Leveraging Factored State Representations for Enhanced Efficiency in Reinforcement Learning

Suau, M.

DOI
10.4233/uuid:e19a0363-d9cf-4f33-8936-757c268f27a1

Publication date
2024

Document Version
Final published version

Citation (APA)
Suau, M. (2024). *Leveraging Factored State Representations for Enhanced Efficiency in Reinforcement Learning*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:e19a0363-d9cf-4f33-8936-757c268f27a1

Leveraging Factored State Representations for Enhanced Efficiency in Reinforcement Learning

Dissertation

for the purpose of obtaining the degree of doctor at

Delft University of Technology

by the authority of the Rector Magnificus
prof.dr.ir. T.H.J.J. van der Hagen,
chair of the Board for Doctorates

to be defended publicly on
Friday 19, January 2024 at 12:30 hours

by

Miguel SUAU DE CASTRO

Master of Science in Mathematical Modelling and Computation,
Technical University of Denmark,
born in Lugo, Spain.

This dissertation has been approved by the promotors.

Composition of the doctoral committee:

Rector Magnificus,	chairperson
Dr. F.A. Oliehoek	Delft University of Technology, promotor
Dr. M.T.J. Spaan	Delft University of Technology, promotor

Independent members:

Dr. S. Magliacane	University of Amsterdam
Dr. E. Mocanu	University of Twente
Prof.dr. A. Jonsson	Pompeu Fabra University
Prof.dr.ir. B. De Schutter	Delft University of Technology
Prof.dr.ir. J.W.C. van Lint	Delft University of Technology

Printed by: ProefschriftMaken (`www.proefschriftmaken.nl`)
Front and Back: Adriana Bravo Salvà

An electronic version of this dissertation is available at

`https://repository.tudelft.nl/`

A mis padres, Lourdes y Antonio.

Summary

Reinforcement learning techniques have demonstrated great promise in tackling sequential decision-making problems. However, the inherent complexity of real-world scenarios presents significant challenges for its application. This thesis takes a fresh approach that explores the untapped potential of factored state representations as a means to enhance the efficiency of reinforcement learning.

Factored representations involve variables describing various features of the environment. These variables, along with their possible values, define the agent's states. Unlike standard representations, factored representations provide a unique perspective that enables us to gain deeper insights into the underlying structure of the environment and refine our understanding of the problem at hand.

By analyzing variable dependencies, we can abstract simplified representations of the environment states and construct computationally lightweight models. To do so, we will explore potential factorizations of key functions governing the reinforcement learning problem, such as transitions, rewards, policies, or value functions. These factorizations can be achieved by exploiting variable redundancies and leveraging relations of conditional independence.

This thesis proposes a set of methods that are shown to improve the efficiency and scalability of reinforcement learning in complex scenarios. We hope that the findings of this research contribute to showcasing the potential of factored representations and serve as inspiration for future research in this direction.

Samenvatting

Reinforcement learning technieken hebben veelbelovende resultaten laten zien bij sequential decision-making problemen. Echter zorgt de hoge complexiteit van de echte wereld nog voor aanzienlijke uitdagingen bij de toepassing ervan. Deze dissertatie volgt een nieuwe benadering die het potentieel van gefactoriseerde toestandsrepresentaties onderzoekt als een middel om de efficiëntie van reinforcement learning te verbeteren.

Gefactoriseerde representaties gebruiken variabelen om verschillende kenmerken van de omgeving te beschrijven. Deze variabelen, samen met hun mogelijke waarden, definiëren de toestanden van de agent. In tegenstelling tot standard representaties bieden gefactoriseerde representaties een uniek perspectief waardoor we een dieper inzicht krijgen in de onderliggende structuur van de omgeving en ons begrip van het probleem kunnen verfijnen.

Door de afhankelijkheden tussen variabelen te analyseren, kunnen we vereenvoudigde representaties van de toestand van de omgeving verkrijgen en computationeel goedkopere modellen bouwen. Om dit te bereiken onderzoekt dit proefschrift het gebruik van gefactoriseerde representaties van transities, rewards en zogenaamde 'value functions'. Deze factorisaties kunnen worden bereikt door gebruik te maken van variabele redundanties en relaties van voorwaardelijke onafhankelijkheid.

Dit proefschrift stelt een reeks methoden voor die de efficiëntie en schaalbaarheid van reinforcement learning in complexe scenario's verbeteren. We hopen dat de resultaten van dit onderzoek zullen bijdragen aan het aantonen van het potentieel van gefactoriseerde representaties en toekomstig onderzoek in deze richting zullen inspireren.

Resumen

Las técnicas de aprendizaje por refuerzo han demostrado ser muy prometedoras para abordar problemas de toma de decisiones secuenciales. Sin embargo, la inherente complejidad del mundo real sigue planteando importantes retos para su aplicación. Esta tesis adopta un nuevo enfoque que explora el potencial de las representaciones de estado factorizadas como medio para mejorar la eficiencia del aprendizaje por refuerzo.

Las representaciones factorizadas utilizan variables para describir diversas características del entorno. Estas variables, junto con sus posibles valores, definen los estados del agente. A diferencia de las representaciones estándar, las representaciones factorizadas proporcionan una perspectiva única que nos permite obtener una visión más profunda de la estructura subyacente del entorno y refinar nuestra comprensión del problema en cuestión.

Analizando las dependencias entre variables, podemos obtener representaciones simplificadas del estado del entorno y construir modelos computacionalmente más ligeros. Para lograrlo, esta tesis investiga posibles factorizaciones de las funciones que rigen el problema de aprendizaje por refuerzo, como transiciones, recompensas, políticas o funciones de valor. Estas factorizaciones pueden lograrse explotando possibles redundancias entre las variables y aprovechando relaciones de independencia condicional.

Esta tesis propone un conjunto de métodos que demuestran mejorar la eficiencia y escalabilidad del aprendizaje por refuerzo en escenarios complejos. Esperamos que los resultados de esta investigación contribuyan a mostrar el potencial de las representaciones factorizadas y sirvan de inspiración para futuras investigaciones en esta dirección.

Acknowledgements

This section is dedicated to the many individuals whose invaluable support made my Ph.D. journey possible.

First and foremost, I would like to express my gratitude to my supervisors, Frans Oliehoek and Matthijs Spaan, for believing in me when I applied to the Ph.D. program, for supporting me in the face of rejections, for teaching me not to take criticism personally, for challenging my ideas, and overall for helping me become a better researcher.

I would also like to thank my colleagues from the Influence project: Elena Congeduti, Rolf Starre, Aleksander Czechowski, and Alexander Mey, who played a fundamental role both academically and personally. I had a lot of fun with all of you in the office and during our travels. A special thank you to Jinke He, without whom completing my Ph.D. would not have been possible. Our extensive discussions about research and life were a highlight of my journey.

My time at Delft would not have been the same without the people at the Interactive Intelligence group, including Elie Saad, Ding Ding, Frank Kaptein, Thomas Moerland, Merijn Bruijnes, Luciano Cavalcante Siebert, Bernd Dudzik, Myrte Tielman, Catholijn Jonker, Willem-Paul Brinkman, Pradeep Murukannaiah, Catharine Oertel, Wouter Pasman, and those who joined us later: Nele Albers, Enrico Liscio, Carolina Jorge, Mani Tajaddini, Siddharth Mehrotra, Maria Tsfasman, Emma van Zoelen, Ruben Verhagen, Pei Yu Chen, Sietze Kuilman, Zuzanna Osika, Mohammed Al Owayyed, Frank Broz. Special thanks go to my friends Franziska Burger and Ilir Kola. Fran, thank you for always being so kind and welcoming, and for organizing all our social activities. Ilir, thank you for the unwavering support and advice during challenging times and for making my time in Delft much more enjoyable. I will miss hanging out with you, watching football, and sharing pizza. I would also like to thank Anita Hoogmoed for patiently handling all my inquiries

and Ruud de Jong for always being ready to help with a huge smile.

I am grateful to all the people at the RL group from whom I learned a great deal over these years, including Mustafa Mert Çelikok, Robert Loftin, Thiago Dias Simão, Canmanie Ponnamlbalam, Qisong Yang, Moritz Zanger, Davide Mambelli, Oussama Azizi, and Wendelin Böhmer, among others.

A heartfelt thanks to my fellow Spaniards in Delft, Marcos Pieras and Daniel Jarne. Marcos, thank you for making me feel closer to home; I enjoyed our bike rides and walks around Delft. Dani, you have been a huge support when I needed to relieve stress from work, always ready to hang out and have fun.

Cambio al castellano para agradecer a mis amigos: Luis Munar, Ana Rojas, Aina Riutord, Adriana Bravo, Gonzalo Mateos, Carlos Morell, Almudena Tecglen, Guillermo Abolafia, Laura Torra, Isabel Escolar, María de Juan, Kiko Sagristá, Guillermo de España, Andrés de España, Xisco Mascaró, Toni Trian, Biel Oliver, Miguel Colom, Valentín Galdós, Carlos Bolado y Alejandro Bolado, entre otros, el estar conmigo en los buenos y malos momentos. Espero teneros siempre cerca.

También me gustaría agradecer el apoyo a mi familia, a mis hermanas Elena e Inés, a mis cuñados Kike y Mariana, a mis sobrinos Marcos y Tomás, y en especial a mis padres, Antonio y Lourdes. Mamá, gracias por preocuparte tanto por mí, por estar siempre dispuesta a ayudarme y por el cariño en los momentos difíciles. Papá, gracias por apoyarme en mis estudios, enseñarme a valorar el trabajo y el esfuerzo, por hacerme sentir seguro, y por ayudarme en la toma de decisiones. Gran parte de todo esto os lo debo a vosotros. Me siento muy afortunado de tener una familia que me apoya y me quiere.

Finalmente, quisiera atribuir este éxito a mi novia, mi confidente, y mi mejor amiga, Clara. Gracias por estar a mi lado, por animarme en los peores momentos, por soportar mi mal carácter, por complementarme y hacerme mejor persona. Nunca habría logrado esto sin ti. Te quiero.

Contents

Chapter 1

Introduction

> *In the series of things those which follow are always aptly fitted to those which have gone before; for this series is not like a mere enumeration of disjoint things, which has only a necessary sequence, but it is a rational connection: and all existing things are arranged together harmoniously, so the things which come into existence exhibit no mere succession, but a certain wonderful relationship.*
>
> –Marcus Aurelius, Meditations

1.1 Motivation

Reinforcement learning (RL; Sutton and Barto, 2018) has emerged as a powerful framework for addressing sequential decision-making problems. RL involves training an agent to navigate an unknown environment through trial and error, learning from the feedback it receives in the form of rewards. The ultimate objective is to develop a policy that maximizes the cumulative reward over time. While RL has demonstrated effectiveness in small-scale problems, it faces significant challenges when applied to complex real-world scenarios (Dulac-Arnold et al., 2019), primarily due to their inherent complexity and dimensionality, which can hinder learning efficiency (Botvinick et al., 2019).

This thesis explores the potential of factored state representations (Boutilier et al., 2000) in addressing some of these challenges. Factored representations rely on a set of variables

describing certain features of the environment to determine the agent's state. Each combination of these variables defines a unique state.

Factored representations are convenient for several reasons. First, unlike standard representations that simply enumerate states, treating them as independent entities, factored representations explicitly capture relationships between states that can be exploited to learn useful state representations. The similarity between two states can be assessed by comparing the values of their corresponding state variables. Second, factored representations provide a natural way of forming state abstractions. Ignoring certain state variables has the effect of grouping together states that share the same values for the remaining ones (Boutilier et al., 2000; Dietterich, 2000). Finally, and perhaps most importantly, factored state representations can provide valuable insights into the structure of RL problems as they allow for a much finer inspection of the environment dynamics. Both transition and reward probabilities can be expressed in terms of the state variables; hence, we can find compact representations for them by analyzing the relationships between these variables (Boutilier et al., 1995; Kearns and Koller, 1999; Guestrin et al., 2003; Nair et al., 2005; Hengst et al., 2002; Jonsson and Barto, 2005).

1.2 Objective

The primary objective of this thesis is to improve the efficiency of RL. This is achieved by examining the probabilistic dependencies among variables in factored state representations and trying to identify ways to streamline certain computations. To do so, we will rely on the use of probabilistic graphical models (PGMs; Koller and Friedman, 2009). PGMs offer a compact and intuitive way of visualizing the joint probability distribution of a group of variables in a system. In particular, we will employ Dynamic Bayesian Networks (DBNs; Dean and Kanazawa, 1989; Murphy, 2002), which are a type of PGMs specifically designed to capture the interactions between variables over time. Below we describe the two approaches that we explore in this thesis to try to accomplish the above objective.

1.2.1 Local Simulators

Most practical applications of RL rely on simulators to provide initial training for the agent before deploying it in the real world (Bellemare et al., 2020; Degrave et al., 2022). Simulators are essential because modern RL methods typically require large amounts

of data to learn effective policies. However, simulators of complex environments can be expensive to run.

Yet, in many real-world problems, agents directly interact with only a subset of state variables. For example, consider the task of training an agent to control the traffic lights at a specific intersection in a large city. In this case, the agent's observations may be limited to information about the traffic in the local neighborhood. Additionally, since the impact of the agent's actions is local, the reward function may also be conditioned on local information only. Thus, one may argue that simulating the entire system is unnecessary because observations and rewards depend solely on the local subset of state variables. However, local state transitions can still be affected by non-local variables. The traffic patterns in the local neighborhood are affected by the traffic conditions in the rest of the city.

Fortunately, in the traffic example and in many other real-world systems, although the local neighborhood may be affected by the full set of state variables, it is only directly influenced by the road segments connecting the intersection to the rest of the city. In the language of probability theory, we can say that the local neighborhood is conditionally independent of the rest of the city, given these road segments. This property is leveraged by the theoretical framework of influence-based abstraction (IBA; Oliehoek et al., 2021) to demonstrate that by monitoring the value of those variables that influence the local region directly, the rest of the non-local state variables can be abstracted away, thus simplifying the computation of local state transitions. In Chapters 3 and 4, we will show how these theoretical insights can be translated into a practical method to design local simulators that mimic the influence of the non-local variables through machine learning models.

1.2.2 State Abstraction

Another approach to enhance the efficiency of RL is through state abstraction (Li et al., 2006). State abstraction involves grouping states into clusters and treating all states within the same cluster as unique entities. Most state abstraction methods (Singh et al., 1994a; McCallum, 1995b; Abel, 2020) consider states as a whole and form clusters based on behavioural equivalence (Givan et al., 2003; Ferns et al., 2006). In contrast, when working with factored representations, removing variables from the representation naturally leads to the grouping of states. This means we can form state abstractions by identifying the important state variables and ignoring the irrelevant ones (Boutilier et al., 2000; Dietterich,

2000). In fully observable environments, assuming the reward always depends on the same group of variables, this process simply implies determining whether a variable is needed to predict the current reward and the next transition.[1]

Unfortunately, performing state abstraction is more delicate under partial observability since a variable may be relevant even if it does not directly influence the next transitions and rewards. In such cases, before removing a variable, it is crucial to ensure that it neither influences nor is influenced by any of the hidden state variables, or at least not directly. Again, we can do so by analyzing the dependencies between variables. Exploring this idea will be the central focus of Chapter 5.

Finally, in Chapter 6 we shift the focus slightly away from efficiency and delve into the topic of generalization. We show that under certain policies agent's may be able to remove additional variables from their representation, which are deemed unnecessary under such policy but which may still be required when following other policies. This occurs due to spurious correlations induced by the policy between the state variables and the agent's future transitions and rewards. Removing additional variables can be advantageous as long as the agent consistently adheres to a fixed policy. However, it can lead to catastrophic outcomes if the agent deviates from its usual trajectories.

1.3 Contributions

Here we outline the main contributions of this thesis. These are divided in four items which correspond to the four papers the thesis is based on. The first two focus on the objective of designing lightweight local simulators (Section 1.2.1), while the second two investigate state abstraction (Section 1.2.2). Each paper is described in a separate chapter providing a more detailed explanation of its respective contributions.

- **Chapter 3: Influence-Augmented Local Simulators (Suau et al., 2022d).** In this paper, we study how to build local simulators of complicated systems. The proposed method, Influence-Augmented Local Simulators (IALS), which builds on the IBA framework, can be applied to problems where agents interact with a reduced portion of a larger environment while still being affected by global dynamics. IALS combines the use of local simulators to closely mimic the local dynamics with

[1] Strictly speaking, a variable is only relevant if it is necessary to predict the current reward and the next *abstract* state (Givan et al., 2003).

learned models that capture the influence of the global system. Through extensive experimentation, IALS is shown to considerably speed up the agent's training time while matching the performance of agents trained in the true environment.

- **Chapter 4: Distributed Influence-Augmented Local Simulators (Suau et al., 2022a).** This paper extends the IBA framework to multi-agent systems, demonstrating that simultaneous learning is possible without incurring major computational costs. We show how networked environments can be factorized into multiple regions such that the agents in the environment can be trained in separate simulators that run independently and in parallel. We call the resulting method distributed influence-augmented local simulators (DIALS). Our experiments reveal that DIALS not only reduces the runtime and improves scalability but also seems to mitigate the inherent non-stationarities of multi-agent learning.

- **Chapter 5: Influence-Aware Memory (Suau et al., 2022b).** This paper presents a method to improve learning in partially observable environments by abstracting away observation variables that have no direct influence on, or are not influenced by, the hidden state variables. By removing these irrelevant variables from the agent's memory, we can overcome convergence difficulties and enhance learning efficiency. To implement this idea, we propose a deep learning architecture that combines a feedforward and a recurrent neural network in parallel. The recurrent neural network receives only the variables that are strictly necessary for predicting the hidden state variables. Additionally, we introduce a mechanism that allows the agent to automatically select which variables to retain.

