the following definition for state abstraction bisimulation metric $d(x,y)\ \ \forall x,y \in \mathcal{S}_1^+$ (Lemma 4.1 of [**3**]) leveraging the Wasserstein distance function between distributions $\mathcal{W}$:

$$d(x,y) = 0 \Leftrightarrow R^1(x,a^1) = R^1(y,a^1) \ \text{ and } \ \mathcal{W}\left(T(\cdot|x,a^1),T(\cdot|y,a^1)\right) = 0 \ \ \forall a^1 \in \mathcal{A}^1 \ . \tag{2.2}$$

For agent abstraction, we are interested in further decomposition of the augmented state space, $\mathcal{S}_1^+$. To illustrate, let us focus on whether an abstraction is valid between only a pair of agents, $i \neq 1$ and $j \neq 1$ $(i \neq j)$, for which we consider the decomposition $\mathcal{S}_1^+ = \mathcal{S} \times \mathcal{A}^i \times \mathcal{A}^j \times \mathcal{A}^{\text{rest}}$, where $\mathcal{A}^{\text{rest}}$ denotes the joint action space of all other agents not including agent 1. We can then narrow the scope of the metric onto $\mathcal{A}^i \times \mathcal{A}^j$.

Definition 2. *A **bisimilar agent abstraction metric** for agent 1 on a pair of agents i and j with any pair of joint actions $a^{ij} = (a^i,a^j)$ and $a^{i'j'} = (a^{i'},a^{j'})$ satisfies:*

$$\begin{aligned} d(a^{ij},a^{i'j'}) = 0 \Leftrightarrow R^1(s,a^1,a^i,a^j,a^{rest}) = R^1(s,a^1,a^{i'},a^{j'},a^{rest}) \ \ \textit{and} \\ \mathcal{W}\left(T(\cdot|s,a^1,a^i,a^j,a^{rest}),T(\cdot|s,a^1,a^{i'},a^{j'},a^{rest})\right) = 0 \ \ \forall a^1 \in \mathcal{A}^1 \ . \end{aligned} \tag{2.3}$$

For example, consider a partition, $\mathcal{C} = \{C_k\}$, on $\mathcal{A}^i \times \mathcal{A}^j$. Then, $a^{ij} \in C_k$ and $a^{i'j'} \in C_{k'}$ for some k and k', respectively. The semi-metric $d_{\mathcal{C}}(a^{ij},a^{i'j'}) = 1 - \delta_{kk'}$ always satisfies Equation 2.3 when C_k and $C_{k'}$ contain only a^{ij} and $a^{i'j'}$, respectively. This is true when $\mathcal{C}$ is the set of all singletons for which $|\mathcal{C}| = |\mathcal{A}^i \times \mathcal{A}^j|$. We are interested instead in partitions that compress the joint action space, *i.e.* for which $|\mathcal{C}| < |\mathcal{A}^i \times \mathcal{A}^j|$. The exactness of partitions, however, limits their usefulness as a basis for constructing bisimulation metrics, especially in the typical case of stochastic joint action dependencies. We can thus broaden our notion of agent abstraction by specifying the following ϵ-approximate abstraction, following the general form proposed in [**3**].

Definition 3. *An **ϵ-bisimilar agent abstraction** by agent 1 for the joint action a^{ij} of agents i and j within a given neighborhood ϵ identifies any two joint actions a^{ij} and $a^{i'j'}$ if*

[11] There are cases where this may not be true; however, it does not call in question the underlying benefits of doing such abstraction.

the metric

$$d(a^{ij},a^{i'j'}) = \max_{a^1\in\mathcal{A}^1}\Big[c_R\Big|R^1(s,a^1,a^i,a^j,a^{rest}) - R^1(s,a^1,a^{i'},a^{j'},a^{rest})\Big| + c_T\mathcal{W}\Big(T(\cdot|s,a^1,a^i,a^j,a^{rest}),T(\cdot|s,a^1,a^{i'},a^{j'},a^{rest})\Big)\Big] \leq \epsilon\,, \qquad (2.4)$$

where c_R and c_T are weighting constants such that $c_R, c_T \geq 0$, $c_R + c_T \leq 1$, and $c_T \geq \gamma$, where γ is the discount factor.

The central result of this formulation is that the optimal value function defined over an agent-abstracted state space (based on this kind of metric) is guaranteed to be within $\frac{2\epsilon}{c_R(1-\gamma)}$ of the optimal value function on $\mathcal{S}$ for agent 1 (following Theorem 5.2 of [**3**]). The primary goal of constructing an agent abstraction is then to maximize the compression of the joint action space such that $|\mathcal{A}^i \times \mathcal{A}^j|/|\mathcal{C}|$ is as large as possible (*i.e.* $|\mathcal{C}|$ as small as possible) while keeping ϵ in Equation 2.4 as small as possible.

2.2. Unique Joint Actions Experienced as a Bisimilar Agent Abstraction

Note that an arbitrary fixed policy π^i need not leverage its full action space when used in the environment. We can denote by $|\mathcal{A}^i_w|$ the total number of unique actions in a realized action sequence up to time w that agent i has taken in the environment. Thus, $|\mathcal{A}^i_w| \leq \min\{w,|\mathcal{A}^i|\}$ so that in the limit $w \to \infty$, $|\mathcal{A}^i_\infty| \leq |\mathcal{A}^i|$. Without considering any additional structure then, this result can be used to obtain an improved minimax regret bound, $\Omega\left(\sqrt{HT|\mathcal{S}|\left(\prod_{i=2}^N |\mathcal{A}^i_\infty|\right)|\mathcal{A}^1|}\right)$. One important structural constraint of MARL not exploited in this result is the fact that every policy π^i is a function of the current state $\boldsymbol{s}$. Thus, to the extent that the actions taken by each agent are correlated with this state, it is very possible that large regions of the joint action space will never be experienced at any single state. Exploiting these correlations for a subset $\boldsymbol{K} \equiv \{i_1,\ldots,i_K\} \subset \boldsymbol{N}$, of $K = |\boldsymbol{K}|$ agents suggests an improvement factor of as much as $\sqrt{\left(\prod_{k=1}^K |\mathcal{A}^{i_k}_\infty|\right)/|\mathcal{A}^{\boldsymbol{K}}_\infty|}$ in the minimax regret, where $|\mathcal{A}^{\boldsymbol{K}}_\infty|$ denotes the number of unique joint actions experienced in $\mathcal{A}^{\boldsymbol{K}} \equiv \mathcal{A}^{i_1}\times\cdots\times\mathcal{A}^{i_K}$.