- **Chapter 6: Policy Confounding (Suau et al., 2023).** This paper uncovers the phenomenon of policy confounding, whereby, as a result of repeatedly sampling from a particular policy, agents learn state representations that are insufficient to predict future transitions and rewards in trajectories other than those followed under said policy. This is because the policy, acting as a confounder, introduces spurious correlations that the agent exploits to predict transitions and rewards. We provide a mathematical characterization of this phenomenon and a series of clarifying examples that illustrate how it occurs.

Chapter 2

Background

> *Jargon is both a necessity and a curse.*
>
> –Richard Hamming, The Art of Doing Science and Engineering: Learning to Learn

---

In this chapter, we will give a brief overview of the materials that are most relevant to understanding the rest of the thesis. In particular, we will review the topics of reinforcement learning and probabilistic graphical models. The latter will be utilized as analytical tools to analyze the structure of the environment and investigate the probabilistic dependencies among variables. Although we will touch on these topics only briefly, we will provide references to other works that cover them more extensively.

2.1 Reinforcement Learning

Reinforcement learning (RL; Kaelbling et al., 1996; Sutton and Barto, 2018) is concerned with training an agent to act in an environment in order to solve a particular task. The agent, which initially holds little knowledge of the world, must design a strategy by interacting with the environment. That is, by taking actions and observing how the environment evolves as a result of these actions. The strategy, normally referred to as a policy, is nothing else than a function that maps the agent's observations to actions. The agent receives feedback from the environment in the form of numerical rewards and,

through trial and error, learns which actions yield high rewards.

2.1.1 Markov Decision Processes

The agent-environment interaction is normally modeled as a Markov Decision Process (Puterman, 2014).

Definition 2.1.1. A Markov decision process (MDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$ where $\mathcal{S}$ is the set of possible states of the environment, $\mathcal{A}$ is the set of actions that are available to the agent, $T : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transitions function which, for all $s_t, s_{t+1} \in \mathcal{S}$ and $a \in \mathcal{A}$, determines the probability $T(s_{t+1} \mid s_t, a_t)$ of transitioning from s_t at time t to state s_{t+1} at time $t+1$ given that the agent took action a_t, and $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a function that determines the reward $R(s_t, a_t)$ received by the agent for being at state s_t and taking action a_t. We will sometimes use the shorthand R_t for simplicity.

The goal for the agent is to learn a policy $\pi : S \to \Delta(A)$ that maximizes the expected value Q, which is defined as the expected cumulative reward of being in state s_t, taking action a_t and following π thereafter,

$$Q^{\pi}(s_t, a_t) = \mathop{\mathbb{E}}_{\substack{s_{t+1} \sim T(\cdot \mid s_t, a_t) \\ a_{t+1} \sim \pi(\cdot \mid s_{t+1})}} \left[\sum_{k=t}^{K} \gamma^k R(s_k, a_k) \;\middle|\; s_t, a_t \right], \tag{2.1}$$

where the expectation is taken with respect to the policy π and the transitions function T. From these two distributions, we can draw entire state-action trajectories $\langle s_t, a_t, ..., a_{K-1}, s_K \rangle$, also known as episodes, from which we can obtain the rewards at every timestep. K is a discrete random variable, known as the horizon, that determines the episode length. K can vary from episode to episode and may depend on the environment state. The summation inside the expectation is normally referred to as return G_t. The parameter $\gamma \in (0, 1]$ is the so-called discount factor, which ensures that the Q values are bounded when K goes to infinity, and determines how much the agent should care about immediate compared to future rewards. Alternatively, we can also express Q as

$$Q^{\pi}(s_t, a_t) = \mathop{\mathbb{E}}_{\substack{s_{t+1} \sim T(\cdot \mid s_t, a_t) \\ a_{t+1} \sim \pi(\cdot \mid s_{t+1})}} \left[R(s_t, a_t) + \gamma Q^{\pi}(s_{t+1}, a_{t+1}) \right], \tag{2.2}$$

The above expression, known as the Bellman equation (Bellman, 1957), reveals a nice property about the value function, which is that the value at a particular state can be

written recursively as a function of the value at its successor states. This property is the basis of many RL methods, some of which are reviewed in the next section.

2.1.2 Reinforcement Learning Methods

If the transitions function T is known, an optimal policy π^*, can be found through dynamic programming (Bertsekas, 2005). Unfortunately, this is typically not the case in RL, and the MDP must be solved by directly interacting with the environment. Yet, most RL methods rely on dynamic programming techniques, such as policy or value iteration (Puterman, 2014), to compute π^*. The idea of these methods is first to obtain an estimate $\hat{Q}^\pi(s_t, a_t)$ of the value of the current policy π, and then build a new policy π' that improves on the previous one by acting greedily with respect to $\hat{Q}^\pi(s_t, a_t)$. That is, at every state, π' selects the action that maximizes $\hat{Q}^\pi(s_t, a_t)$. This process is repeated multiple times until a fixed point is reached. Assuming that at every iteration, states are visited infinitely often and that the Q estimates are sufficiently close to the true values (i.e., they use a sufficiently large number of samples), the process is guaranteed to converge to the optimal policy π^* and optimal value function Q^*.

The main difference with respect to dynamic programming is that Q must be estimated from the experiences collected from the environment. The simplest way to estimate Q is to use Monte Carlo (MC) rollouts. Given an initial policy, we can obtain an estimate $\hat{Q}^\pi(s_t, a_t)$ by sampling multiple trajectories from the environment and averaging over the returns. An often more efficient alternative to Monte Carlo rollouts is temporal difference (TD) learning. Rather than sampling full trajectories, TD learning truncates them after, say, n timesteps and uses $\hat{Q}^\pi(s_{t+n+1}, a_{t+n+1})$ to estimate the value of the remaining part of the trajectory. TD Learning is used in classical RL methods such as Q-learning (Watkins and Dayan, 1992) and Sarsa (Rummery and Niranjan, 1994), and it is also at the core of some of today's most sophisticated RL methods (Mnih et al., 2015; Van Hasselt et al., 2016; Schulman et al., 2017; Hessel et al., 2018).[1]

2.1.3 Partial Observations

So far, we have assumed that states are fully observable to the agent. This is, unfortunately, not the case in many settings. Some of the information may actually be hidden or very

[1]The reader is referred to Sutton and Barto (2018, Chapters 4-9) for more details about classical RL methods.

hard to access. Think, for instance, of the game of blackjack; the cards in the deck are not visible to the players. Yet, players can estimate which ones are more likely to appear next by keeping track of the ones that have already been burned. This is a classic example of a partially observable setting. In the same way we did it with the fully observable case, we can also represent these types of problems mathematically using the Partially Observable MDP framework (Kaelbling et al., 1996).

Definition 2.1.2 (POMDP)**.** A partially observable Markov decision process (POMDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, T, R, \Omega, O \rangle$ where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, T and R are the transitions and rewards functions, as defined in Definition 2.1.1, Ω is the set of observations, and $O : \mathcal{S} \to \Delta(\Omega)$ is the observation or emission function, which, for all $s_t \in \mathcal{S}$ and $o_t \in \Omega$, determines the probability $O(o_t \mid s_t)$ of observing o_t given that the agent is at state s_t.

Since the agent receives only a partial observation o_t of the latent state s_t, a policy based only on the most recent information can be sub-optimal (Singh et al., 1994b). In general, the agent must keep track of past actions and observations to make the right action choices. The optimal policy is, therefore, a mapping from the past action-observation history (from now on simply referred to as history), $h_t = \langle o_1, a_1, ..., a_{t-1}, o_t \rangle$, to a probability distribution $\Delta(A)$ over actions A, $\pi : H \to \Delta(A)$, where H is the set of all possible histories of any length.

2.1.4 State Abstraction

It can happen that the state space, as defined by the MDP or POMDP models, is too granular. That is, the models may make more distinctions than are actually needed for the agent to solve the task. This occurs when two states are behaviorally equivalent from the perspective of the agent. The theory of state abstraction focuses on investigating these situations and provides tools for grouping states together (Singh et al., 1994a; McCallum, 1995b; Dietterich, 2000). The coarsest type of state abstraction guaranteeing the resulting abstract MDP to be equivalent to the original MDP is known as model-irrelevance state abstraction (Li et al., 2006) or bisimulation (Dean and Givan, 1997; Givan et al., 2003).

Definition 2.1.3 (Model-Irrelevance State Abstraction)**.** Let $\Phi : \mathcal{S} \to \bar{\mathcal{S}}$ be a function that maps ground states $s_t \in \mathcal{S}$ to abstract states $\bar{s}_t \in \bar{\mathcal{S}}$. Φ is a model-irrelevance state

abstraction if, for all $s_t \in S$, $a_t \in A$, $s'_t \in \{s_t\}^\Phi$, and $s_{t+1} \in S$,

$$R(s_t, a_t) = R(s'_t, a_t)$$

and

$$\sum_{s'_{t+1} \in \{s_{t+1}\}^\Phi} T(s'_{t+1} \mid s_t, a_t) = \sum_{s'_{t+1} \in \{s_{t+1}\}^\Phi} T(s'_{t+1} \mid s'_t, a_t),$$

where $\{s_t\}^\Phi = \{s'_t \in S : \Phi(s'_t) = \Phi(s_t)\}$ is s_t's equivalence class under Φ.

Simply put, Φ is a model-irrelevance state abstraction if it only groups together two states when, for any given action, these yield the same immediate reward, and their probabilities of transitioning to every possible abstract state are the same. State abstractions satisfying these conditions are guaranteed to be equivalent to the original representation. That is, for any policy and action, the expected value Q is the same when conditioning on the abstract states $\bar{s}_t$ or on the ground states s_t. This implies that the optimal policy is preserved when solving the abstract MDP induced by a model-irrelevance state abstraction. Unfortunately, finding a function Φ satisfying these conditions is often as hard as solving the original MDP. Hence, most of the work on state abstraction focuses on finding approximations (Abel et al., 2016; van der Pol et al., 2020; Zhang et al., 2021; Agarwal et al., 2021; Abel, 2020), some of which rely on metrics to measure the similarity between two states (Ferns et al., 2006; Castro, 2020). Despite potentially incurring a value loss, these approximations can be efficiently learned, are effective in shrinking the state space, and thus can help in solving the RL problem.

2.1.5 Function Approximation

The methods outlined in Section 2.1.2 are useful for solving moderately sized MDPs that have around a hundred or even a thousand states. However, real-world problems frequently have many more states, and, even if we could apply state abstractions, obtaining accurate point estimates of the value at each (abstract) state may require an impractically large number of samples. Moreover, we may also run into memory problems, given that the value estimates for each state-action pair must be stored separately in an array.

One way to address these limitations is to use function approximation. Instead of storing the value of each state explicitly, a parametric function can be trained to predict the value at any given state based on the observation variables that are provided as input. The hope

is that by training on a relatively small number of samples, the function approximator can exploit the hidden structure of the problem to accurately predict values, and even generalize to states that the agent has not encountered.

Neural networks are a popular choice when it comes to selecting a type of function approximator. The subfield of RL that deals with the integration of neural networks into the RL framework is known as Deep RL. The two main categories of methods for training neural networks for RL are value-based and policy-based methods. Value-based methods (e.g., DQN; Mnih et al., 2015) follow the same principle as value iteration. The optimal value function is learned iteratively, by estimating the value of the current policy while simultaneously improving it by acting greedily. The most common objective function for training a neural network to approximate the value function is the mean-squared error (MSE). A gradient can be easily derived from the MSE to optimize the weights of a neural network through gradient descent. In contrast, policy-based methods aim to learn the optimal policy directly. These methods rely on the policy gradient (Marbach and Tsitsiklis, 2001; Sutton et al., 1999) to optimize a neural network whose output is a distribution over the actions. Yet, only a few methods (e.g., REINFORCE; Williams, 1992) truly optimize the policy directly using MC estimates of the return. Most methods are actually hybrid. In the sense, that despite optimizing a policy network through the policy gradient, they also learn a value function (e.g. actor-critic methods: A3C, Mnih et al., 2016; PPO, Schulman et al., 2017).[2]

Deep RL has demonstrated great success in tackling high-dimensional domains, including, large board games with millions of states (Silver et al., 2016), complex video games with raw images as observations (Vinyals et al., 2019), and robotic tasks with continuous action spaces Lillicrap et al. (2016). Yet, as we discuss in Chapters 3 and 4, neural networks are extremely sample-inefficient, and most applications rely on simulators to generate synthetic experiences that the agent can learn from before being deployed in the real world.

2.2 Probabilistic Graphical Models

A probabilistic graphical model (PGM) is nothing else than a visual representation of a probability distribution. PGMs do not convey any more information than what a set

[2]For an introduction to RL with function approximation, the reader is referred to Sutton and Barto (2018, Chapters 10-13).

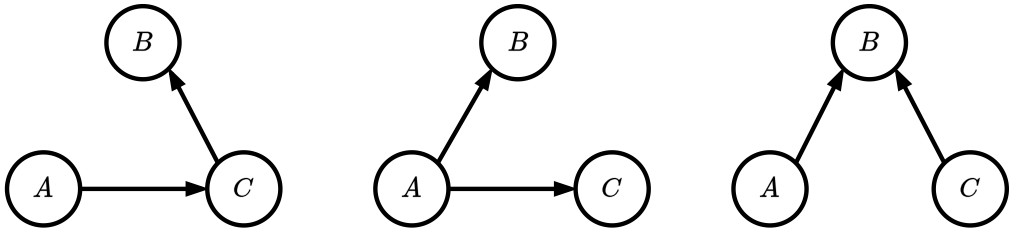


Figure 2.1: Three PGMs expressing different probabilistic relationships between three random variables, A, B, and C.

of equations defining the dependencies between the variables forming the distribution can convey. However, they allow us to readily visualize the structure of the distribution. In the following, we shall discuss some of the insights that PGMs can provide and show how complex computations can be simplified by making use of them. As in the previous section, we will only cover the essentials here.[3]

2.2.1 Factorization

Figure 2.1 shows three simple examples of PGMs. Specifically, these are known as Bayesian networks (BNs) or directed acyclic graphs (DAGs) (Pearl, 1988; Jensen and Nielsen, 2007). The three nodes in each of the diagrams represent random variables. The edges show the dependencies between variables, and the arrows indicate the direction of these dependencies. For instance, in the first diagram, we see that A is connected to C. We say that A is the parent of C, while C is node A's child. We also see B is connected to C, which means that C itself is a parent of B. Moreover, we say that A is an ancestor of B and B is A's descendant. As such, given the probabilistic dependencies shown in the first diagram, we can apply the product rule to decompose the joint probability of A, B, and C as

$$\Pr(A, B, C) = \Pr(B \mid C) \Pr(C \mid A) \Pr(A). \tag{2.3}$$

In the second diagram, we see that A is still connected to C, but the edge going from C to B is now gone. There is also another edge connecting A and B. Hence, in this second case, $\Pr(a, b, c)$ is given by

$$\Pr(A, B, C) = \Pr(A \mid C) \Pr(C \mid A) \Pr(A). \tag{2.4}$$

[3]The reader is referred to Bishop (2006, Chapter 8) for an overview, and to Koller and Friedman (2009) for full coverage of PGMs.

Finally, in the third diagram, we have that both A and C are connected to B,

$$\Pr(A, B, C) = \Pr(B \mid A, C) \Pr(C) \Pr(A). \tag{2.5}$$

What we did here is known as factorization, and although these examples were fairly simple, PGMs are generally very useful for determining how complicated distributions can be factorized. In fact, the above can be generalized into an expression that works for PGMs defining joint distributions of any kind. Let X be a set of variables $X = \{X^1, X^2, ..., X^n\}$. Assuming we are provided a diagram like the ones in Figure 2.1 showing the dependencies among variables, we can factorize their joint probability $\Pr(x)$ as

$$\Pr(X) = \prod_{i=1}^{n} \Pr(X^i \mid \mathbf{PA}(X^i)) \tag{2.6}$$

where $\mathbf{PA}(X^i)$ denotes the parents of X^i. For the above to hold, it is important that the PGM contains no cycles. That is, any path starting from a particular node should never end at that same node.

2.2.2 Conditional Independence and D-Separation

Another advantage of PGMs is that they give away any conditional independences between the variables forming the distribution. Two variables A and B are said to be conditionally independent given a third one C, written as $A \perp\!\!\!\perp B \mid C$, if

$$\Pr(A \mid B, C) = \Pr(A \mid C) \tag{2.7}$$

One can test for conditional independence, by, starting from expressions like the ones in (2.3), (2.4), and (2.5), trying to get to an expression like (2.7). However, depending on the number of variables, this process can be really tedious.

In contrast, using PGMs, we can immediately tell whether two variables A and B are conditionally independent given a third one C by testing for d-separation (Pearl, 1988). This involves checking whether the paths connecting the two nodes representing these variables are active or inactive. If, given C, there are no active paths connecting A and B, then we can guarantee that A and B are conditionally independent. In such cases, we say that the two variables are d-separated. We will use again the examples in Figure 2.1 to define what we mean by active and inactive paths.

In the first PGM, we see that A influences B through C. Therefore, in general, A and B are not independent. However, if we are given C, then A becomes irrelevant since C blocks the only path connecting A and B. Paths of the form $A \rightarrow C \rightarrow B$ are known as head-to-tail. Head-to-tail paths are open unless the middle node (in this case, C) is given.

In the second PGM, B and C are both children of A, and thus they cannot influence each other. Nevertheless, B can give away information about A, which in turn can be used to infer B. Hence, B and C are generally not independent unless A is given, in which case A, C cannot help in inferring B since we already know everything about A. Paths of the form $B \leftarrow A \rightarrow C$ are known as tail-to-tail. Tail-to-tail paths are open unless the middle node (in this case, A) is given.

The last PGM is perhaps the most interesting of the three. In this case, both A and C influence B, but they do not influence each other. Moreover, knowing A does not tell us anything about C since, in contrast to the second case, they do not have any common ancestors. However, if B was given, then knowing A may tell us something about C. We can illustrate this with an example from Pearl (1988) (taken from Scheines, nd). Let us assign the three Boolean variables 'dead battery', 'car will not start', and 'empty tank' to nodes A, B, and C in the third PGM, respectively. Knowing that the battery is dead tells us that the car will not start, but it does not tell us anything about the gas tank (nor it does knowing that the battery is charged). For all we know, the gas tank could also be empty. In contrast, if we know that the battery is not dead and that the car does not start, the tank being empty becomes more likely. Paths of the form $A \rightarrow B \leftarrow C$ are known as head-to-head. Head-to-head paths are closed unless the middle node (in this case, B) is given. The difference with respect to the previous two cases is that node B in the third PGM is a collider. Colliders act differently than non-colliders in the sense that when their values are given, any paths between two nodes that pass through the collider become active. Moreover, if rather than being given the collider itself, we are given one of its descendants, the path will also become active. Going back to the car example, being told that the car's owner could not go to work by car and knowing that the car's battery was charged, reveals more information about the gas tank than knowing only that the owner could not use the car.

As we will see in the following chapters, conditional independence is important because it can be exploited to (1) reduce the complexity of generative models (e.g. simulators; Chapters 3 and 4), and (2) narrow down the function class we start from when learning inference models (e.g. policies or value functions; Chapter 5).

2.2.3 Confounding

In causal inference (Pearl et al., 2016; Peters et al., 2017), the term confounding describes the emergence of a spurious correlation between two variables as a result of a third variable, known as the confounder, that influences both. A spurious correlation is a statistical association that does not respond to a causal relation.

A popular example is the spurious correlation between foot size and reading ability (Pearl and Mackenzie, 2018). If we collected a dataset containing only the foot size and the reading grades of a hundred children of different ages, we would see a strong correlation between the two. These two apparently unrelated variables happen to be statistically associated due to the influence of a hidden confounder: the children's age.

Learning in the presence of hidden confounders can be challenging, as machine learning models may pick up on spurious correlations and fail to generalize (Arjovsky, 2021). In Chapter 6, we will use PGMs to demonstrate how the agent's policy can sometimes act as a confounder and introduce spurious correlations.

2.2.4 Dynamic Bayesian Networks

We now describe the type of PGMs we will use throughout this thesis to represent RL problems. These are known as Dynamic Bayesian Networks (DBNs; Dean and Kanazawa, 1989; Murphy, 2002).[4] DBNs relate variables with one another across time. Figure 2.2 is an example of DBN. The edges in the graph show how variables at a particular point in time are related to other variables and to themselves at adjacent timesteps. DBNs are particularly well-suited to describe sequential decision problems like the ones we study in RL since transition dynamics are encoded in the network structure. Moreover, the Markov property of MDPs guarantees that edges cannot jump over multiple timesteps. That is, edges only connect variables within the same timestep (i.e., intra-stage connections) or from one timestep to the next one. This implies that a DBN showing only two timesteps, known as a two-stage DBN (2DBN), is sufficient to represent the transition dynamics. Figure 2.2 shows an example of a DBN representation of an MDP. We use circles for state variables, boxes for actions, and diamonds for rewards.

[4]Strictly speaking, the graphical models we will use can be considered a hybrid of DBNs and influence diagrams (Shachter, 1988), as we incorporate the agent's actions and rewards into the model.

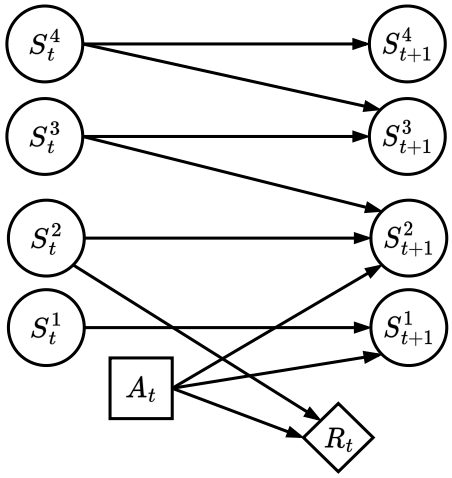


Figure 2.2: A DBN representation of an MDP with four state variables. We use circles for state variables, squares for actions, and diamonds for rewards.

2.3 Factored Representations

We now describe the types of models we will work with and provide their formal definitions.

Our work is based on the assumption that states are represented by a set of variables. This is common when parametric functions are used to model policies and value functions. These variables, also known as factors, typically describe features of the agent's state in the environment (e.g., location, velocity, orientation, etc.). Therefore, the set of states is given by the Cartesian product of these variables (i.e., the set of all combinations of values these variables can take). Factoring the state space has an effect on observations, rewards, and transitions, which can also be modeled in terms of the state variables.

As argued in Chapter 1, the main benefit of representing states using variables is that we can inspect the dynamics of the environment in more detail and potentially abstract away certain variables in observations, transitions, and reward distributions. We seek to exclude from the representation all those variables that are irrelevant for the agent to solve the task. Starting from the FPOMPD framework (Definition 2.3.1), we will be analyzing the relationships between state variables to find more efficient ways of representing the problem. The concepts reviewed in this section are the basis of our work on influence-augmented local simulators covered in Chapters 3 and 4, and influence-aware memory, Chapter 5.

2.3.1 Factored POMDPs

As this thesis focuses on partially observable settings, we now extend the POMDP framework to include factored representations (Boutilier et al., 1999).

Definition 2.3.1 (FPOMDP)**.** A factored partially observable Markov decision process (FPOMDP) is a tuple $\langle \mathcal{S}, S, \mathcal{A}, T, R, \Omega, O \rangle$ where $\mathcal{S}$ is the set of states, S is the set of state variables $S = \{S^1, ..., S^k\}$, such that every state $s_t = \langle s_t^1, ..., s_t^k \rangle \in \times_i \text{dom}(S^i) = \mathcal{S}$ is a k-dimensional vector, $\mathcal{A}$ is the set of actions, T and R are the transition and reward functions, as defined in Definition 2.1.1, Ω is the observation space, and O is the observation function, as defined in Definition 2.1.2.

2.3.2 Locality

One important property of the problems we tackle in this thesis is locality. In many situations, the agent's observations of the environment are limited and depend only on a local subset of the state variables. Moreover, given that the agent's effect on the environment is also limited (i.e., actions cannot influence all state variables), the same can be argued about the reward function. These two insights can be translated directly into a new framework, the *Local-FPOMDP* (Witwicki and Durfee, 2010a; Oliehoek et al., 2012), which is a special case of FPOMDP (Definition 2.3.1).

Definition 2.3.2. A Local-FPOMDP is an FPOMDP where O and R depend only on a subset of state variables $X = \{X^1, ..., X^{|X|}\} \subseteq S$ known as the local state variables. Each combination of these variables determines a different local state $x_t = \langle x_t^1, ..., x_t^{|X|} \rangle \in \times_i \text{dom}(X^i)$. Hence, we can write $O(o_t \mid s_t) = \dot{O}(o_t \mid x_t)$ and $R(s_t, a_t) = \dot{R}(x_t, a_t)$, where $\dot{O}$ and $\dot{R}$ are the local observation and local reward functions. Further, we use Y to denote the set of non-local state variables, $Y = X \setminus S$.

Looking at the above definition, one can argue that sampling from the full transition function $T(s_{t+1} \mid s_t, a_t)$ is overly complicated. Note that both rewards and observations condition only on local variables X_t. Hence, we may be able to define instead a new transition function $\bar{T} : \times_i \text{dom}(X^i) \times \mathcal{A} \to \Delta(\times_i \text{dom}(X^i))$ that models only the local state variables X, such that, for all $x_t, x_{t+1} \in \times_i \text{dom}(X^i)$ and $a \in \mathcal{A}$, $\bar{T}$ determines the probability $\bar{T}(x_{t+1} \mid x_t, a_t)$ of transitioning to the local state x_{t+1} given the current local state x_t and the action a_t. The problem, though, is that X_{t+1} may still depend on other non-local state variables $Y = F \setminus X$, and thus $\bar{T}$ is generally not well defined.

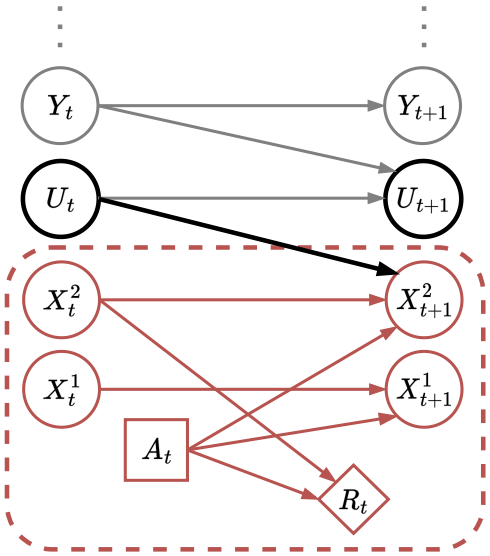


Figure 2.3: A DBN representation of a Local-FPOMDP. Red nodes represent local state variables, whereas grey nodes represent non-local state variables. The three dots on top of Y indicate that there can be, potentially many, other non-local variables. The red box delimits the agent's local region of interaction. The nodes highlighted in black depict the influence source U, which is the only non-local variable influencing the local region directly.

2.3.3 Influence-Based Abstraction

Here we briefly describe the framework of influence-based abstraction (IBA; Oliehoek et al., 2012).[5] From the previous section, we know that local states may sometimes depend on non-local state variables, which forces us to model the full set of state variables. However, as we will see in the following chapters, in many local-FPOMDPs, only a fraction of the state variables will *directly influence* the local region. IBA exploits this property to simplify the transitions function of the original FPOMDP. The idea is most easily understood by means of a DBN representing the problem.

The DBN in Figure 2.3 shows the dynamics of a local-FPOMDP prototype. The agent's local region, corresponds to the variables, denoted by $X = \langle X^1, X^2 \rangle$, that lie within the red box. The diagram also shows the non-local variables Y and U. The three dots on top of Y indicate that there can be, potentially many, other non-local variables. The

[5]The reader is referred to Oliehoek et al. (2021) for an in-depth exposition of the IBA framework.

non-local variable U is known as the influence source and, as shown in the DBN, is the only variable that influences the local region directly. Hence, we know that given U_t, X_{t+1} is conditionally independent of Y_t, $\Pr(x_{t+1} \mid x_t, u_t, y_t) = \Pr(x_{t+1} \mid x_t, u_t)$, which means that we can further simplify our model and convert the local-FPOMDP into an influence-augmented local model (IALM).

Definition 2.3.3 (IALM)**.** An influence-augmented local Model (IALM) is a local-FPOMDP with local transition function $\dot{T}(x_{t+1} \mid x_t, u_t, a_t)$, where, $u_t \in \times_{i=1}^{k} \text{dom}(U^i)$ are known as the influence sources, and are defined as the subset of non-local state variables, $U = \{U^1, ..., U^{|U|}\} \subseteq Y$, that influence X directly. Further, we define the influence distribution $I(u_t \mid l_t) = \Pr(u_t \mid l_t)$, where $l_t = \langle x_0, a_0, ..., a_{t-1}, x_t \rangle$ represents the action local state history (ALSH) with L_t denoting the set of local state X and action variables A in a an ALSH of length t. Using $\dot{T}$ and I, the probability of the next local state being x_{t+1} given the past ALSH l_t and the action a can be computed as

$$\Pr(x_{t+1} \mid l_t, a_t) = \sum_{u_t} \dot{T}(x_{t+1} \mid x_t, u_t, a_t) I(u_t \mid l_t). \tag{2.8}$$

Note that, as opposed to the Local-FPOMDP, the transition function $\dot{T}$ in the IALM is defined purely in terms of the local state variables and the influence sources. However, since the influence sources are not explicitly modeled, we need to infer their value through the the influence distribution. Yet, this representation of the problem can be shown to be equivalent (Oliehoek et al., 2012), and in many cases, computationally much lighter than the original FPOMDP. The problem is that even if we had the full transitions function T, computing the influence distribution I is generally intractable due to combinatorial explosion. Nonetheless, as will be shown in Chapter 3, we can approximately capture $I(u_t \mid l_t)$ by fitting machine learning models to samples collected from the environment.

2.3.4 D-Separating Set

In the same way that not all state variables are strictly relevant if we want to compute the local transitions, not all variables in L_t are needed in order to model the influence sources. For instance, if we take the DBN from Figure 2.3, we see that $\langle X_0^2, ..., X_{t+1}^2 \rangle$ and $\langle A_0, ..., A_t \rangle$ d-separate (Section 2.2.2) U_{t+1} from the rest of the ALSH. Hence, we know that U_{t+1} and L_{t+1} are conditionally independent given $\langle X_0^2, ..., X_{t+1}^2 \rangle$ and $\langle A_0, ..., A_t \rangle$. This set of variables, which is generally known as the d-separating set (d-set; Oliehoek et al., 2012), is highlighted in red in Figure 2.4.

Definition 2.3.4 (d-set)**.** The d-separating set (d-set) is the subset of variables in an ALSH, $D_t \subseteq L_t$, such that the influence sources U_t and the remaining variables in the ALSH, $L_t \setminus D_t$, are conditionally independent given D_t; $U_t \perp\!\!\!\perp L_t \setminus D_t \mid D_t$.

This insight can be used to simplify the influence distribution, since we have that, for all $u_t \in \times_i \text{dom}(U^i)$ and $l_t \in \times_i \text{dom}(L_t^i)$, $I(u_t \mid l_t) = \Pr(u_t \mid d_t)$. Moreover, as we will show in Chapter 5, it can also be incorporated into the agent's policy as an inductive bias to facilitate learning.

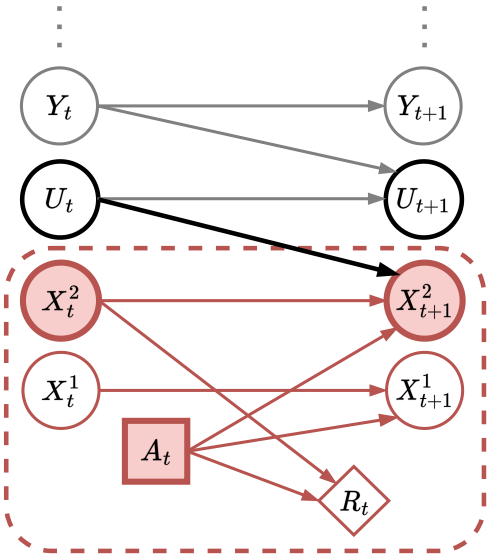


Figure 2.4: A DBN representation of the same Local-FPOMDP from Figure 2.3. The d-separating set is highlighted in red.

Chapter 3

Influence-Augmented Local Simulators

> *I basically know of two principles for treating complicated systems in simple ways: the first is the principle of modularity and the second is the principle of abstraction. I am an apologist for computational probability in machine learning because I believe that probability theory implements these two principles in deep and intriguing ways – namely through factorization and through averaging. Exploiting these two mechanisms as fully as possible seems to me to be the way forward in machine learning.*
>
> –Michael I. Jordan, 1997 (quoted in Murphy, 2022)

Learning effective policies for real-world problems is still an open challenge for the field of reinforcement learning (RL). The main limitation being the amount of data needed and the pace at which that data can be obtained. In this chapter, we study how to build lightweight simulators of complicated systems that can run sufficiently fast for deep RL to be applicable. We focus on domains where agents interact with a reduced portion of a larger environment while still being affected by the global dynamics. Our method combines the use of local simulators with learned models that mimic the influence of the global system. The experiments reveal that incorporating this idea into the deep RL workflow can considerably accelerate the training process and presents several opportunities for the future.[1,2]

[1] This chapter is based on Suau et al. (2022c) and Suau et al. (2022d).

[2] Source code is available at `https://github.com/INFLUENCEorg/IALS`

3.1 Introduction

The remarkable success of Deep Reinforcement Learning (RL) on paper is in stark contrast with its narrow applicability to real-world problems. Among many other reasons, the most important factor preventing the practical deployment of this framework is perhaps its high sample complexity (Botvinick et al., 2019). This is a very well-known issue, and there is a long list of previous works that, in one way or another, have tried to alleviate it (e.g., Kakade, 2003; Mnih et al., 2015; Ha and Schmidhuber, 2018). Nonetheless, there have been relatively few real-world successes thus far. Here, rather than proposing yet another method that tries to solve the problem directly, we present a more pragmatic approach to get around it. Our solution is based on the observation that Deep RL's best results have been obtained in domains like video games (Bellemare et al., 2013; Vinyals et al., 2019) or simulated environments (Brockman et al., 2016; Ganesh et al., 2019; Bellemare et al., 2020) where data collection is extremely fast. Unfortunately, real-world problems are typically more complex and simulators, if available, are usually very slow (Dulac-Arnold et al., 2019).

In this work, we design lightweight versions of large simulators with the goal of speeding up the overall training process. The method we propose applies to domains where agents only interact with a reduced local part of a larger environment, yet they are indirectly being affected by the global dynamics. Traffic control is one example of such environments. Say, for instance, that we wanted to train an agent to control the traffic lights at a particular intersection in a very large city. To do so, we could build a small local simulator that captures only the information that is directly relevant to the agent (traffic density in the neighborhood; van der Pol and Oliehoek 2016). However, after training, we may find that an agent that does very well in the small simulator performs poorly in the real intersection. The performance gap would be caused by a data distribution shift (Quionero-Candela et al., 2009; Arjovsky, 2021). Even though the simulator might be able to closely mimic the local dynamics (i.e. cars moving within the intersection), it would fail to account for the interactions of the local neighborhood with the rest of the city. Thus, the agent learns a policy based on certain transition dynamics that turn out to be very different in the real world. Alternatively, we could try to model the dynamics of a sufficiently large portion of the city, but this would surely result in a very slow simulator.