For illustration, consider two arbitrary agents i and j and the set of unique joint actions taken in the environment over a window of time w, which we denote $\mathcal{A}^{i,j}_w$ ($|\mathcal{A}^{i,j}_w| \leq w$). Note that in the limit $w \to \infty$, $|\mathcal{A}^{i,j}_\infty| \leq |\mathcal{A}^i_\infty||\mathcal{A}^j_\infty| \leq |\mathcal{A}^i \times \mathcal{A}^j| = |\mathcal{A}^i||\mathcal{A}^j|$. Some examples of these sets are given in Figure 2.1(a-c). Importantly, the partition of $\mathcal{A}^i \times \mathcal{A}^j$ into singleton sets of the unique action pairs that are counted to obtain $|\mathcal{A}^{i,j}_\infty|$ (with the complement of their union added as an element) must by definition satisfy Equation 2.3 and thus serves as a straightforward (if not optimal) bisimilar agent abstraction since it is easy to implement. We

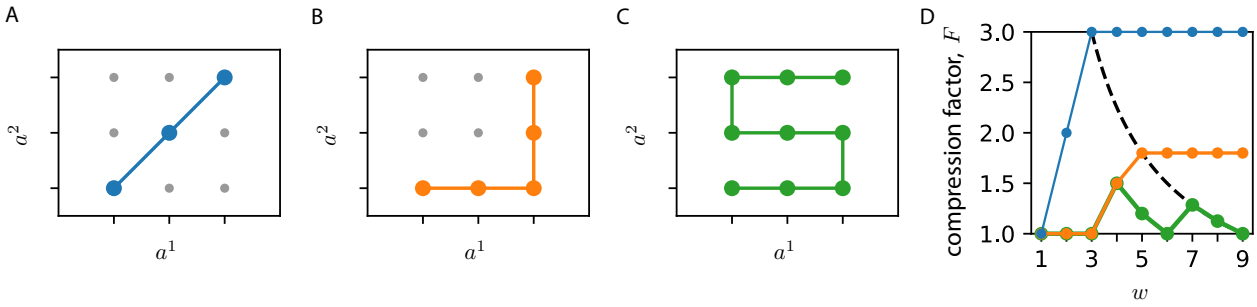


Fig. 2.1. Illustrative examples of abstracting the joint action space of two agents. Three contrived cases of trajectories in the action space (a^1,a^2) of two agents (the coverage of the respective trajectories is shown in color): **(a)** the copy case where the two agents behave identically; **(b)** the iterative case, where agents take turns sequencing through their actions; and **(c)** the space-filling case via snaking along coordinate directions. **(d)** The compression factor $F(\boldsymbol{a}_{0w}^{\{1,2\}})$ Equation 2.5 as a function of w for cases (a-c) (same colors). The maximum possible value $|\mathcal{A}|^{K-1}$ is attained by case (a; blue). Cases (a) and (b) grow to a fixed value $F(\boldsymbol{a}_{0\infty}^{\{1,2\}}) > 1$ with w because their trajectories do not fill the joint action space. The values for trajectories that do fill the space (e.g. case (c)) end up on $\max\{1,|\mathcal{A}|^K/w\}$ (black-dashed line). ($K = 2$, $|\mathcal{A}| = 3$.)

suggest some strategies for retrieving optimally compressed abstractions in the discussion, but leave their development to future work.

The form of the regret improvement factor and this metric that partitions the joint action space into visited joint actions suggests a definition for a crude measure of the utility of abstracting an action block, i.e. the joint action trajectories of a subset of $\boldsymbol{K}$ agents over a time interval from t to t', denoted $\boldsymbol{a}_{tt'}^{\boldsymbol{K}}$: The *compression factor* achievable by abstracting an action block $\boldsymbol{a}_{tt'}^{\boldsymbol{K}}$ (formed from the subset $\boldsymbol{K} \subset \boldsymbol{N}$, of $K = |\boldsymbol{K}|$ agents over the interval from t to t') with unique joint actions is the multiplicative factor,

$$F(\boldsymbol{a}_{tt'}^{\boldsymbol{K}}) := \left(\prod_{k=1}^{K} n(\boldsymbol{a}_{tt'}^{\{i_k\}})\right) \Big/ n(\boldsymbol{a}_{tt'}^{\boldsymbol{K}}) \geq 1 \,. \tag{2.5}$$

where $n(\boldsymbol{a}_{tt}^{\boldsymbol{K}})$ is the number of unique joint actions in the action block for the agent subset $\boldsymbol{K}$.

This factor is largest for the contrived case of $|\mathcal{A}|$-periodic joint action sequences with non-repeating single agent actions in the period (here we assumed for simplicity that all agents have the same action space, $\mathcal{A}$). In this case, $F(\boldsymbol{a}_{tt'}^{\boldsymbol{K}}) = x^K/x = x^{K-1}$ for $x = \min\{t'-t,|\mathcal{A}|\}$ (Figure 2.1(a)). See Figure 2.1 for more examples.