One important property of the traffic domain is that, although the agent's local problem may be *affected* by many external variables (traffic densities in other parts of the city), it

is only *directly influenced* by the road segments that connect the intersection with the rest of the city. Hence, we can simply monitor the traffic densities at these road segments since, from the agent's local perspective, they summarize the effect of all the external variables. This insight is not specific to the traffic domain. It is in fact common in networked systems (e.g. warehouse commissioning, Claes et al. 2017; electrical power grids, Wang et al. 2021; heating systems, Gupta et al. 2021; telecommunication networks, Suau et al. 2021) that interactions between different components occur through a limited number of variables.

Supported by the formal framework of *influence-based abstraction* (IBA; Oliehoek et al., 2021), we exploit the above property to build local simulators that mirror the response of the global system through the so-called *influence predictor*.

Contributions The main contribution of this chapter is the integration of the IBA framework with the Deep RL workflow. Our experiments reveal, that the combination of local simulators and influence predictors can considerably accelerate the reinforcement learning process. We also demonstrate both theoretically and empirically that, under mild conditions, the memory needs of the influence predictor are fully determined by the agent's memory capacity. Moreover, we study the impact that distribution shifts caused by changes in the agent's policy may have on the influence predictor and explore how to prevent the model from picking up on spurious correlations that are not invariant across policies. Finally, we investigate the effect of transfer learning and show how inaccurate simulators might also be able to render effective policies.

3.2 Related Work

The problem of sample complexity has been extensively studied by the RL community. Among many others, the most promising solutions are: replaying previous experiences to make more efficient use of the available data (Mnih et al., 2015; Schaul et al., 2016a; van Hasselt et al., 2019), or learning surrogate models of the environment dynamics (Sutton, 1990; Ha and Schmidhuber, 2018; Schrittwieser et al., 2020; Moerland et al., 2023). Yet, these techniques are only effective when provided with enough real samples. If not, replay buffers might not be sufficient to obtain good policies, and surrogate models might generalize poorly. An alternative is to train agents with synthetic data coming from a simulator. However, most real-world scenarios are excessively complex and simulators, if available, are computationally expensive (Dulac-Arnold et al., 2019). Here we argue that

building a simulator of the entire system is often unnecessary. In fact, as we explain in the following sections, in many situations, we can get away by just modeling the dynamics around the agent's local neighborhood.

A few prior works have investigated the computational benefits of factorizing large systems into independent local regions (Nair et al., 2005; Varakantham et al., 2007; Kumar et al., 2011; Witwicki and Durfee, 2011). Unfortunately, since local regions are often coupled to one another, such factorizations are not always appropriate. Nonetheless, in many cases, the interactions between regions occur through a limited number of variables. Using this property, the theoretical work by Oliehoek et al. (2021) on influence-based abstraction (IBA) describes how to build influence-augmented local simulators (IALS) of local-FPOMDPs, which model only the variables in the environment that are directly relevant to the agent while monitoring the response of the rest of the system with the influence predictor. The problem is the exact computation of the conditional influence distribution is intractable, and we can only try to estimate it from data. Congeduti et al. (2021) provide theoretical bounds on the value loss when planning with approximate influence predictors. The work by He et al. (2020) has empirically demonstrated the advantage of this approach to improve the efficiency of online planning in two discrete problems.

Here, we extend the method to more realistic problems and study how to integrate the IBA framework with Deep RL. This has profound implications that do not arise in the planning context, namely the relation between the agent's memory capacity and the history dependence of the influence predictor (Section 3.3.1), and the problem of off-policy generalization (Section 3.3.2). Moreover, while He et al. showed that the IALS outperforms the global simulator only when the time budget is limited, our results reveal that the IALS can train policies in a fraction of the time and that these can match the same performance as policies trained on the GS, without imposing any time constraints, and despite the IALS being only approximate.

3.3 Influence-Augmented Local Simulators

In the following, we describe how we make use of the IALM formulation (Definition 4.3.3) to design lightweight simulators that can speed up the long training times imposed by neural network policies. Figure 3.1 (right) shows a diagram of the influence-augmented local simulator (IALS), which is composed of a *local simulator* and an *approximate influence*

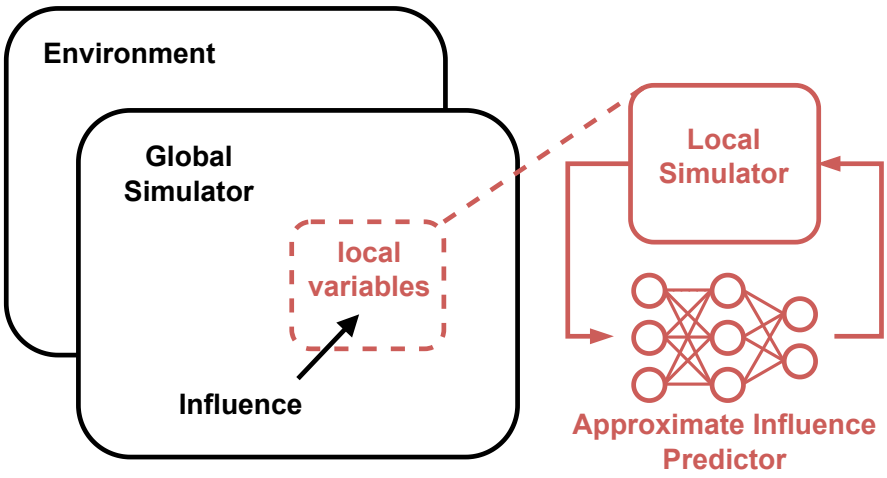


Figure 3.1: A conceptual diagram of the IALS.

predictor.

Local simulator (LS): The LS is an abstracted version of the environment that only models a small portion of it. As opposed to a global simulator (GS), which should closely reproduce the dynamics of every state variable, the LS focuses on characterizing the transitions of those variables x_t that the agent directly interacts with, $\dot{T}(x_{t+1}|x_t, u_t, a_t)$. Recall that, in our setting, both the reward R and observation O functions are defined in terms of x_t and a_t.

Approximate influence predictor (AIP): The AIP monitors the interactions between the local region and the external variables y_t by estimating $I(u_t|l_t)$. Ideally, we would like the approximation to match the true influence distribution. However, due to combinatorial explosion, computing the exact probability $I(u_t|l_t)$ is generally intractable (Oliehoek et al., 2021), and so we can only try to approximate it using, for instance, a parametric function. We write $\hat{I}_\theta$ to denote the AIP, where θ are the parameters, which need to be learned from data. Replacing the true influence distribution with an approximation implies that we are no longer guaranteed to find the optimal policy (Congeduti et al., 2021). Nonetheless, as we show in our experiments, it is often worth trading accuracy for computational efficiency. We model $\hat{I}_\theta$ using a neural network, which we train on a dataset of N samples of the form (l_n, u_n) collected from the GS (see Algorithm 4 in Appendix A.7). Since role of the AIP is

to estimate the conditional probability of the influence sources u_t given the past ALSH, we can formulate the task as a classification problem.[3] The neural network is optimized using the expected cross-entropy loss,

$$L(\hat{I}_\theta) = -\frac{1}{N}\sum_{n=0}^{N} \log \hat{I}_\theta(u_n|l_n), \tag{3.1}$$

which minimizes the KL divergence between the true probabilities $I(u_t|l_t)$ and those predicted by $\hat{I}_\theta(u_t|l_t)$.[4] Once the network has been trained, we can simulate trajectories with the IALS (Algorithm 5 in Appendix A.7). These trajectories can then be used to train policies with any standard Deep RL method (Mnih et al., 2015; Schulman et al., 2017).

In the next two sections we discuss two important considerations when training AIPs for the RL setting. We first show that, since the agent's memory is inevitably finite, we normally will not need to train complicated AIPs that condition on the full ALSH (Section 3.3.1). In Section 3.3.2, we study the impact that distribution shifts caused by changes in the agent's policy may have on the AIP, and explore how to prevent the AIP from picking up on spurious correlations that are not invariant across policies.

3.3.1 Finite Memory Agents and AIP History Dependence

As mentioned before, we can only rely on approximations of $I(u_t|l_t)$ since computing the exact distribution that conditions on the full ALSH is intractable. Unfortunately though, even with the most sophisticated RNNs (Hochreiter and Schmidhuber, 1997; Cho et al., 2014), learning long term temporal dependencies is also very difficult. However, it is worth noting that, if capturing long term dependencies is hard for the AIP, it is too for the agent. In fact, it is very common in Deep RL to find policies that have access only to finite memories (Mnih et al., 2015; Oh et al., 2016a) or that are trained on short sequences (Schmidhuber, 1991; Hausknecht and Stone, 2015). Thus, if the agents memory is finite, one might well wonder whether the extra level of accuracy that an influence predictor which conditions on the full ALSHs provides is needed. The result below shows that, the history dependence of the influence predictor is, under mild conditions, determined by the agent's memory capacity.

[3] Note that we assume for simplicity that the influence sources U are discrete variables. However, our approach can generalize to continuous variables by formulating the AIP learning problem as a regression rather than a classification problem.

[4] Please refer to Appendix A.6 for more details on the implementation of the AIP.

Theorem 3.3.1. *Let $l_{t-k:t}$ be a truncated version of l_t containing only the last k action-local-state pairs and let $a_{0:t-k}$ be the sequence of past actions from time 0 up to k. Let the agent's policy $\bar{\pi}$ be a function of $h_{t-k:t}$ $\bar{\pi}(a_t|h_{t-k:t})$, where $h_{t-k:t}$ is also a truncated version of h_t. If for all $u_t, a_{0:t-k}$ we have $P(u_t|l_{t-k:t}, a_{0:t-k}) = P(u_t|l_{t-k:t})$, then an influence predictor that conditions only on $l_{t-k:t}$, $\bar{I}(u_t|l_{t-k:t})$, is sufficient to guarantee no loss in value.*[5]

Proof. Found in Appendix A.1.

Intuitively, a finite memory agent whose policy conditions on the last k elements in the AOH is only capable of computing an expectation over the action-values given these k elements. In turn, the distribution of u_t given the full ALSH l_t, $I(u_t|l_t)$, is effectively washed away by the same expectation. The condition $P(u_t|l_{t-k:t}, a_{0:t-k}) = P(u_t|l_{t-k:t})$ is needed for $P(u_t|l_{t-k:t})$ to be well-defined independently of the policy. Intuitively, this just means that $a_{0:t-k}$ cannot have any long-term effects on the influence distribution (See proof in Appendix A.1 for more details). The upshot is that the agent's memory capacity limits the temporal dependencies of the influence predictor,[6] meaning that I may as well condition directly on $l_{t-k:t}$ rather than l_t. This insight is empirically evaluated in Section 3.4.4.

3.3.2 Off-Policy Generalization and D-sets

Given that the *true influence distribution* conditions on the full ALSH, it is, in principle, independent of the exploratory policy π_0 that we use to collect the data. Indeed, if we would represent $I(u_t|l_t)$ with a table, we could compute unbiased estimates of the true probabilities $I(u_t|l_t)$ by using any arbitrary policy that visits every possible ALSH. The problem arises when we use function approximators to estimate the influence sources. In particular, if we call $P^{\pi_0}(l_t, u_t)$ the stationary joint distribution of influence sources and ALSHs induced by the exploratory policy π_0 that we use to collect the data $D^{\pi_0} = \{(l_1, u_1), ...(l_N, u_N)\}$ from the GS (as described by Algorithm 4 in Appendix A.7), we can write

$$\theta^* = \arg\min_{\theta} L(\hat{I}_\theta, D^{\pi_0}), \tag{3.2}$$

where we see that, in fact, the loss in (3.1) depends on D^{π_0}. Therefore, because we are fitting $\hat{I}_\theta$ to D^{π_0}, the model will be biased towards P^{π_0}. This is not so bad considering

[5] Note that this refers only to the value of polices of the form $\bar{\pi}(a_t|h_{t-k:t})$, which might perform arbitrarily worse than policies that condition on the full AOH, $\pi(a_t|h_t)$ (Singh et al., 1994b).

[6] More details on the practical implications of this result are given in Appendix A.6.

that we want the influence predictions to be more reliable for those ALSHs that the agent visits more often. However, it can pose a problem when the policy π that we train starts deviating from π_0. In general, we want the AIP to perform well on any distribution that the agent may experience during training, $\{P^\pi\}_{\pi\in\Pi}$, where Π denotes the set of possible policies. We then rewrite the optimization problem in (3.2) as

$$\theta^* = \arg\min_\theta \max_{\pi\in\Pi} L(\hat{I}_\theta, D^\pi) \tag{3.3}$$

to indicate that we wish to minimize the worst-case loss.

The key issue here is that, before training, we have only access to the exploratory policy π_0. Equation (3.3) is an instance of a well-studied problem in the field of out-of-distribution generalization (Quionero-Candela et al., 2009; Arjovsky, 2021). In principle, according to Arjovsky (2021, Section 3.6.1), finding a solution to (3.3) by minimizing (3.2) requires, **(i)** $\text{supp}(P^\pi) \subseteq \text{supp}(P^{\pi_0})$, **(ii)** infinite number of samples, and **(iii)** infinite capacity models.

The first condition is met so long as $\pi_0(a_t|l_t) > 0$ for all a_t and l_t. The second and third conditions, on the other hand, can never be fulfilled in practice. On top of that, high-capacity influence predictors are particularly undesirable for our purpose because they are computationally demanding. The consequence of training low-capacity models on finite datasets is that they may pick-up on spurious correlations (Pearl, 2000) between ALSHs and influence sources.[7] These correlations could be just an artifact of π_0 and may disappear after the policy is updated. One way to prevent the above, is to find a representation of l_t that elicits an invariant predictor of u_t across all P^π (Peters et al., 2016; Arjovsky et al., 2019; Krueger et al., 2021).

Definition 3.3.1 (invariant predictor)**.** A subset of variables D_t from L_t elicit an invariant predictor of U_t across policies $\pi \in \Pi$ if for all $d_t \in \times_i \text{dom}(D_t^i)$, $u_t \in \times_i \text{dom}(U_t^i)$ in the intersection of supports, $\text{supp}(P^{\pi_1}(d,u)) \cap \text{supp}(P^{\pi_2}(d,u))$, we have

$$P^{\pi_1}(u_t|d_t) = P^{\pi_2}(u_t|d_t) \quad \text{for all} \quad \pi_1, \pi_2 \in \Pi. \tag{3.4}$$

Theorem 3.3.2. *A subset of variables D_t from L_t is guaranteed to elicit an invariant predictor of U_t, across all $\pi \in \Pi$, if (i) D_t constitutes a d-separating set (Definition 2.3.4) and (ii) all policies are functions of the local AOH, $\pi(h_t)$.*

Proof. Found in Appendix A.1.

[7] A visual example of a spurious correlation appearing in the traffic problem is given in Appendix A.2.

Note that, according to the result above, the full ALSH l_t does elicit an invariant predictor of u_t since it is, by definition, a d-set. Hence, feeding l_t should be sufficient for the model to find stable and invariant correlations. The problem, however, is that in practice, models tend to converge to low-capacity solutions that require little "effort" to learn (Arjovsky, 2021). As such, the representations formed by an influence predictor that is fed the full ALSH may or may not constitute a d-set. The solution we propose is to feed solely a minimal d-set d_t^* into the AIP $\hat{I}_\theta$. Not only does this reduce the dimensionality of the input space but also prevents the AIP from learning shortcuts (Geirhos et al., 2020) resulting from spurious correlations that are not stable under policies other than π_0[7]. Provided we have the DBN of the problem, there are many algorithms available that can help us find a minimal d-set (Acid and De Campos, 1996; Tian et al., 1998). If not, some domain knowledge may be sufficient in most cases, to remove a few variables from l_t and prevent confounding (Suau et al., 2022b).

Working in the joint space (d_t, u_t) instead of (l_t, u_t) has a another positive effect on off-policy generalization. Since l_t includes the agents actions the distribution $P^{\pi_0}(l, u)$ might be arbitrarily far from a distribution in $\{P^\pi(l, u)\}_{\pi \in \Pi}$. This is problematic because our low-capacity influence predictor may be unable to generalize well under large distribution shifts.

Proposition 3.3.1. *Let π_0 be the exploratory policy used to collect the dataset. Then, for all $\pi \in \Pi$, $u_t \in \times_i \text{dom}(U_t^i)$, $l_t \in \times_i \text{dom}(L_t^i)$*

$$KL(P^{\pi_0}(d_t, u_t)||P^\pi(d_t, u_t)) \leq KL(P^{\pi_0}(l_t, u_t)||P^\pi(l_t, u_t))$$

where KL is the Kullback–Leibler divergence.

Proof. Found in Appendix A.1.

The result above shows that the distributions of pairs (d^*, u) induced by two policies in Π are never further apart than their full AOHs counterparts. In fact, if actions and d-sets are only weakly-coupled the distributions may be very close. In the extreme case, if d-sets are causally independent with respect to the agent's actions, the influence sources can be considered exogenous (Boutilier et al., 1996; Chitnis and Lozano-Pérez, 2020) and the two distributions are equivalent.

Corollary 3.3.3. *Let $a_{0:t}$ be the past sequence of actions from 0 to t. If $P(d_t|do(a_{o:t})) = P(d_t)$ for all d_t and $a_{0:t}$. Then, for any $\pi_1 \neq \pi_2 : \pi_1, \pi_2 \in \Pi$ we have that, for all*

$u_t \in \times_i \text{dom}(U_t^i)$, $l_t \in \times_i \text{dom}(L_t^i)$, $P^{\pi_1}(d_t, u_t) = P^{\pi_2}(d_t, u_t)$ *even though* $P^{\pi_1}(l_t, u_t) \neq P^{\pi_2}(l_t, u_t)$.

Proof. Found in Appendix A.1.

Of course, in some domains, certain instantiations of the influence sources may only occur when very specific action sequences are followed. In such cases, a finite dataset collected with a random uniform policy will not be sufficient to fully cover the joint space of influence sources and d-sets, and thus the AIP may be unable to generalize well. If this happens, assuming a GS is available during training, the obvious solution would be to retrain the AIP every certain number of policy updates. Nonetheless, as we explain in the next section, in many cases, it might not be worth paying the computational cost associated to this solution since inaccurate simulators might still provide good enough experiences to learn from.

3.3.3 Sufficiently Similar Training Conditions

Up to this point we have discussed the importance of training accurate AIPs that are also invariant to distribution shifts caused by changes in the agent's policy. Nonetheless, here we pose a different question: to what extent is it possible to achieve near-optimal performance with inaccurate AIPs? As a matter of fact, it is very common in real-life engineering to model individual components in isolation without considering whether they belong to a larger system. Here we will fall short of giving a complete answer but we will make some observations to suggest that agents might not always require the most accurate AIPs. A view that is also supported by our experiments.

First, if the IALS can produce at least a few observation sequences that are similar to the ones generated by the true environment, an agent with sufficient capacity will be able to learn from those, and perform well in the real world. Given that one of the premises of our method is the need for accurate LS, even if the influence predictor was entirely random, the chances of getting just a small percentage of useful experiences may be in fact quite high. Yet, as discussed in Section 3.3, different frequencies in some of the transitions might change the value of certain ALSHs and in turn affect the agent's policy. This will occur if some of the unrealistic trajectories generated by an inaccurate AIP overlap partly with the realistic ones. Our second argument, is that different influence distributions may still lead to the same, or very similar, optimal policy (Becker et al., 2003). That is, even if the agent

is trained on an inaccurate IALS, some of the strategies that the agent will develop might be transferable and could also be useful when followed under real trajectories (Lazaric, 2012).

3.4 Experiments

The goal of the experiments is to: (1) Study whether we can reduce training times by replacing the GS with the IALS. (2) Measure the impact of the AIP accuracy on the learning process and the agent's final performance. (3) Investigate the memory needs of the AIP when the agent's memory is finite. The experiments are conducted on two benchmark domains: traffic control and warehouse commissioning. While these environments have been designed to emulate real-world challenges, it is essential to emphasize that they only bear a limited resemblance to the complexities and nuances inherent in realistic scenarios (Geroliminis and Daganzo, 2008; Knoop et al., 2015). Hence, the results obtained should not be automatically extrapolated to the real world. For a comprehensive review of strategies for real road traffic control, the interested reader is referred to Papageorgiou et al. (2003); Qadri et al. (2020).

3.4.1 Experimental Setup

Agents are trained separately with PPO (Schulman et al., 2017) on (1) the global simulator (GS), (2) the influence-augmented local simulator (IALS) with an AIP trained offline on a dataset collected from the GS, (3) an IALS that uses an untrained AIP to make predictions (untrained-IALS). To measure the agent's performance, training is interleaved with periodic evaluations on the GS. The results are averaged over 5 runs with different random seeds.

3.4.2 Traffic Control

Figure 3.2 shows a grid-like traffic network composed of 25 intersections. The agent controls the traffic lights at one of the intersections only. The rest of the traffic lights are controlled by fixed actuators that use sensors to adapt to the traffic. The policies used in this experiment for the actuated traffic light controllers were extensively optimized by Wu et al. (2017). We train agents separately on the two intersections highlighted in Figure

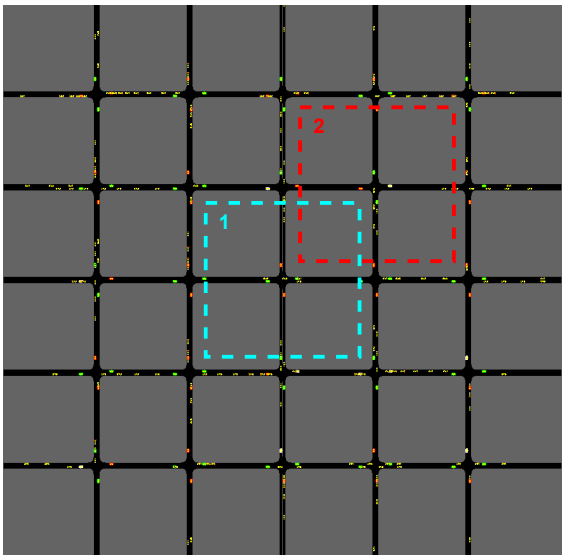


Figure 3.2: A screenshot of the traffic environment. We train agents separately on the two intersections highlighted by the blue and the red dashed boxes. The rest of the intersections are controlled by fixed actuators that use sensors to adapt to the traffic. The goal is to maximize the average speed of cars within the intersection. The agent can only observe cars inside the dashed boxes.

3.2. The goal is to maximize the average speed of cars within the intersection. The agent can only observe cars inside the dashed boxes.

GS, LS, AIP, and D-set

The GS and LS are built using Flow (Wu et al., 2017) and SUMO (Lopez et al., 2018). The GS simulates the entire traffic grid (Figure 3.2) while the LS only models the local neighborhood of the intersection being trained (Figure A.3 in Appendix A.4). The influence sources u_t are binary variables indicating whether or not a car will be entering the simulation from each of the four incoming lanes at the current timestep. The AIP $\hat{I}_\theta$ is a feedforward neural network that we train offline on a dataset of (d_t, u_t) pairs collected from the GS. The d-set d_t is a length 37 binary vector encoding the location of cars along the four incoming lanes. Traffic light information is not included in d_t to prevent confounding (Section 3.3.2).[8] Since the two intersections highlighted in Figure 3.2 are

[8] A visual example of a spurious correlation appearing in the traffic problem is given in Appendix A.2.

influenced differently by the rest of the traffic network we train separate AIPs for each of them.

Results

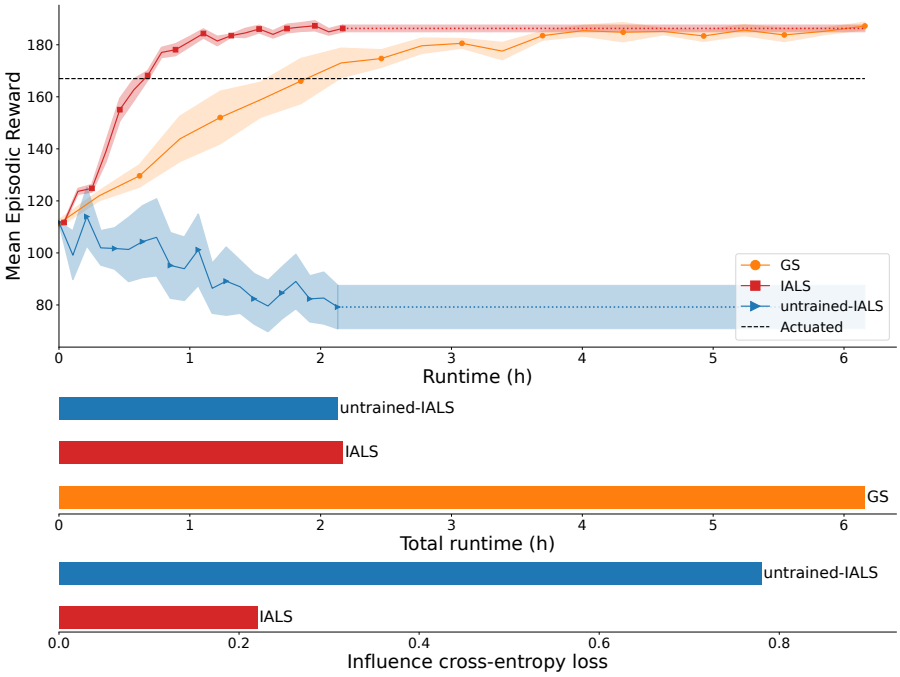


Figure 3.3: **Top:** Learning curves of agents trained with the GS, the IALS and the untrained-IALS on intersection 1 (Figure 3.2) as a function of wall-clock time. The dotted horizontal lines show the final performance of the agents after 2M timesteps of training. **Middle:** Total runtime of training for 2M training steps on the three simulators. **Bottom:** Cross entropy loss for the trained and untrained AIPs.

The plot at the top of Figure 3.3 are the learning curves of agents trained with the GS, the IALS, and the untrained-IALS to control the traffic lights at intersection 1 (Figure 3.2).[9] The plot shows the mean episodic reward as a function of real wall-clock time, which for IALS includes the time for data collection and the AIP training time. Agents are trained for 2M timesteps on all three simulators. The dotted horizontal lines at the end of the red and blue curves show the agent's final performance. The black horizontal line indicates the performance of the actuated traffic light controller. The two bar charts at the bottom show the total training time when using each of the three simulators, and

[9]Results for intersection 2 are provided in Appendix A.5.1.

the AIP's accuracy with and without training. The results suggest that policies trained on the IALS (red) can match the performance of those trained on the GS (orange) in about 1/3 of the total training time, despite the IALS is not as accurate as the GS. This is in line with our hypothesis in Section 3.3.3. Similar influence distributions, $I(u_t|l_t) \approx \hat{I}_\theta(u_t|l_t)$, may lead to the same, or very similar, optimal policy. More experiments exploring this phenomenon are provided in Appendix A.5.3. In contrast, since the distribution $P^\pi(l_t, u_t)$ induced by the untrained AIP is very different from the true distribution, as evidenced by the high cross entropy loss (blue bar bottom chart), agents trained on the untrained-IALS (blue) perform much worse. A table with a breakdown of the runtimes is included in Appendix A.3. A comparison of the mean episodic reward as a function of the number of timesteps is provided in Appendix A.5.2.

3.4.3 Warehouse Commissioning

A team of 36 robots (blue and purple) need to fetch the items (yellow) that appear with probability 0.02 on the shelves (black dashed lines) of the warehouse in Figure 3.4. Each robot has been designated a 5×5 square region and can only collect the items that appear on the shelves at the edges. The regions overlap so that each of the 4 item shelves in a robot's region is shared with one of its 4 neighbors. The blue robots have been programmed to go for the oldest item in their region. The purple robot that is inside the region highlighted by the red box, still needs to be trained. This robot receives as observations a bitmap encoding its own location and a set of 12 binary variables that indicate whether or not each of the 12 items within its region needs to be collected. The purple robot, however, cannot see the location of the other robots even though all of them are directly or indirectly influencing it through their actions.

GS, LS, AIP, and D-set

The GS simulates the entire warehouse (Figure 3.4), while the LS models only the 5×5 square region delimited by the red box (Figure A.3 in Appendix A.4). The influence sources u_t encode the location of the four neighbor robots. The AIP is a GRU (Cho et al., 2014) that we train offline on a dataset of (d_t, u_t) pairs collected from the GS. If the AIP predicts that any of the neighbor robots is at one of the 12 cells within the red box and there is an active item on that cell, that item is removed and the purple robot can no longer collect it. The d-set d_t includes the history of the 12 item variables and

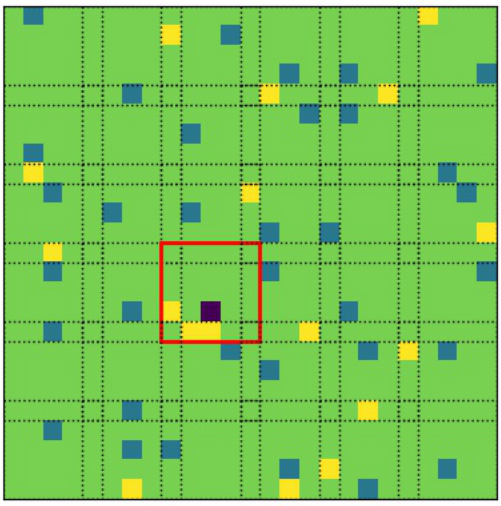


Figure 3.4: A screenshot of the warehouse environment. The robots (blue and purple) need to fetch the items (yellow) that appear on the shelves (black dashed lines). Each robot can only collect the items that appear on their designated region. We train the purple robot inside the red box. The blue robots have been programmed to go for the oldest item in their region. The purple robot only receives information about its own location and what items need to be collected. However, cannot see the location of the other robots.

12 additional binary variables encoding whether or not the controlled robot was (is) at one of the item locations. The latter variables are meant to differentiate between an item that is gone because the controlled robot collected it from an item that was picked up by the neighbor robots. The rest of variables in l_t (i.e. the robot's history of locations) are unnecessary for predicting u_t, and thus susceptible of becoming confounders (Section 3.3.2).

Results

The plot at the top of Figure 3.5 shows the learning curves of the warehouse robot as a function of real wall-clock time, which for IALS includes the time for data collection and the AIP training time. Agents are trained for 2M timesteps on all three simulators. The dotted horizontal lines at the end of the red and blue curves show the agent's final performance. The short horizontal line at the beginning of the red curve represents the time for data collection and the AIP's training time. The two bar charts at the bottom show the total training time when using each of the three simulators, and the AIP's

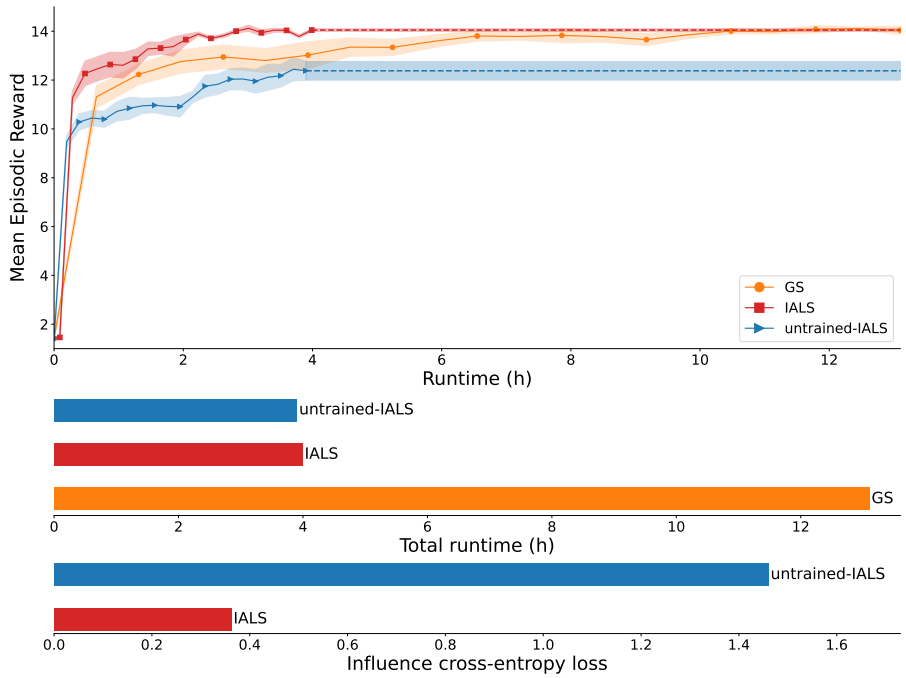


Figure 3.5: **Top:** Learning curves of agents trained with the GS, the IALS and the untrained-IALS on the the warehouse environment as a function of wall-clock time. The dotted horizontal lines at the end of the red and blue curves show the final performance of the agents after 2M timesteps of training. **Middle:** Total runtime of training for 2M steps on the three simulators. **Bottom:** Cross entropy loss for the trained and untrained AIPs.

accuracy with and without training. Again, we see that robots trained on the IALS (red) are able to reach the same performance as those trained on the GS (orange) in about 1/3 of the total training time despite the IALS is only approximate. Moreover, robots trained on the untrained-IALS (blue) perform reasonable well on the GS. Although the frequency at which items disappear with the untrained-IALS differs very much from that of the true environment, the basic strategy on how to collect items can still be learned. These results further confirm our hypothesis that inaccurate simulators may, in some cases, render effective policies (Section 3.3.3). More experiments experiments exploring this phenomenon are provided in Appendix A.5.3. A table with a breakdown of the runtimes is included in Appendix A.3. The learning curves as a function of the number of timesteps are provided in Appendix A.5.2.

3.4.4 Finite Memory Agents and AIP History Dependence

Here we investigate whether our theoretical result from Theorem 3.3.1 also holds in practice. We want to show that when the agent's memory is finite, meaning it can only access observations from k timesteps in the past, an influence predictor which conditions on the same history length is sufficient. We test this on the warehouse domain. To make the need for memory more evident, we modify the environment so that items always disappear from the robot's region after exactly 8 timesteps. We first train, AIPs with and without memory. We call the resulting simulators M-IALS and NM-IALS respectively. A histogram showing for how long items are active before disappearing under each of the simulators is shown in Figure 3.6. As expected, while the former can reach an accuracy of 100% (items always remain for 8 timesteps with the M-IALS), the spectrum is much wider for the NM-IALS. This is because the latter can only estimate the marginal distribution $P^{\pi}(u_t|o_t)$. Then, we train agents with (M) and without memory (NM) on the M-IALS and the NM-IALS. The results for all four combinations are shown in Figure 3.6. As indicated by the theory, the plot shows that when agents have no memory AIPs may condition only on the current observation (red). In such cases, the extra level of detail that a more accurate history-dependent AIP can provide is wasted (green). In contrast, when agents can distinguish observations from one another based on their memories from the past, AIPs that make predictions by looking at the history are fundamental. This is evidenced by the gap between the blue and orange curves. It is worth to point out that, even though the memory agent performs worse when trained on the NM-IALS (orange) than when trained on the M-IALS (blue), it can still outperform the agents with no memory (red and green). We posit that this is because, with the NM-IALS items still disappear after 8 timesteps *on average* (see dashed vertical line in the bottom left histogram in Figure 3.6), which allows the memory agent to learn to not go for an item if this has been active for a long time. This once again suggests that, in some cases, inaccurate influence predictors might still provide good enough experiences to learn from.