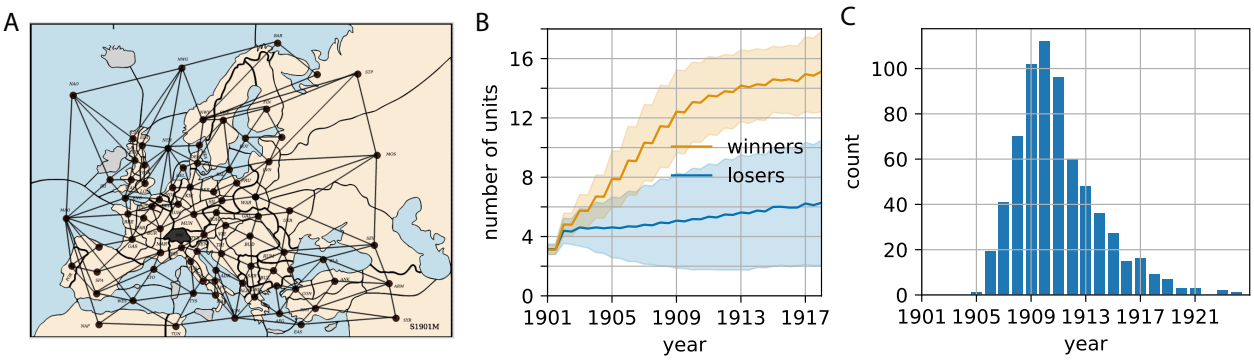


Fig. 3.1. The Diplomacy dataset. **(a)** The graph of positions and movement lines along which units move (adapted from [**14**]). There are $|\mathcal{V}| = 81$ positions, $|\mathcal{V}^j| = \{3,4\}$ number of unit-build locations/player, $|\mathcal{V}_{\text{supply}}| = 34$ supply centers, $n_{\text{players}} = 7$ players and $n_{\text{units}} = 18$ units/player. **(b)** Number of in-the-game units versus time in the game (2 time steps/year). Mean and standard deviation over $n_{\text{games}} = 10^3$ randomly selected games from the [**14**] no-press Diplomacy dataset of the $\sim 10^5$ human-player games are shown, grouped by the winning player (orange), and the rest (blue). Players aim to increase over the course of the game the size of the pool of units they control. **(c)** Histogram of game durations, $p(t_{\text{f}})$, over the same games as used in (b).

3. Measuring Player Control in Multi-Unit, Multi-Player Games

Here, we investigate the compression factor (Definition 2.2) as a measure of higher-level agency. In particular, when applied to a set of multi-agent trajectories, does it reflect the degree to which they can be said to be coordinating? To have access to a ground truth higher-level agency to test with, we focus on two-level, multi-agent settings, in which a set of higher-level controllers ('players', indexed by $j = 1,\ldots,n_{\text{players}}$) mutually compete for resources using their control of a set of lower-level agents ('units', with each player allotted the same number of n_{units} units indexed by $i = 1,\ldots,n_{\text{units}}$). We marginalize out the effects of the players that are not directly tied to unit actions, leaving a multi-agent Markov decision process (without reward) of player-labeled units indexed by (i,j). We then perform two analyses: (1) using compression factors to classify pairs units as belonging to the same versus two different players; and (2) the compression factor dependence on the number of a subset of units that belong to the same player. As an application, we focus on the board game *Diplomacy*, for which we analyzed 1000 games of a large dataset of human-played games (see Figure 3.1; [**14**]). In this section, we first give a precise formulation of a unit-level description of the game in 3.1 (to which we transformed the player action-structured data from [**14**]), then in 3.2 we present the statistics of the compression factor computed on multi-unit action sequences constructed from the data.

3.1. Multi-Unit Markov Games on Graphs

Here we present a unit-level formulation of Markov games suited to describing even complicated games like Diplomacy. The environment is a graph, $\mathcal{G} = (\mathcal{V},\mathcal{E})$, with set of unit positions $\mathcal{V}$ and set of valid lines $\mathcal{E}$ along which units move between positions. We augment the graph with a set of $n_{\text{players}} \cdot n_{\text{units}}$ *out-of-game* positions (see Appendix 1.1), denoted $v_\emptyset$, one for each unit. The state of the ith unit of the jth player is then $s^{ij} \in \mathcal{S} = \{v_\emptyset\} \cup \mathcal{V}$. The state of the environment is then the tuple of positions occupied by all units, $\boldsymbol{s} = (s^{11},\ldots,s^{1n_{\text{units}}},s^{21},\ldots,s^{n_{\text{players}}n_{\text{units}}}) \in \mathcal{S}^{\otimes(n_{\text{players}}\cdot n_{\text{units}})}$[12]. There are fixed, player-labelled unit-build locations, $\mathcal{V}^j \subset \mathcal{V}$, $\mathcal{V}^j \cap \mathcal{V}^{j'} = \emptyset$ to which units of that player transition from their *out-of-game* position when they are 'built'. Units transition to their *out-of-game* position when they are 'disbanded' from any in-game position as a result of an engagement (for details about engagement and other aspects of a Markov formulation of Diplomacy, see Section 1.1).

Gameplay requires action selection for all the units, which we consider stochastic. At the player-level, action selection arises from a given set of player policies, $\{\pi^j = \pi(a^{1j},\ldots,a^{n_{\text{units}}j}|\boldsymbol{s})\}_{j=1}^{n_{\text{players}}}$. Note that the conditioning on the state $\boldsymbol{s}$ means that each player could play the same, putative optimal policy, π^*, in which case their play is distinguished by the different starting positions of their respective units, encoded in $\boldsymbol{s}_0$. For any given set of player policies, the joint action given the state $\boldsymbol{s}$ is determined by the effective joint policy $\boldsymbol{\pi} = (\pi^1,\ldots,\pi^{n_{\text{players}}})$. Thus, for a given environment state distribution, $p(\boldsymbol{s})$, the game state distribution is $p^{\boldsymbol{\pi}}(\boldsymbol{s},\boldsymbol{a}) = \boldsymbol{\pi}(\boldsymbol{a}|\boldsymbol{s})p(\boldsymbol{s})$.