3.5 Conclusion

This chapter has offered a practical solution to allow the application of Deep RL methods to large systems, where performing exhaustive simulations can not be afforded. We focused on domains where, although the agent only interacts with a local portion of the environment, it is influenced by the global dynamics. The main idea was to replace the

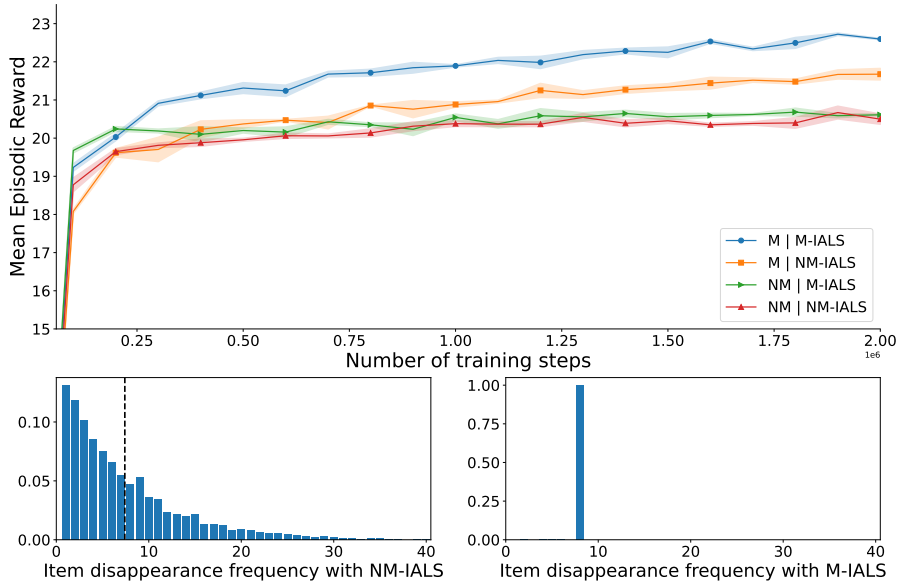


Figure 3.6: **Top:** Learning curves of agents with (M) and without memory (NM) trained on M-IALS and NM-IALS. **Bottom:** item disappearance frequencies with NM-IALS and with M-IALS.

computationally inefficient global simulator by a lightweight version that only models the agent's local problem. However, as we showed in our experiments, directly doing this sometimes translates in a distribution shift on the agent's experience that yields poor performing policies. A good simulator needs to account for the interactions between the local region and the global dynamics. The results of our experiments suggested that by combining a pretrained influence predictor with the local simulator, we could speed up the learning process considerably while matching the performance of agents trained on the global simulator. Moreover, we analyzed the consequences of training influence predictors on data distributions that are different from those the predictor sees when deployed and resolved that, when possible, the human designer should remove from the input those variables that are irrelevant for predicting the influence sources. Finally, in line with the results in Section 3.3.1, the last experiment revealed that the agent's memory capacity limits the memory needs of the influence predictor.

Future work may explore how to train multiple agents using independent IALS. This could lead to even further speedups since agents could train on separate simulators running in parallel. However, having agents learning simultaneously would imply that the influence

distributions would no longer be stationary since a change in any of the agents' policies could alter the other agents' local dynamics. Hence the method would need to be reworked such that the AIPs can handle moving targets. Another direction for future research is to study how to deal with environments where coverage of untrained and trained policies is different. That is, environments where random policies are unlikely to reach certain ALSHs. If the AIPs cannot generalize to the new data points, a solution would be to retrain the AIPs when the agent starts visiting new ALSHs. Nonetheless, paying the computational cost associated to this solution might not be necessary in many cases since, as we discuss in section 3.3.3, slightly inaccurate simulators might be sufficient to train good performing policies.

Chapter 4

Distributed Influence-Augmented Local Simulators

Truth... is much too complicated to allow for anything but approximations.

–John von Neumann, The Mathematician

Due to its high sample complexity, simulation is, as of today, critical for the successful application of reinforcement learning. Many real-world problems, however, exhibit overly complex dynamics, making their full-scale simulation computationally slow. In this chapter, we show how to factorize large networked systems of many agents into multiple local regions such that we can build separate simulators that run independently and in parallel. To monitor the influence that the different local regions exert on one another, each of these simulators is equipped with a learned model that is periodically trained on real trajectories. Our empirical results reveal that distributing the simulation among different processes not only makes it possible to train large multi-agent systems in just a few hours but also helps mitigate the negative effects of simultaneous learning.[1,2]

[1]This chapter is based on Suau et al. (2022a).

[2]Source code is available at `https://github.com/INFLUENCEorg/DIALS`.

4.1 Introduction

Imagine we have to train a team of agents to control the traffic lights of a very large city, so large that we simply cannot control all traffic lights using a single policy. The first step would be to split the problem into multiple sub-regions. A natural division would be to assign one traffic light to each agent. Then, since the agents act locally, we would limit their observations to contain only local information. This partial observability could affect their optimal policies but would also make each individual decision-making problem more manageable (McCallum, 1995b; Dearden and Boutilier, 1997). Moreover, we may also want to reward agents only for what occurs in their local neighborhood such that we reduce the variance of the returns (Spooner et al., 2021) and facilitate credit assignment (Castellini et al., 2020). Finally, we could train all agents together on a big traffic simulator that reproduces the global dynamics. However, if the city is truly large, it could take weeks or even months to optimize their policies. That is assuming training actually converges.

One may argue that, since the agents' observations and rewards are local, we could as well train them on separate simulators that model only the local transition dynamics (i.e. cars moving within each of the sub-regions; van der Pol and Oliehoek 2016). This approach might work if the agents' local transitions are isolated from the rest of the system (Becker et al., 2003), but would probably break when the local regions are coupled. This is because the local simulators would fail to account for the fact that the agents' local regions belong to a larger system and depend on one another. A solution is to model the influence the global system exerts on each local region. In many scenarios, such as in the traffic problem, even though the local regions may be affected by many external variables (e.g.traffic densities in other parts of the city), they are only directly influenced by a small subset of them (e.g., road segments that connect the intersections with the rest of the city). This subset of variables is known as the influence sources. The theoretical framework of Influence-Based Abstraction (Oliehoek et al., 2021) shows that by monitoring the posterior distribution of the influence sources given the action local state history (ALSH), one can simulate realistic trajectories that match those produced by the global simulator. The resulting simulator, known as the influence-augmented local simulator (IALS), has been proven effective in single agent scenarios when combined with planning (He et al., 2020) and reinforcement learning (RL) algorithms (Chapter 3; Suau et al., 2022d).

In this chapter, we extend the IBA framework to multi-agent domains. We show how to factorize large networked systems, such as the previous traffic example, into multiple

sub-regions so that we can replace the global simulator (GS) with a distributed network of IALSs that can run independently and in parallel. There is one important caveat to this. The IBA framework assumes only a single agent is learning at a time. This assumption is needed to make the influence distributions stationary. This implies that in our case, since we want the agents to learn simultaneously, previously computed influence distributions would no longer be valid after the agents update their policies. The naive solution would be to recompute new influence distributions every time any agent updates its policy. However, we argue that this is not only impractical, since recomputing the distributions is not without costs, but also undesirable. The theoretical results in Section 4.4.1 demonstrate that multiple (similar) joint policies may induce the same influence distributions and that even when they vary a little, they can still elicit the same optimal policies. Further, our insights in Section 4.4.3 hint that what seems to be a problem at first, may in fact be an advantage since in many situations, maintaining the previous influence distributions implies that the local transitions, although biased, remain stationary.

Contributions The main contributions of this chapter are: (1) adapting IBA to multi-agent reinforcement learning (MARL),[3] and demonstrating that simultaneous learning is possible without incurring major computational costs, (2) showing that by distributing the simulation among different processes, we can parallelize training and scale up to systems with many agents, (3) revealing that the non-stationarity issues inherent to MARL are partly mitigated as a result of this training scheme.

4.2 Related Work

A few prior works have investigated the computational benefits of factorizing large systems into independent local regions (Nair et al., 2005; Varakantham et al., 2007; Kumar et al., 2011; Witwicki and Durfee, 2011). Unfortunately, since local regions are often coupled to one another, such factorizations are not always appropriate. Nonetheless, in many cases, the interactions between regions occur through a limited number of variables. Using this property, the theoretical work by Oliehoek et al. (2021) on influence-based abstraction (IBA) describes how to build influence-augmented local simulators (IALS) of local-POMDPs, which model only the variables in the environment that are directly

[3]Although the original IBA formulation (Oliehoek et al., 2012) is already framed as multi-agent, it assumes agents learn one at a time while the other agents' policies are fixed.

relevant to the agent while monitoring the response of the rest of the system with the influence predictor. The problem is that the exact computation of the conditional influence distribution is intractable, and we can only try to estimate it from data. Congeduti et al. (2021) provide theoretical bounds on the value loss when planning with approximate influence predictors. The work by He et al. (2020) has empirically demonstrated the advantage of this approach to improve the efficiency of online planning in two discrete toy problems. Suau et al. (2022d) scale the method to high-dimensional problems by integrating the IBA framework with single-agent RL showing that the IALS can train policies much faster than the GS. In this chapter, we extend the IBA solution to MARL and explain how to build a network of independent IALS such that we can train agents in parallel.

One of the consequences of training agents on independent simulators is that the non-stationarity issues arising from having the agents learn simultaneously are partly mitigated. There is a sizeable body of literature that concentrates on this issue (Hernandez-Leal et al., 2017), we include a review of these works in Appendix B.2 for completeness. However, we note that our main purpose is to scale MARL up to systems with many agents. Hence, we are not concerned here with comparing our method with those that exclusively target non-stationarity, especially given that, for scalability reasons, these cannot be applied to the high-dimensional problems we consider here.

4.3 Preliminaries

4.3.1 Problem Formulation

The type of problems we describe in the introduction can be formulated as factored partially observable stochastic games (Hansen et al., 2004), which are defined as follows.

Definition 4.3.1 (FPOSG)**.** A factored partially observable stochastic game (FPOSG) is a tuple $\langle N, \mathcal{S}, S, \mathcal{A}, T, \{R^i\}, \Omega, \{O^i\}\rangle$ where $N = \{1, ..., n\}$ is the set of n agents, $\mathcal{S}$ is the set of states, S is the set of state variables $S = \{S^1, ..., S^{|S|}\}$, such that every state $s_t \in \times_j \text{dom}(S^j) = \mathcal{S}$ is a vector $s_t = \langle s_t^1, ..., s_t^{|S|}\rangle$, $\mathcal{A} = \times_{i\in N}\mathcal{A}^i$ is the set of joint actions $a_t = \langle a_t^1, ..., a_t^n\rangle$, with $\mathcal{A}^i$ being the set of actions for agent i, $T : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition function, which determines the probability $T(s_{t+1}|s_t, a_t)$ of transitioning to s_{t+1} given s_t and the joint action a_t, $R^i : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function for agent i, with $R^i(s_t, a_t)$ being the reward that agent i receives for taking the joint action a_t in s_t

, $\Omega = \times_{i \in N} \Omega^i$ is the set of joint observations $o_t^i = \langle o_t^1, ..., o_t^n \rangle$, with Ω^i being the set of observations for agent i, and O^i is the observation function for agent i, which determines the probability $O^i(o_t^i | s_t)$ that agent i observes o_t^i given s_t.

Solving the FPOSG implies finding the policy π^i for each agent i that maximizes the expected Q-value (Equation 2.1). However, agents receive only partial observations o^i of the true state s, which are not necessarily Markovian. Therefore, optimal policies are, in general, history-dependent in a POSG. Hence, we define the agents' policies $\pi^i(a_t^i | h_t^i)$ as mappings from action-observation histories (AOH), $h_t^i = \langle o_1^i, a_1^i, ..., a_{t-1}^i, o_t^i \rangle$, to probability distributions over actions, such that agent i's optimal policy π^{i*} is the one that for every AOH h_t^i selects the action with the highest Q-value.

Given the structural assumptions we made in the introduction about the agents' only being able to observe and be rewarded for what occurs in their local neighborhood, we can narrow down the problem formulation and work with a specific class of FPOSGs called local-FPOSGs (Oliehoek et al., 2021), which better encompass the problems we consider here.

Definition 4.3.2 (Local-FPOSG)**.** A local-FPOSG is an FPOSG where, for all $i \in N$, O^i and R^i depend only on agent i's actions A_i and a local subset of state variables $X^i = \{X^{i,1}, ..., X^{i,m}\} \subseteq X$, known as the local state variables. Each combination of these variables determines a different local state for agent i, $x_t^i = \langle x_t^{i,1}, ..., x_t^{i,|X|} \rangle \in \times_j \text{dom}(X_t^{i,j})$. Hence, for all $i \in N$, $s_t \in \mathcal{S}$, $o^i \in \Omega^i$, and $a \in \mathcal{A}$, we can write $O^i(o_t^i | s_t) = \dot{O}^i(o_t^i | x_t^i)$ and $R^i(s_t, a_t) = \dot{R}^i(x_t^i, a_t^i)$, where $\dot{O}^i$ and $\dot{R}^i$ are the local observation and reward functions for agent i. Further, we use Y^i to denote agent i's set of non-local state variables, $Y^i = X^i \setminus F$.

It is easy to show that, from the perspective of agent i, if the policies of all other agents $-i$ are fixed, the local-FPOSG is just a local-FPOMDP (Definition 2.3.1). Hence, just as we did with the local-FPOMDP in Section 2.3.3, we can factorize the above model into multiple influence-augmented local models, one for each agent.

Definition 4.3.3 (IALM)**.** An influence-augmented local Model (IALM) for agent i is a local-FPOMDP with local transition function $\dot{T} : \times_j \text{dom}(X^{i,j}) \times \mathcal{A}^i \to \Delta(\times_j \text{dom}(X^{i,j}))$, which determines the probability $T^i(x_{t+1}^i | x_t^i, u_t^i, a_t^i)$ of transitioning to x_{t+1}^i given x_t^i, a_t^i and the influence sources u_t^i. Further, we define agent i's influence distribution I which for all $u_t^i \in \times_j \text{dom}(U_t^{i,j})$, determines the probability $I^i(u_t^i | l_t^i)$ of u_t^i given agent i's action local state history (ALSH) $l_t^i = \langle x_0^i, a_0^i, ..., a_{t-1}^i, x_t^i \rangle$. Such that, using $\dot{T}$ and I, the probability

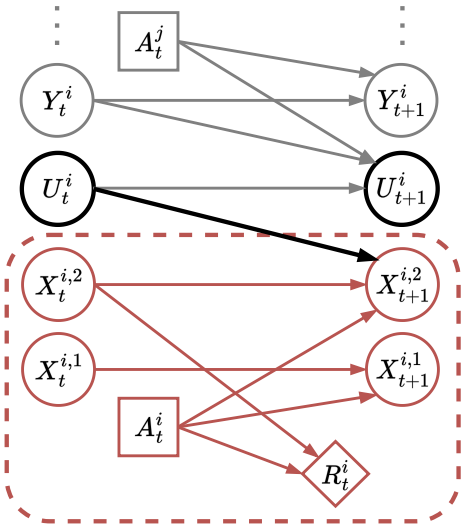


Figure 4.1: **Left:** A Dynamic Bayesian Network showing agent i's transition dynamics in a local-form FPOSG prototype. **Right:** A conceptual diagram of the IALS.

of agent i's next local state x_{t+1}^i can be computed as

$$\Pr(x_{t+1}^i \mid l_t^i, a_t^i) = \sum_{u_t^i} \dot{T}^i(x_{t+1}^i \mid x_t^i, u_t^i, a_t^i) I^i(u_t^i \mid l_t^i). \tag{4.1}$$

4.3.2 Influence-Augmented Local Simulators

Here we briefly describe how the IALM formulation can be used in practice to build the IALS (Suau et al., 2022d), which consists of a *local simulator* and an *approximate influence predictor*.

Local simulator (LS): The LS is an abstracted version of the environment that only models a small portion of it. As opposed to a global simulator (GS), which should closely reproduce the dynamics of every state variable, the LS focuses on characterizing the transitions of those variables X^i that agent i directly interacts with, $\dot{T}^i(x_{t+1}^i | x_t^i, u_t^i, a_t^i)$.

Approximate influence predictor (AIP): The AIP monitors the interactions between agent i's local region X^i, the external variables Y^i, and the other agents' actions A_{-i}, by estimating $I^i(u_t^i | l_t^i)$. Since, due to combinatorial explosion, computing the exact probability

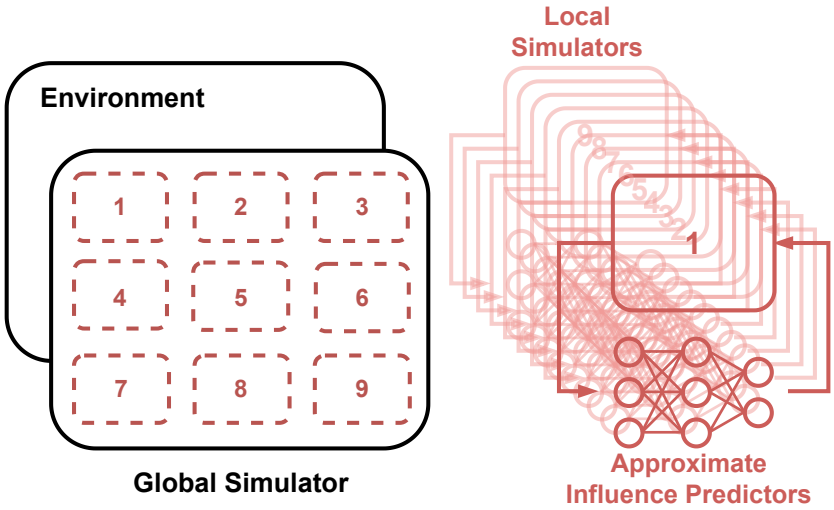


Figure 4.2: A conceptual diagram of the DIALS

$I^i(u_t^i|l_t^i)$ is generally intractable (Oliehoek et al., 2021), a neural network is used instead to approximate the influence distribution. Thus, we write $\hat{I}_{\theta^i}$ to denote agent i's AIP, where θ^i are the network parameters. The AIP $\hat{I}_{\theta^i}$ is trained on a dataset D^i of N samples of the form (l_t^i, u_t^i) collected from the GS. Since the role of the AIP is to estimate the conditional probability of the influence sources u_t^i given the past ALSH, we can formulate the task as a classification problem and optimize the network using the expected cross-entropy loss (Bishop, 2006).

4.4 Distributed Influence-Augmented Local Simulators

As mentioned in the previous section, local-form FPOSGs are solved iteratively in the IBA framework. This means that only a single agent can update its policy at a time. Here, we relax this assumption and discuss the advantages and disadvantages of simultaneous learning. Proofs for all the theoretical results in this section can be found in Appendix B.1.

4.4.1 Enabling Parallelization

The main reason to disallow simultaneous learning is that changes in the other agents' policies can affect agent i's influence distribution $I^i(u_t^i|l_t^i)$, which may become non-stationary.

This renders previously computed influences useless because they no longer capture the true response of the global system.

This restriction, however, prevents IBA from unlocking its full potential. The fact that each agent's IALS is independent of the others means that the computations can be distributed among different processes that can run in parallel. Hence, putting aside the non-stationarity issue, and assuming no overhead costs in spawning an increasing number of processes, the total runtime of the method would stay constant if the dimensionality of the global system grew, either because the number of non-local variables or the number of agents increased. This is in contrast to having agents learn simultaneously in the same GS, in which case larger environments imply longer runtimes. Moreover, since each IALS simulates only a portion of the environment the total amount of memory space needed would be split among the different processors. Hence, we could run the simulation on multiple machines with small memory rather than one big machine with very large memory.

In principle, one could prevent the AIPs from becoming stale by simply updating all $\{\hat{I}_{\theta^i}(u_t^i|l_t^i)\}_{i\in N}$ every time any of the other agents changes its policy. However, this creates a difficult moving target problem and makes the whole method very inefficient since, especially in deep RL, policies are updated very frequently. Fortunately, as we argue in the following, in many cases, paying the extra cost of retraining the AIPs is neither necessary nor desirable.

Multiple Joint Policies May Induce the Same Influence Distribution

In the following, we show that multiple joint policies may often map onto the same influence distribution $I^i(u_t^i|l_t^i) \in \Psi^i$ for agent $i \in N$.

Lemma 4.4.1. *Let $\Pi = \times_{i\in N}\Pi^i$ be the product space of joint policies with Π^i being the set of policies for agent i. Moreover, let $\Psi = \times_{i\in N}\Psi^i$ be the product space of joint influences, with Ψ^i being the set of influence distributions for agent i. Every joint policy $\pi \in \Pi$ induces exactly one influence distribution $I^i \in \Psi^i$ for every agent $i \in N$.*

Proposition 4.4.1. *The space of joint policies $\Pi = \times_{i\in N}\Pi^i$ is necessarily greater than or equal to the space of joint influences $\Psi = \times_{i\in N}\Psi^i$, $|\Pi| \geq |\Psi|$. Moreover, there exist local-form FPOSGs for which the inequality is strict.*

The advantages of this result were shown empirically by Witwicki and Durfee (2010b), who demonstrated that planning times can be reduced by searching the space of joint influences rather than the space of joint policies, which is often much larger. In fact, in the extreme case of local transition independence (Becker et al., 2003),[4] we have that for all joint policies π there is a single $\{I^i\}_{i \in N}$

Corollary 4.4.2. *Let agent i's influence sources u_t^i be independent of the other agents' actions a^{-i}. Then, for any joint policy $\pi \in \Pi$, there is a unique influence distribution $I^{i*} \in \Psi^i$ for every agent $i \in N$ and $|\Pi| \gg |\Psi| = 1$.*

The result above implies that, in this particular case, we would only need to train the AIPs once at the beginning. Although we do not expect the situation in Corollary 4.4.2 to be the norm, we do believe that in many scenarios, such as in the two environments we explore here, we would not need to retrain the AIPs very often because similar joint policies will influence the local regions in very similar, if not in the same, ways. Moreover, the next result shows that even when this is not the case, an outdated I^i computed from an old joint policy might still produce the same optimal policy for agent i.

Multiple Influence Distributions May Induce the Same Optimal Policy

We use the simulation lemma (Kearns and Singh, 2002) to prove that if two influence distributions are similar enough they will induce the same optimal policy.

Lemma 4.4.3. *Let M_1^i and M_2^i be two IALMS differing only on their influence distributions $I_1^i(u_t^i|l_t^i)$ and $I_2^i(u_t^i|l_t^i)$. Let $Q_{M_1^i}^{\pi^i}$ and $Q_{M_2^i}^{\pi^i}$ be the value functions induced by M_1^i and M_2^i for the same π^i. If, for all $h_t^i \in H_t^i$, $u_t^i \in \times_j \mathrm{dom}(U_t^{i,j})$, I_1^i and I_2^i satisfy*

$$\sum_{l_t^i, u_t^i} P(l_t^i \mid h_t^i) \left| I_1^i(u_t^i \mid l_t^i) - I_2^i(u_t^i | l_t^i) \right| \leq \xi, \tag{4.2}$$

then

$$\left| Q_{M_1^i}^{\pi^i}(h_t^i, a_t^i) - Q_{M_2^i}^{\pi^i}(h_t^i, a_t^i) \right| \leq \bar{R} \frac{(K-t)(K-t+1)}{2} \xi \tag{4.3}$$

for all $\pi^i \in \Pi^i$, $h_t^i \in H_t^i$, and $a_t^i \in \mathcal{A}^i$, where K is the horizon and $\bar{R} = ||R||_\infty$

Intuitively, Lemma 4.4.3 shows that the difference in value between M_1^i and M_2^i is upper-bounded by the maximum difference between I_1^i and I_2^i times a constant. Actually, if the

[4] As opposed to IBA, Becker et al. (2003) assume agents are tied by a shared global reward.

action-gap (Farahmand, 2011) (i.e. value difference between the best and the second best action) in one of the IALMs is larger than twice the difference between the $Q^{\pi^i}_{M_1}$ and $Q^{\pi^i}_{M_2}$ the IALMs share the same optimal policy.

Theorem 4.4.4. *Let M^i_1 and M^i_2 be two IALMS differing only on their influence distributions $I^i_1(u^i_t|l^i_t)$ and $I^i_2(u^i_t|l^i_t)$. M^i_1 and M^i_2 induce the same optimal policy π^{i*} if, for some Δ,*

$$Q^{\pi^{i*}}_{M^i_1}(h^i_t, \bar{a}^i_t) - Q^{\pi^{i*}}_{M^i_1}(h^i_t, \hat{a}^i_t) > 2\Delta \quad \forall h^i_t, \hat{a}^i_t \neq \bar{a}^i_t \tag{4.4}$$

with

$$\left|Q^{\pi^i}_{M^i_1}(h^i_t, a^i_t) - Q^{\pi^i}_{M^i_2}(h^i_t, a^i_t)\right| \leq \Delta \quad \forall h^i_t, a^i_t, \pi^i, \tag{4.5}$$

where $\bar{a}^i_t = \arg\max_{a^i_t} Q^{\pi^{i*}}_{M^i_1}(h^i_t, a^i_t)$

Combining Lemma 4.4.3 and Theorem 4.4.4, we see that, because the difference in value between M^i_1 an M^i_2 depends on ξ (Lemma 4.4.3), the closer the distributions I^i_1 and I^i_2 are, the more likely it is that M^i_1 and M^i_2 share the same optimal policy. Note that we have no control over the action gap as it is domain-dependent. In some domains, the gap might be large and we can be more relaxed about not retraining the AIPs. In some others, the gap might be small and we may need to retrain the AIPs more frequently.

4.4.2 Algorithm

After the analysis above, we are now ready to present our method, which we call Distributed Influence-Augmented Local Simulators (DIALS). Algorithm 1 describes how we can train multi-agent systems with DIALS. As mentioned earlier, the key advantage of using DIALS is that simulations can be distributed among different processes, and thus training can be fully parallelized. This enables MARL to scale to very large systems with many learning agents. Moreover, following from our theoretical results, AIP training, which can also be done in parallel, is performed only every certain number of timesteps. The hyperparameter F in Algorithm 1 controls the AIPs' training frequency. The effect of F on the learning performance is empirically investigated in Section 4.5.

4.4.3 Mitigating the Negative Effects of Simultaneous Learning

The results in the previous section showed that the AIPs may not need to be retrained every single time the policies are updated, as the influence distributions may often stay

Algorithm 1 MARL with DIALS

Initialize policies $\{\pi^i\}_{i\in N}$ and AIPs $\{\hat{I}_{\theta^i}\}_{i\in N}$
repeat
  Collect datasets $\{D^i\}_{i\in N}$ from GS ▷ See Algorithm 4 in Appendix B.3
  in parallel, for $i \in N$ **do**
    Train AIP $\hat{I}_{\theta^i}$ on dataset D^i ▷ See Section 4.3.2
  end for
  in parallel, for $i \in N$ **do**
    for F steps **do** ▷ F is the AIPs' training frequency
      Simulate trajectories with IALS $\langle \dot{T}^i, \dot{R}^i, \dot{O}^i, \hat{I}_{\theta^i} \rangle$ ▷ See Algorithm 5 in Appendix B.3
      Train policy π^i ▷ Using any standard RL method
    end for
  end for
until end of training

the same or vary only a little. Yet, we now argue that, even when changes in the joint policy do affect the influence distributions $I^i(u_t^i|l_t^i)$ significantly, it may be advantageous not to retrain the AIPs.

First, we have already mentioned that when all agents learn simultaneously in the same simulator the transition dynamics often look non-stationary from the perspective of each individual agent. This may result in sudden performance drops caused by oscillations in the value targets (Claus and Boutilier, 1998). In contrast, when using independent IALS to train our agents, the transition dynamics remain stationary unless we update the AIPs. Hence, by not updating the AIPs too frequently, we get a biased but otherwise more consistent learning signal that the agents can rely on to improve their policies.

Second, we posit that the poor empirical convergence of many off-the-shelf Deep RL methods (Hernandez-Leal et al., 2017; Yu et al., 2021) is also because stochastic gradient descent updates often result in policies that perform worse than the previous ones. Thus, when learning together, agents may try to adapt to other agents' poor performing policies. These policies, however, are likely to be temporary as they are just a result of the inherent stochasticity of the learning process. Similarly, in many environments, agents shall take exploratory actions before they can improve their policies, which may also negatively impact cooperation if they learn simultaneously (Zhang et al., 2009, 2010). In our case, we can again benefit from the fact that the AIPs need to be purposely retrained, and do so only when the policies of the other agents have improved sufficiently.

Even though further theoretical analysis would be needed to be more conclusive about the

benefits of using independent simulators, the observations above give reasons to believe that what we initially described as a problem may in fact be an advantage. This view is also supported by our experiments.

4.5 Experiments

The goal of the experiments is to: (1) test whether we can reduce training times by replacing GS with DIALS, (2) investigate how the method scales to large environments with many learning agents, (3) evaluate the convergence benefits of using separate simulators to train agents rather than a single GS, and (4) study the effect of the AIPs' training frequency F on the agents' learning performance.

4.5.1 Experimental Setup

Agents are trained independently with PPO (Schulman et al., 2017)[5] on (1) the global simulator (GS), (2) distributed influence-augmented local simulators (DIALS) with AIPs trained periodically on datasets collected from the GS using the most recent joint policy, (3) DIALS with untrained AIPs (untrained-DIALS).

To measure the agent's performance, training is interleaved with periodic evaluations on the GS. The results are averaged over 10 random seeds on all except on the largest scenarios (10×10) for which, due to computational limitations we could only run 5 seeds. We report the mean return of all learning agents. We also compare the simulators in terms of total runtime. For DIALS this includes the agents' training time, the AIPs' training time, and the time for data collection.

4.5.2 Environments

Traffic control The first domain we consider is a multi-agent variant of the traffic control benchmark proposed by Vinitsky et al. (2018). In this scenario, agents are requested to manage the lights of a big traffic network. Each agent controls a single traffic light and can only observe cars when they are inside the intersection's local neighborhood. Their

[5]The vanilla PPO algorithm with decentralized value functions (independent PPO; IPPO) has been shown to perform exceptionally well on several multi-agent environments (de Witt et al., 2020; Yu et al., 2021).

goal is to maximize the average speed of cars within their respective intersections. To demonstrate the scalability of the method we evaluate DIALS on four different variants of the traffic network with 4, 25, 49, and even 100 intersections (agents). A screenshot of the traffic network with 25 intersections is shown in Appendix B.6. The GS and LS are built using Flow (MIT License) (Wu et al., 2017) and SUMO (Eclipse Public License Version 2) (Lopez et al., 2018). The GS simulates the entire traffic network while each LS models only the local neighborhood of each intersection $i \in N$ (Figure A.3 in Appendix B.6). We use the same LS for every agent-intersection but since, depending on where they are located, they are influenced differently by the rest of the traffic network, we have separate AIPs, $\{I_{\theta^i}\}_{i \in N}$, for each them. These are feedforward neural networks with the same architecture but different weights θ^i, trained periodically with frequency F on datasets $\{D^i\}_{i \in N}$ collected from the GS. The influence sources u_t^i are binary variables indicating whether or not a car will be entering from each of the four incoming lanes.

Warehouse Commissioning The second domain we consider is a warehouse commissioning task (Suau et al., 2022d). A team of robots (blue) needs to fetch the items (yellow) that appear with probability 0.02 on the shelves (dashed black lines) of the warehouse (see Figure B.5 in Appendix B.6). Each robot has been designated a 5×5 square region and can only collect the items that appear on the shelves at the edges. The regions overlap so that each of the 4 item shelves in a robot's region is shared with one of its 4 neighbors. The robots receive a reward between $[0, 1]$ when collecting an item. The exact value depends on how old the item is compared to the other items in their region. This is to encourage the robots to collect the oldest items first. The robots receive as observations a bitmap encoding their own location and a set of 12 binary variables that indicate whether or not a given item needs to be collected. The robots, however, cannot see the location of the other robots even though all of them are directly or indirectly influencing each other through their actions. We built four variants of the warehouse with 4, 25, 49, and 100 robots (agents). A screenshot of the warehouse with 25 robots is shown in Appendix B.6. The GS simulates the entire warehouse while the LS models only a 5×5 square region (Figure A.3 in Appendix B.6). We use the same LS for every robot (agent) but since depending on where they are located they are influenced differently by the rest of the robots, we have separate AIPs, $\{\hat{I}_{\theta^i}\}_{i \in N}$, for each of them. These are GRUs (Cho et al., 2014) with the same architecture but different weights θ^i, which we train periodically with frequency F on datasets $\{D^i\}_{i \in N}$ collected from the GS. Robot i's influence sources u_t^i encode the location of the four neighbor robots. If its AIP $\hat{I}_{\theta^i}$ predicts that any of the

neighbor robots is at one of the 12 cells within its region, and there is an active item on that cell, that item is removed and robot i can no longer collect it.

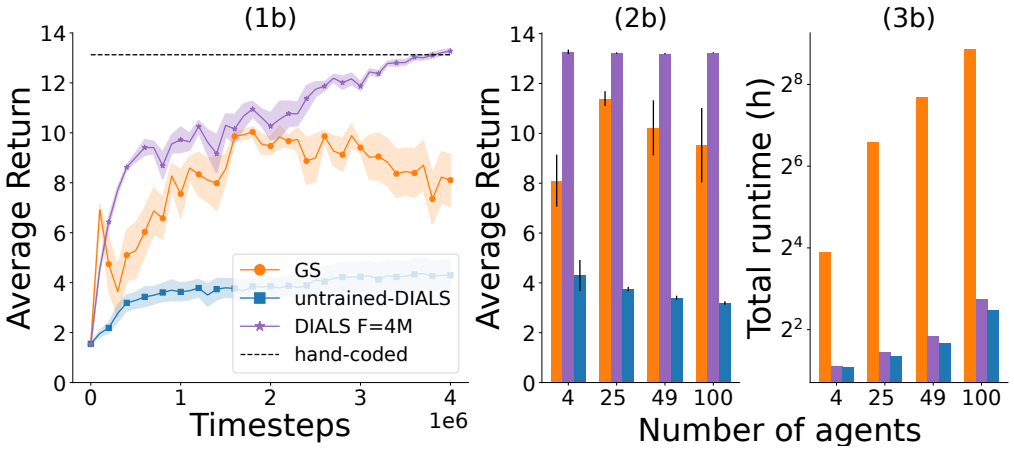
It is essential to emphasize that, while the environments above have been designed to emulate real-world challenges, they only bear a limited resemblance to the complexities and nuances inherent in realistic scenarios (Geroliminis and Daganzo, 2008; Knoop et al., 2015). Hence, the results obtained should not be automatically extrapolated to the real world. For a comprehensive review of strategies for real road traffic control, the interested reader is referred to Papageorgiou et al. (2003); Qadri et al. (2020).