The game dynamics are given by $T^{\boldsymbol{\pi}}(\boldsymbol{s}'|\boldsymbol{s}) = \sum_{\boldsymbol{a}} T(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a})\boldsymbol{\pi}(\boldsymbol{a}|\boldsymbol{s})$. Here, $T(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a})$ is a deterministic map and encodes the game rules, including resolutions of engagement for complicated spatial configurations. In contrast, the game evolution $T^{\boldsymbol{\pi}}(\boldsymbol{s}'|\boldsymbol{s})$ inherits stochasticity from $\boldsymbol{\pi}$, such that the variance of state distributions over games increases with time in the game from zero at their shared initial state, $\boldsymbol{s}_0$. In particular, the distribution of the environmental state evolves as the Markov chain given by $T^{\boldsymbol{\pi}}(\boldsymbol{s}'|\boldsymbol{s})$, $p^{\boldsymbol{\pi}}(\boldsymbol{s}') = T^{\boldsymbol{\pi}}(\boldsymbol{s}'|\boldsymbol{s})p^{\boldsymbol{\pi}}(\boldsymbol{s})$. For game time $t = 0,1,\ldots$, we have $p^{\boldsymbol{\pi}}(\boldsymbol{s}_t) = (T^{\boldsymbol{\pi}})^t\, p(\boldsymbol{s}_0)$, with $\boldsymbol{s}_t = (s_t^{11},\ldots,s_t^{n_{\text{players}}n_{\text{units}}})$. Note that for Diplomacy, the initial state distribution,$p(\boldsymbol{s}_0) = 1_{\{\boldsymbol{s}_0\}}$, is concentrated on $\boldsymbol{s}_0$, the deterministic starting state. Thus, $p^{\boldsymbol{\pi}}(\boldsymbol{s}_t,\boldsymbol{a}_t) = \boldsymbol{\pi}(\boldsymbol{a}_t|\boldsymbol{s} = \boldsymbol{s}_t)p^{\boldsymbol{\pi}}(\boldsymbol{s}_t)$ with $\boldsymbol{a}_t = (a_t^{11},\ldots,a_t^{n_{\text{players}}n_{\text{units}}})$.

The state distribution depends on time throughout the game even for fixed $\boldsymbol{\pi}$, since the dynamics approaches the termination condition linearly in n_{units} (*i.e.* one captured supply center allows for transitioning one unit into the game), while the mixing time of $T^{\boldsymbol{\pi}}(\boldsymbol{s}'|\boldsymbol{s})$ that sets the characteristic time until the stationary distribution is reached scales exponentially with n_{units} (keeping $n_{\text{units}}/|\mathcal{V}|$ fixed).

[12]Positions can not be occupied by more than one unit so $s^{ij} = s^{i'j'}$ only when $i' = i$ and $j' = j$.

A realization of a game initialized at $\boldsymbol{s}_0$ is produced by sampling joint actions according to $\boldsymbol{a}_t \sim \boldsymbol{\pi}(\cdot|\boldsymbol{s} = \boldsymbol{s}_t)$ and successive states from $\boldsymbol{s}_{t+1} \sim T(\cdot|\boldsymbol{s} = \boldsymbol{s}_t,\boldsymbol{a} = \boldsymbol{a}_t)$. A complete realization of a game is the corresponding sequence of 'environment state'-'joint action' pairs $\tau_t = (\boldsymbol{s}_t,\boldsymbol{a}_t)$, $\boldsymbol{\tau} = (\tau_1,\ldots,\tau_{t_\mathrm{f}})$, where $t_\mathrm{f} = \min\{t|\mathrm{T}(\boldsymbol{s}_t)\}$ is the (stochastic) time at which the termination condition is first satisfied and the game ends. The distribution over games for this $\boldsymbol{\pi}$ is denoted $p^{\boldsymbol{\pi}}(\boldsymbol{\tau}) = \sum_{t_\mathrm{f}=1}^{\infty} p^{\boldsymbol{\pi}}(\boldsymbol{\tau}|t_\mathrm{f})p^{\boldsymbol{\pi}}(t_\mathrm{f})$ with $p^{\boldsymbol{\pi}}(\boldsymbol{\tau}|t_\mathrm{f}) = \left(\prod_{t=0}^{t_\mathrm{f}-1} T^{\boldsymbol{\pi}}(\boldsymbol{s}_{t+1}|\boldsymbol{s} = \boldsymbol{s}_t,\boldsymbol{a} = \boldsymbol{a}_t)\boldsymbol{\pi}(\boldsymbol{a}_t|\boldsymbol{s} = \boldsymbol{s}_t)\right) 1_{\{\boldsymbol{s_0}\}}$ and $p^{\boldsymbol{\pi}}(t_\mathrm{f})$ the distribution of game durations. For a subset of units, $\boldsymbol{K} = \{i_k\}_{k=1}^{K}$ for $i_k = (i,j)$ for the ith unit of player j, the action block is denoted $\boldsymbol{a}_{tt'}^{\boldsymbol{K}} = (\boldsymbol{a}_t^{\boldsymbol{K}},\ldots,\boldsymbol{a}_{t'}^{\boldsymbol{K}})$. Over the full set of agents, we use $\boldsymbol{a}_{tt'} \equiv \boldsymbol{a}_{tt'}^{\boldsymbol{N}}$ for simplicity. Action blocks are realizations from the distribution, $p^{\boldsymbol{\pi}}(\boldsymbol{a}_{tt'}^{\boldsymbol{K}}) = \sum_{t_\mathrm{f}} \sum_{\boldsymbol{s}_{0t_\mathrm{f}},\boldsymbol{a}_{t'+1,t_\mathrm{f}}^{\boldsymbol{K}}} p^{\boldsymbol{\pi}}(\boldsymbol{\tau}|t_\mathrm{f})p^{\boldsymbol{\pi}}(t_\mathrm{f}|\{t_\mathrm{f} > t'\})$.

3.2. Compression Factor for Multi-Unit Abstraction

Within a realization $\boldsymbol{\tau}$ of the game, the number of unique joint actions of the $\boldsymbol{a}_{tt'}^{\boldsymbol{K}}$ block, i.e. the joint actions of the $\boldsymbol{K}$ subset of agents over the interval from t to t', can be written

$$n(\boldsymbol{a}_{tt'}^{\boldsymbol{K}}) = \sum_{\tilde{\boldsymbol{a}}^{\boldsymbol{K}} \in \mathcal{A}^K} \Theta\left(\sum_{\tilde{t}=t}^{t'} \delta_{\boldsymbol{a}_{\tilde{t}}^{\boldsymbol{K}},\tilde{\boldsymbol{a}}^{\boldsymbol{K}}}\right) , \tag{3.1}$$

where $\Theta(x) = 1$ for $x > 0$ and 0 otherwise. Note that $n(\boldsymbol{a}_{tt'}^{ij}), n(\boldsymbol{a}_{tt'}^{\boldsymbol{K}}) \leq t' - t + 1$. Using this in the definition of the compression factor Definition 2.2 $F(\boldsymbol{a}_{tt'}^{\boldsymbol{K}})$ makes it a random variable depending on the game realization $\boldsymbol{\tau}$ sampled from $p^{\boldsymbol{\pi}}(\boldsymbol{\tau})$. For example, the expected compression factor over the game ensemble for the joint policy $\boldsymbol{\pi}$ is then

$$\overline{F(\boldsymbol{a}_{tt'}^{\boldsymbol{K}})} = \sum_{\boldsymbol{a}_{tt'}^{\boldsymbol{K}}} p^{\boldsymbol{\pi}}(\boldsymbol{a}_{tt'}^{\boldsymbol{K}})F(\boldsymbol{a}_{tt'}^{\boldsymbol{K}}) . \tag{3.2}$$

We can increase the signal in this factor by conditioning on subsequences for which $\left\{s_{\tilde{t}}^{\boldsymbol{K}} \in \mathcal{V}^K \;\forall\; \tilde{t} \in \{t,t+1,\ldots,t+t'\}\right\}$, *i.e.* the event that all the K agents are in the game between t and t'. The probability of this event vanishes quickly with increasing K and $t'-t$.

To distinguish player information, however, we need only compare pairs of units from the same and different players, $\boldsymbol{P}_{\mathrm{same}} = \{i_1,i_3\}$ and $\boldsymbol{P}_{\mathrm{different}} = \{i_1,i_2\}$, respectively, with $\boldsymbol{K} = \{i_1 = (i,j),i_2 = (i',j'),i_3 = (i'',j))\}$ with $j' \neq j$. We thus track the actions of a given unit triple $\boldsymbol{K}$ in the game over the interval from t to t'. The corresponding compression factor abstracting the pair of agents over the window from t to t' is

$$F(\boldsymbol{a}_{tt'}^{\boldsymbol{P}}) = n(\boldsymbol{a}_{tt'}^{ij})n(\boldsymbol{a}_{tt'}^{i'j'})/n(\boldsymbol{a}_{tt'}^{\boldsymbol{P}}) , \tag{3.3}$$

for unit pair, $\boldsymbol{P} = \{(i,j),(i',j')\}$. Applying this to the pair of same and pair of different player units in the triple blocks of length w that begin at time t in the game gives us

$$\chi_{t,w}^{\boldsymbol{K}} = (F(\boldsymbol{a}_{tt+w}^{\boldsymbol{P}_{\mathrm{same}}}), F(\boldsymbol{a}_{tt+w}^{\boldsymbol{P}_{\mathrm{different}}})) . \tag{3.4}$$

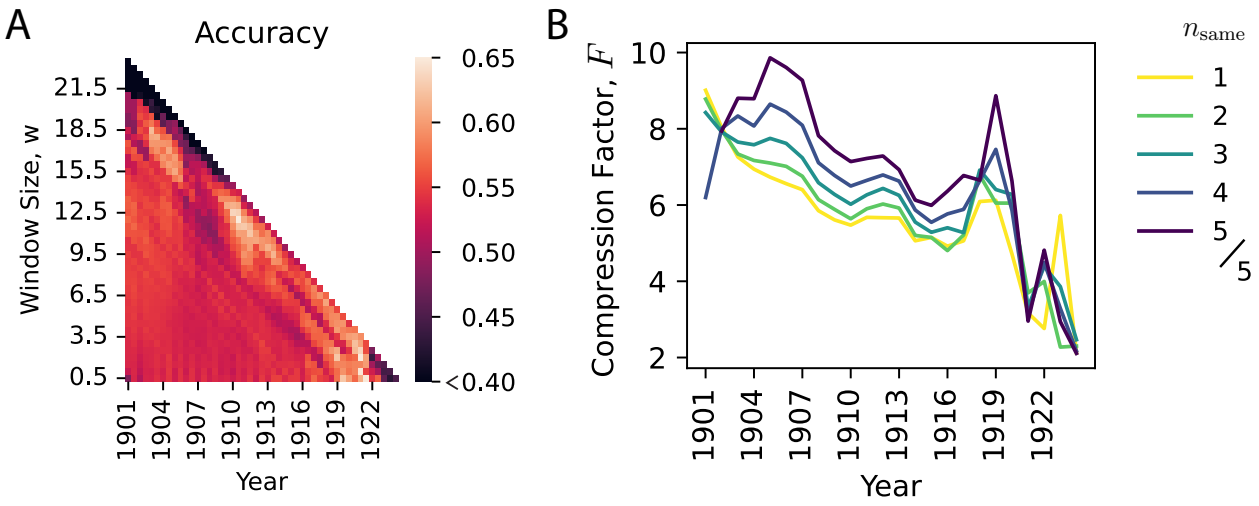


Fig. 3.2. Compression factor for subsets of units from Diplomacy. **(a)** The average same-different classification accuracy based on Equation 3.4 as a function of block window size, w, and block start time, t, in years ($n_{\text{games}} = 10^3$). **(b)** The average compression factor, $\bar{F}$ Equation 3.2, as a function of block start time for $n_{\text{same}} = 1,2,3,4,5$ out of $K = 5$ agents in a subset that belong to the same player (average over window size, w; $n_{\text{games}} = 10^3$).

In obtaining the following results, we coarse-grained the single agent action space into $\mathcal{A} \in \{\text{H,M,S}\}$ (see Appendix 1.1). We calculated $\chi^{\boldsymbol{K}}_{t,w}$ over the set of all in-game unit triples $\mathcal{K} = \{\boldsymbol{K}\}$ ordered by t and w. To assess the discriminability of this variable, we compute the classification accuracy based on the fraction of the number of $\chi^{\boldsymbol{K}}_{t,w}$ tuples where $F(\boldsymbol{a}^{\boldsymbol{P}_{\text{same}}}_{tt+w}) > F(\boldsymbol{a}^{\boldsymbol{P}_{\text{different}}}_{tt+w})$ to the number of $\chi^{\boldsymbol{K}}_{t,w}$ tuples where $F(\boldsymbol{a}^{\boldsymbol{P}_{\text{same}}}_{tt+w}) \neq F(\boldsymbol{a}^{\boldsymbol{P}_{\text{different}}}_{tt+w})$. This accuracy is plotted as a function of t and w in Figure 3.2(a). The diagonal structure shows that there are periods in the game that are more informative than others. The highest discriminability occurs at an intermediate time in game.

Beyond discriminability, it remains to show that the compression factor correlates with how many units, $n_{\text{same}} = 1,\ldots,K$, in $\boldsymbol{K}$ belong to the same player (with the remaining $K - n_{\text{same}}$ units randomly sampled from the remaining players). We conditioned on blocks in which all K units are consistently in the game. This means that a unit is built at the beginning of the block and another is disbanded at its end. For computational limitations, we limited our analysis to $K = 5$. We plot the corresponding average compression factors as a function of block starting time for different n_{same} in Figure 3.2(b). We find that the curves are well-ordered by n_{same} and even reveal periods of enhanced compression when most units belong to the same player.

4. Related Work

Facilitating learning in MARL settings is a focus of much current research, only some of which are related to our work. For example, formation of effective teams and the emergence of player roles in teams [**21**], relates to partitioning the policy space to pull specialized agents from. This is in contrast to partitioning the joint action space as considered here. Learning prototypical/archetypal agents also falls into this interesting, but unrelated research area.

Abstraction, on the other hand, is a well-studied concept in reinforcement learning from which our work based on bisimulation metrics is a direct extension [**3**]. With these metrics now more accessible computationally via modern methods, *e.g.* function approximation [**1, 24**], they are poised to make an impact on RL and MARL in particular. Another related area of work is factorized MDPs (e.g. VAST, CAMPs), which aim to lift the curse of dimensionality by modelling the joint space through some factorized version [**15, 2**]. While this approach builds in high-level agent structure, our work focuses on how an agent might learn this latent structure. However, unlike inverse RL that extracts reward function, e.g. from observed communication [**23**]– our agent abstraction metric does not depend on reward. This makes it more generally useful to cases when rewards are not observed or modelled.

In [**22**], authors cluster an effect-based –measured by the action's induced reward and the change in observations–representation of the joint action space using K-means to facilitate role discovery, while our goal is to find partitions that maximally compress the joint action space from an agent's perspective.

5. Discussion

In this paper, we have presented the concept of agent abstraction and grounded it in existing formulations of abstraction. We also presented a crude metric based on unique joint actions to measure the degree of abstraction and showed it provides some signal in the complex multi-agent setting of Diplomacy. Nevertheless, this metric has some obvious limitations. First, it is realization-dependent at least for non-stationary settings. Second, it does not capture the correlations among experienced joint actions. A more useful version would be based on running estimates of frequencies of joint actions. Given this distribution, optimal compression schemes could be designed that cluster joint actions with equal probability such that the entropy of the distribution over the abstracted space is maximized. This optimization must be regularized by adding the constraint that an agent learning a value function using this space does so with bounded error, in the way that bisimulation metrics for RL have been designed to accomplish. There may be an interesting connection to be made here with the deterministic information bottleneck [**18**]. See Section 1.2 for a formulation and estimation procedure for the conditional mutual information on the action block distribution of a pair of units, $MI(A^{ij}_{tt'}; A^{i'j'}_{tt'} | \boldsymbol{S}_{tt'})$.

Our results on the game Diplomacy deserve some discussion. Why do subsets with few units from a single player give lower compression factors at early times? We speculate this is because the unconditioned case actually has more than 1 agent on average from the same player. More generally, the players in the dataset are played by different humans across samples. Thus, it is unclear to what degree the player label, i.e. the country, constrains this play variability across individual humans. While good games such as Diplomacy sculpt player agency into elaborate roles, individual differences for players of the same country are bound to impact our results, since we have likely not averaged over enough games to cover the space of possibilities. Nonetheless, our paper has taken critical first steps for the community to build on towards developing scalable methods that address agent abstraction. Our work has the promise to open up an important area of future research, particularly for combatting the inherent combinatorial complexity of large scale multi-agent RL applications.

Acknowledgments. The authors would like to thank the team of authors of [**14**], and Philip Paquette in particular, for collecting and providing us with the dataset.

1. Appendix

1.1. Markov Formulation for the Game of Diplomacy

The game always begins from the same state, $\boldsymbol{s}_0$, having units positioned at all respective unit-build locations. The remaining units are initialized at their *out-of-game* position. Occupation of each of the target subset of positions, $\mathcal{V}_{\text{supply}} \subset \mathcal{V}$, called *supply centers*, confers the ability to sustain an additional unit. Thus, the recurrent goal of the game is to capture supply centers in order to build more units to capture more supply centers. The termination condition that ends the game is the event $\mathrm{T}(\boldsymbol{s}) = \{s^{ij} \in \mathcal{V}\ \forall\ i,\ \text{for any } j\}$, *i.e.* when some player j has managed to occupy n_{units} supply centers such that all its units are in the game.

The full game has five distinct seasons of dynamics each year. However, marginalizing over the players allows us to reduce this to only the two seasons when units act. Each season, every unit must either *hold* (H), i.e. do nothing, *move* (M) to an adjacent position, or *support* (S) an adjacent position. We denote the action of the ith unit of the jth player located at position k, $a_k^{ij} \in \mathcal{A}_k = \{\mathrm{H}, \mathrm{M}_1, \ldots, \mathrm{M}_{|E_k|}, \mathrm{S}_1, \ldots, \mathrm{S}_{|E_k|}\}$, where E_k is the set of lines connected to positions k. Thus, the action space for each unit depends on its location (except for *out-of-game* positions from which actions have no effect). We can remove this dependence on location by combining all position-dependent action spaces, such that the action of the ith unit of the jth player $a^{ij} \in \mathcal{A} = \cup_{k=1}^{|\mathcal{V}|} \mathcal{A}_k$, where state-conditioning narrows the accessible actions to $\mathcal{A}_k$ when $s^{ij} = k$. Thus, similar to the joint state $\boldsymbol{s}$, the joint action is denoted $\boldsymbol{a} = (a^{11}, \ldots, a^{1n_{\text{units}}}, a^{21}, \ldots, a^{n_{\text{players}} n_{\text{units}}}) \in \mathcal{A}^{\otimes(n_{\text{players}} \cdot n_{\text{units}})}$.

When a pair of agents *engage*, *i.e.* when at least one acts such that they would occupy the same position, the unit having the larger support wins and can reside in that location, while the loser must retreat or be disbanded. A unit's support is the number of units supporting it, as well as itself. When engagement results in a draw (matching support), the effect of the movement actions precipitating the engagement are nullified.

1.2. Mutual Information Analysis

The respective pair action distribution at a single time t is

$$\begin{aligned} p^{\boldsymbol{\pi}}(a_t^{ij}, a_t^{i'j'} | \boldsymbol{s}_t) &= \sum_{\boldsymbol{a}_t / \{a_t^{ij}, a_t^{i'j'}\}} p^{\boldsymbol{\pi}}(\boldsymbol{s}_t, \boldsymbol{a}_t) \Big/ \sum_{\boldsymbol{a}_t} p^{\boldsymbol{\pi}}(\boldsymbol{s}_t, \boldsymbol{a}_t) \\ &= \begin{cases} \pi^{\{i,i'\}j}(a_t^{ij}, a_t^{i'j} | \boldsymbol{s}_t), & \text{if } j' = j \\ \pi^{\{i\}j}(a_t^{ij} | \boldsymbol{s}_t) \pi^{\{i'\}j'}(a_t^{i'j'} | \boldsymbol{s}_t), & \text{if } j' \neq j\ , \end{cases} \end{aligned} \tag{1.1}$$

where $\pi^{Ij}(\cdot|\boldsymbol{s}) \equiv \sum_{\{a^{ij}\}_{i\notin I}} \pi^j(a^{1j},\ldots,a^{n_{\text{units}}j}|\boldsymbol{s})$ is the marginal policy for the subset I of units of the jth player.

The mutual information between this pair of state sequence-conditioned unit action sequences, $a^{ij}_{tt'}$ and $a^{i'j'}_{tt'}$, is

$$MI(A^{ij}_{tt'}; A^{i'j'}_{tt'}|\boldsymbol{s}_{tt'}) = \sum_{a^{ij}_{tt'},a^{i'j'}_{tt'}} p^{\boldsymbol{\pi}}(a^{ij}_{tt'},a^{i'j'}_{tt'}|\boldsymbol{s}_{tt'}) \log\left[\frac{p^{\boldsymbol{\pi}}(a^{ij}_{tt'},a^{i'j'}_{tt'}|\boldsymbol{s}_{tt'})}{p^{\boldsymbol{\pi}}(a^{ij}_{tt'}|\boldsymbol{s}_{tt'})p^{\boldsymbol{\pi}}(a^{i'j'}_{tt'}|\boldsymbol{s}_{tt'})}\right] \tag{1.2}$$
$$\geq 0 \text{ with } 0 \text{ if } j' \neq j \ (c.f. \text{ Equation 1.1}) ,$$

and where we have denoted the unit action sequence marginals $p^{\boldsymbol{\pi}}(a^{ij}_{tt'}|\boldsymbol{s}_{tt'}) = \sum_{a^{i'j'}_{tt'}} p^{\boldsymbol{\pi}}(a^{ij}_{tt'},a^{i'j'}_{tt'}|\boldsymbol{s}_{tt'})$. This demonstrates that action sequences arising from coordinated units ($j' = j$) can be informative of each other. Can this mutual information serve to measure coordination more broadly and be used as a way to define an effective higher-level agency between observed units, even in the absence of prior information (j)?

Equation 1.2 can be rewritten in a more tractable form as

$$MI(A^{ij}_{tt'}; A^{i'j'}_{tt'}|\boldsymbol{s}_{tt'}) = \mathbb{E}_{p^{\boldsymbol{\pi}}(a^{i'j'}_{tt'}|\boldsymbol{s}_{tt'})}\left[D_{\text{KL}}[p^{\boldsymbol{\pi}}(A^{ij}_{tt'}|a^{i'j'}_{tt'},\boldsymbol{s}_{tt'})||p^{\boldsymbol{\pi}}(A^{ij}_{tt'}|\boldsymbol{s}_{tt'})]\right] , \tag{1.3}$$

with $p^{\boldsymbol{\pi}}(a^{ij}_{tt'}|a^{i'j'}_{tt'},\boldsymbol{s}_{tt'}) = p^{\boldsymbol{\pi}}(a^{ij}_{tt'},a^{i'j'}_{tt'}|\boldsymbol{s}_{tt'})/p^{\boldsymbol{\pi}}(a^{i'j'}_{tt'}|\boldsymbol{s}_{tt'})$. The conditional mutual information, $MI(A^{ij}_{tt'}; A^{i'j'}_{tt'}|\boldsymbol{S}_{tt'})$, averages Equation 1.3 also over $\boldsymbol{s}_{tt'}$. In this form it can be computed directly from a measured set of game trajectories, $\mathcal{D} = \{\boldsymbol{\tau}_g\}_{g=1}^{n_{\text{games}}}$, by approximating the expectation with respect to the game trajectory marginal over the window $p^{\boldsymbol{\pi}}(\boldsymbol{s}_{tt'},a^{i'j'}_{tt'})$ using Monte Carlo estimation [**17, 8**]:

$$MI(A^{ij}_{tt'}; A^{i'j'}_{tt'}|\boldsymbol{S}_{tt'}) \approx \frac{1}{n_{\text{games}}}\sum_{g=1}^{n_{\text{games}}} D_{\text{KL}}[p^{\boldsymbol{\pi}}(A^{ij}_{tt'}|a^{i'j'}_{tt',g},\boldsymbol{s}_{tt',g})||p^{\boldsymbol{\pi}}(A^{ij}_{tt'}|\boldsymbol{s}_{tt',g})] . \tag{1.4}$$

Here, the D_{KL} must still be computed via integration using the unit policies. This estimation still suffers from the curse of dimensionality because of the high dimensions of the distributions of game-long trajectories and so $t' = t + w$ with small w.

References

[1] Pablo Samuel Castro. Scalable methods for computing state similarity in deterministic markov decision processes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 10069–10076, 2020.

[2] Rohan Chitnis, Tom Silver, Beomjoon Kim, Leslie Pack Kaelbling, and Tomas Lozano-Perez. CAMPs: Learning context-specific abstractions for efficient planning in factored mdps. In *CoRL*, 2020.

[3] Norm Ferns, Prakash Panangaden, and Doina Precup. Metrics for finite markov decision processes. In *UAI*, volume 4, pages 162–169, 2004.

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

[5] Norm Ferns and Doina Precup. Bisimulation metrics are optimal value functions. In *UAI*, 2014.

[6] Pedro L Ferreira, Francisco C Santos, and Sérgio Pequito. Risk sensitivity and theory of mind in human coordination. *PLoS Computational Biology*, 17(7):e1009167, 2021.

[7] Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11:1563–1600, 2010.

[8] Natasha Jaques, Angeliki Lazaridou, Edward Hughes, Caglar Gulcehre, Pedro A Ortega, DJ Strouse, Joel Z Leibo, and Nando de Freitas. Intrinsic social motivation via causal influence in multi-agent rl. 2018.

[9] Michael Levin. The computational boundary of a "self": developmental bioelectricity drives multicellularity and scale-free cognition. *Frontiers in psychology*, 10:2688, 2019.

[10] Lihong Li, Thomas J Walsh, and Michael L Littman. Towards a unified theory of state abstraction for mdps. *ISAIM*, 4(5):9, 2006.

[11] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. pages 157–163, jan 1994.

[12] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6382–6393, 2017.

[13] Ian Osband and Benjamin Van Roy. On lower bounds for regret in reinforcement learning. *arXiv preprint arXiv:1608.02732*, 2016.

[14] Philip Paquette, Yuchen Lu, Seton Steven Bocco, Max Smith, Satya O-G, Jonathan K Kummerfeld, Joelle Pineau, Satinder Singh, and Aaron C Courville. No-press diplomacy: Modeling multi-agent gameplay. *Advances in Neural Information Processing Systems*, 32, 2019.

[15] Thomy Phan, Fabian Ritz, Lenz Belzner, Philipp Altmann, Thomas Gabor, and Claudia Linnhoff-Popien. VAST: Value function factorization with variable agent sub-teams. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[16] Daniel J. Povinelli and Todd M. Preuss. Theory of mind: evolutionary history of a cognitive specialization. *Trends in Neurosciences*, 18:418–424, 1995.

[17] D J Strouse, Max Kleiman-Weiner, Josh Tenenbaum, Matt Botvinick, and David J Schwab. Learning to share and hide intentions using information regularization. In *Neural Information Processing Systems (NeurIPS)*, 2018.

[18] DJ Strouse and David J. Schwab. The Deterministic Information Bottleneck. *Neural Computation*, 29(6):1611–1630, 06 2017.

[19] Jonathan Taylor, Doina Precup, and Prakash Panagaden. Bounding performance loss in approximate mdp homomorphisms. *Advances in Neural Information Processing Systems*, 21, 2008.

[20] Gerald Tesauro. Extending q-learning to general adaptive multi-agent systems. page 871–878, 2003.

[21] Tonghan Wang, Heng Dong, Victor R. Lesser, and Chongjie Zhang. ROMA: Multi-agent reinforcement learning with emergent roles. *ArXiv*, abs/2003.08039, 2020.

[22] Tonghan Wang, Tarun Gupta, Anuj Mahajan, Bei Peng, Shimon Whiteson, and Chongjie Zhang. Rode: Learning roles to decompose multi-agent tasks. *arXiv preprint arXiv:2010.01523*, 2020.

[23] Lei Yuan*, Jianhao Wang*, Fuxiang Zhang, Chenghe Wang, Zongzhang Zhang, Yang Yu, and Chongjie Zhang. Multi-agent incentive communication via decentralized teammate modeling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022.

[24] Amy Zhang, Rowan Thomas McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. In *International Conference on Learning Representations*, 2020.