4.5.3 Results

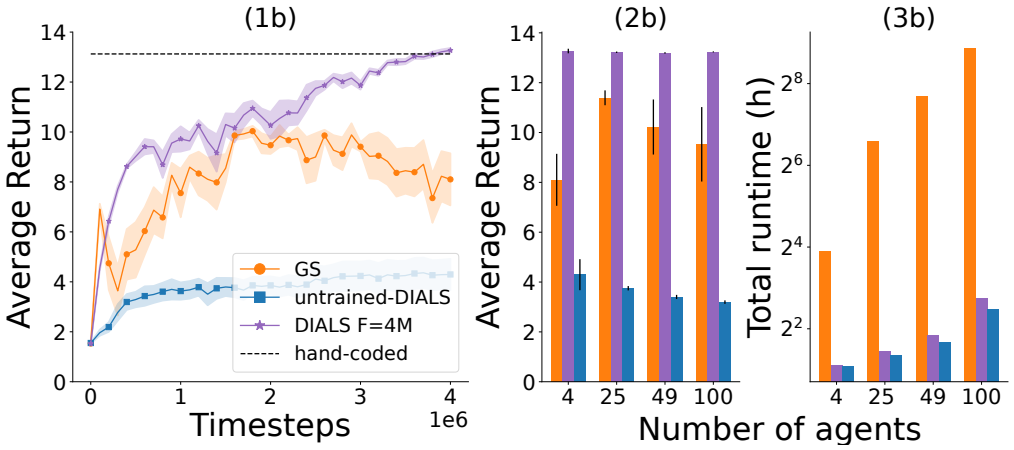
GS vs. DIALS The two plots on the left of Figures 4.3a and 4.3b show the average return as a function of the number of timesteps obtained with GS, DIALS, and untrained-DIALS on the 4-agent traffic and warehouse environments. Shaded areas indicate the standard error of the mean. Agents are trained for 4M timesteps on all three simulators. The results reveal that, while agents trained on DIALS seem to converge steadily towards similar high-performing policies, agents trained with the GS often get stuck in local minima, hence the poor mean episodic reward and large standard error obtained with the GS relative to that of the DIALS. In contrast, the low performance of agents trained with the untrained-DIALS indicates that estimating the influences correctly is important for learning good policies. It is worth noting that the gap between the GS and the DIALS is larger in Figure 4.3b than in Figure 4.3a. We posit that this is because, in the warehouse domain, agents are more strongly coupled. For comparison, the dashed-black lines in the plots on the left of Figures 4.3a and 4.3b show the performance of hand-coded policies. For the traffic domain, we used fixed traffic light controllers that were extensively optimized by Wu et al. (2017). For the warehouse domain, we hand-coded policies that follow the shortest path toward the oldest item in the agent's region. The learning curves for the other scenarios are provided in Appendix B.4 together with further discussion on these results.[6][7]

[6] A video showing the GS of the traffic network and one of the IALS is provided at `https://youtu.be/DgVE6OIQQz8`.

[7] A video showing how agents trained with DIALS perform on the 100-agent variant of the traffic scenario is provided at `https://youtu.be/G9EthZ-G3vo`.



(a) Traffic

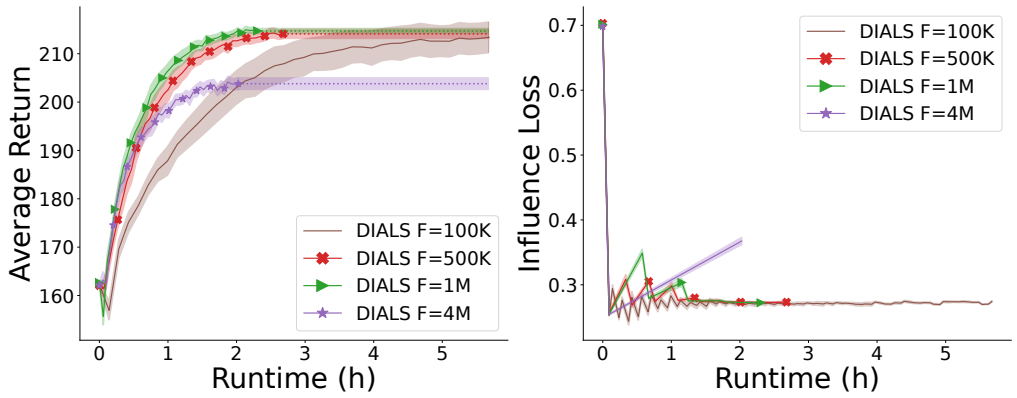


(b) Warehouse

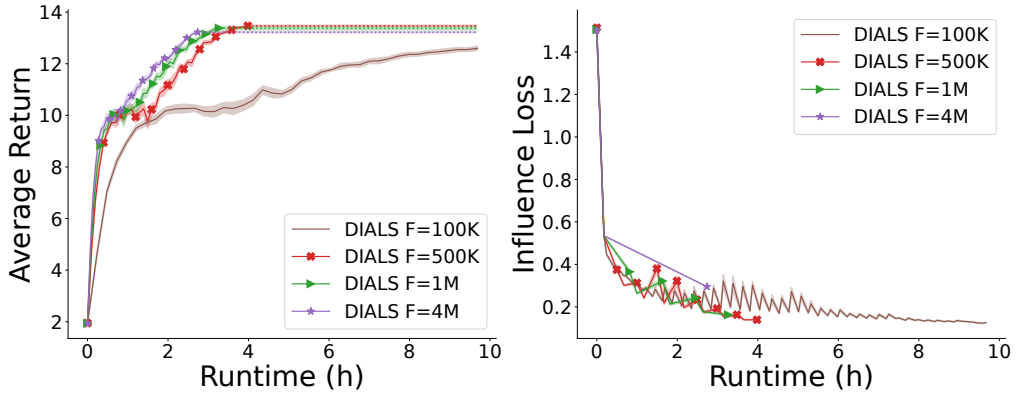
Figure 4.3: **(1a) and (1b)** Learning curves with the three simulators on the 4-intersection traffic and 4-robot warehouse environments. **(2a) and (2b):** Final average return of agents trained with the three simulators for 4M timesteps. **(3a) and (3b):** Total runtime of training with the three simulators for 4M timesteps. The y-axis is in $\log_2$ scale.

Scalability The benefits of parallelization are more apparent when moving to larger environments. The two bar plots in Figures 4.3a and 4.3b depict the final average return and the total run time of training 4, 25, 49, and 100 agents on the two tasks for 4M timesteps. The plots show that DIALS scales far better than the GS to larger problem sizes, while also yielding better-performing policies. For example, training **100 agents** on the traffic network takes less than **6 hours** with the DIALS, whereas training them with the GS would take more than **10 days**. This is a **speedup factor of 40**. In fact, since the maximum execution time allowed by our computer cluster is 1 week, the results reported for the GS in the scenarios of size 10×10 do not correspond to 4M timesteps but the equivalent of 1 week of training. We would also like to point out that, disregarding the overhead costs associated with multiprocessing, the DIALS runtime should remain constant independently of the problem size. However, to update the AIPs, new samples are collected from the GS, which does increase the runtime. This explains the gap between DIALS and untrained-DIALS. That said, the number of samples needed to update the AIPs (80K for traffic and 10K for warehouse) is significantly lower than the samples needed to train the agents (4M), which is why the runtime difference between GS and DIALS is so large. A table with a breakdown of the runtimes is given in Appendix B.7.

AIPs' training frequency Our first results have already demonstrated that isolating the agents in separate simulators and not updating the AIPs too frequently can be beneficial for convergence. We now further investigate this phenomenon by evaluating the agents' learning performance for different values of the hyperparameter F. The two plots on the left of Figures 4.4a and 4.4b show the learning curves for agents trained on DIALS where F is set to 100K, 500K, 1M, and 4M timesteps. In the traffic domain, the gap between the green and the purple curve (Figure 4.4a) suggests that it is important to retrain the AIPs at least every 1M timesteps, such that agents become aware of changes in the other agents' policies. In contrast, in the warehouse domain (Figure 4.4b), we see that training the AIPs only once at the beginning (DIALS $F = 4$M) seems sufficient. In fact, updating the AIPs too frequently (DIALS $F = 100$K) is detrimental to the agents' performance. This is consistent with our hypothesis in Section 4.4.3. The plots on the right show the average cross-entropy (CE) loss of the AIPs evaluated on trajectories sampled from the GS. As explained in Section 4.4 since all agents learn simultaneously, the influence distributions $\{I(u_t^i|l_t^i)\}_{i \in N}$ are non-stationary. For this reason, we see that the CE loss changes as the policies of the other agents are updated. We can also see how the CE loss decreases when the AIPs are retrained, which happens more or less frequently



(a) Traffic 25 agents



(b) Warehouse 25 agents

Figure 4.4: **Left (a) and (b):** Learning curves with DIALS for different values of F on the 25-agent versions of the two environments. **Right (a) and (b):** Influence CE loss as a function of runtime averaged over the 25 AIPs.

depending on the hyperparameter F. Note that the CE not only measures the distance between the two probability distributions but also the absolute entropy. In the warehouse domain, the neighbor robots' locations become more predictable (lower entropy) as their policies improve. This explains why in the first plot from the right the CE loss decreases even though the AIPs are not updated. Also in the same plot, even though by the end of training DIALS $F = 4$M is highly inaccurate, as evidenced by the gap between the purple and the other curves, it is still good enough to train policies that match the performance of those trained with DIALS $F = 500$K and $F = 1$M. This is in line with our results in Section 4.4. The same plots for the rest of the scenarios are provided in Appendix B.4.

4.6 Scope and Limitations

DIALS targets networked environments with well-defined local regions where the interactions between different regions occur through a limited number of variables. There is plenty of examples of domains that have this particular structure including traffic, heating and water systems, logistics, telecommunications, etc. Knowledge of the influence sources U and how these affect the local regions is required for building the DIALS. In most cases, however, (as in the two environments we explored here) some domain knowledge suffices to be able to tell what the influence sources are.

Moreover, having or being able to build high-fidelity local simulators of these local regions is also a requirement. Fortunately, there exist plenty of simulators of real systems that can readily be used such as, SUMO (Lopez et al., 2018), Robosuite (Zhu et al., 2020), BRAX (Freeman et al., 2021). There is also a lot of commercial software for building custom-made simulators such as Mujoco (Todorov et al., 2012), or Unity (Juliani et al., 2018). Also, note that most (if not all) of the work that has applied RL to real-world problems relies on simulation to train the policies offline (Bellemare et al., 2020; Degrave et al., 2022). Hence, we believe that DIALS can have a strong impact on many real-world applications.

Finally, although the experiments reveal that DIALS can considerably accelerate training times, it is also memory-demanding. As shown in Appendix B.8 (Table B.3), the total memory usage with DIALS increases exponentially with the number of simulators/processes. There is thus a trade-off between fast computation and total memory needed. Note, however, that the memory is split among the different processes. Hence, rather than using a big machine with large memory, DIALS can run on several smaller ones with less memory.

4.7 Conclusion

This chapter has offered a practical solution that allows training large networked systems with many agents in just a few hours. We showed how to factorize these systems into multiple sub-regions such that we could build distributed influence-augmented local simulators (DIALS).

The key advantage of DIALS is that simulations can be distributed among different

processes, and thus training can be fully parallelized. To account for the interactions between the different sub-regions, the simulators are equipped with approximate influence predictors (AIPs), which are trained periodically on real trajectories sampled from a global simulator (GS). We demonstrated that, although using DIALS agents learn simultaneously, training the AIPs very frequently is neither necessary nor desirable. Our results reveal that DIALS not only enables MARL to scale up but also mitigates the non-stationarity issues of simultaneous learning.

Future work could analyze this phenomenon from a theoretical perspective, study how to adapt DIALS to more strongly coupled domains where frequent training of the AIPs is important, or design a method to directly estimate how the changes in the agents' policies affect the influence distributions. This is so that the AIPs can readily be updated without having to run the GS to generate new samples.

Chapter 5

Influence-Aware Memory

Muchos años después, frente al pelotón de fusilamiento, el coronel Aureliano Buendía había de recordar aquella tarde remota en que su padre lo llevó a conocer el hielo.

–Gabriel García Márquez, Cien Años de Soledad

Due to its perceptual limitations, an agent may have too little information about the state of the environment to act optimally. In such cases, it is important to keep track of the observation history to uncover hidden state. Recent deep reinforcement learning methods use recurrent neural networks (RNN) to memorize past observations. However, these models are expensive to train and have convergence difficulties, especially when dealing with high dimensional input spaces. In this chapter, we propose *influence-aware memory* (IAM), a theoretically inspired memory architecture that tries to alleviate the training difficulties by restricting the input of the recurrent layers to those variables that influence the hidden state information. Moreover, as opposed to standard RNNs, in which every piece of information used for estimating Q values is inevitably fed back into the network for the next prediction, our model allows information to flow without being necessarily stored in the RNN's internal memory. Results indicate that, by letting the recurrent layers focus on a small fraction of the observation variables while processing the rest of the information with a feedforward neural network, we can outperform standard recurrent architectures in training speed and policy performance. This approach also reduces runtime and obtains better scores than methods that stack multiple observations

to remove partial observability.[1]

5.1 Introduction

It is not always guaranteed that an agent will have access to a full description of the environment to solve a particular task. In fact, most real-world problems are by nature partially observable. This means that some of the variables that define the state space are hidden (McCallum, 1995b). This type of problems can be modeled as *partially observable Markov decision processes (POMDP)* (Kaelbling et al., 1996). The model is an extension of the MDP framework (Puterman, 1994), which, unlike the original formulation, does not not assume states to be fully observable. This implies that the Markov property is no longer satisfied. That is, future observations do not solely depend on the most recent one.

Most POMDP methods try to extract information from the full action-observation history to disambiguate hidden state. We argue however, that in many cases, memorizing all the observed variables is costly and requires unnecessary effort. Instead, we can exploit the structure of our problem and abstract away from our history those variables that have no direct influence on the hidden ones.

Previous work on *influence-based abstraction (IBA)* (Witwicki and Durfee, 2010b; Oliehoek et al., 2012) demonstrates that, in certain POMDPs, the non-Markovian dependencies in the transition and reward functions can be fully determined given a subset of variables in the history. Hence, the combination of this subset together with the current observation forms a Markov representation that is sufficient to compute the optimal policy. In this chapter, we use these theoretical insights to propose a new memory model that tries to correct certain flaws in standard RNNs that limit their effectiveness when applied to reinforcement learning (RL). We identify two key features that make our model stand apart from the most widely used recurrent architectures, LSTMs (Hochreiter and Schmidhuber, 1997) and GRUs (Cho et al., 2014):

1. The input of the RNN is restricted to a subset of observation variables which in principle should contain sufficient information to estimate the hidden state.

2. There is a feedforward connection parallel to the recurrent layers, through which the information that is important for estimating Q values but that does not need to be

[1]This chapter is based on Suau et al. (2022b).

memorized can flow.

Although these two features might be overlooked as minor modifications to the standard architectures, together, they provide a theoretically sound inductive bias that brings the structure of the model into line with the problem of hidden state. Moreover, as shown in our experiments, they have an important effect on convergence, learning speed, and final performance of the agents.

5.2 Related Work

Partial observability: The problem of partial observability has been extensively studied in the past. The main bulk of the work, comes from the planning community where most solutions rely on forming a belief over the states of the environment using agent's past observations (Ng and Jordan, 2000; Pineau et al., 2003; Silver and Veness, 2010). Classic RL algorithms, on the other hand, cannot directly apply the above solution due to the lack of a fully specified transition model. Instead, they learn stochastic policies that rely only on the current observation (Littman, 1994; Jaakkola et al., 1995), or use a finite-sized history window to estimate the hidden state (Lin and Mitchell, 1993; McCallum, 1995a). Curiously enough, even though the previous solutions do not scale to large and continuous state spaces, in the field of Deep RL the problem is most of the times either ignored, or naively overcome by stacking a window of past observations (Mnih et al., 2015). Other more sophisticated approaches incorporate external memories (Oh et al., 2016b) or use RNNs to keep track of the past history (Schmidhuber, 1991; Hausknecht and Stone, 2015; Jaderberg et al., 2019). Although this solution is much more scalable, recurrent models are computationally expensive and often have convergence difficulties when working with high dimensions (McCallum, 1995b; Kaelbling et al., 1996). A few works, have tried to aid the RNN by using auxiliary tasks like predicting game feature information (Lample and Chaplot, 2017) or image reconstruction (Igl et al., 2018). We, on the other hand, recognize that the internal structure of standard RNNs might not always be appropriate and propose a new memory architecture that is better aligned with the RL problem.

Attention: One of the variants of the memory architecture we propose implements a spatial attention mechanism (Xu et al., 2015) to provide the network with a layer of dynamic weights. This form of attention is different from the temporal attention

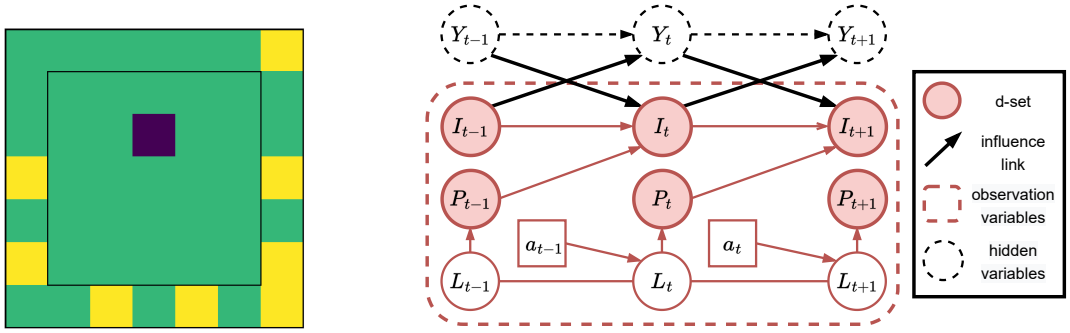


Figure 5.1: Warehouse environment (left). Dynamic Bayesian Network describing the environment dynamics (right).

mechanism that is used in seq2seq models (Luong et al., 2015; Vaswani et al., 2017). While the latter allows the RNN to condition on multiple past internal memories to make predictions, the spatial attention mechanism we use, is meant to filter out a fraction of the information that comes in with the observations. Attention mechanisms have recently been used in the context of Deep RL to facilitate the interpretation of the agent's behavior (Mott et al., 2019; Tang et al., 2020) or to tackle multi-agent problems (Iqbal and Sha, 2019). Similar to our model, the architecture proposed by Sorokin et al. (2015) also uses an attention mechanism to find the relevant information in the game screen and feed it into the RNN. However, their model misses the feedforward connection through which the information that is useful for predicting action values but that does not need to be stored in memory can flow (see Section 5.4.1 for more details).

5.3 Example: Warehouse Commissioning

Figure 1 shows a robot (purple) that needs to fetch the items (yellow) that appear with probability 0.05 on the shelves at the edges of the 7×7 grid representing a warehouse. The robot receives a reward of $+1$ every time it collects an item. The added difficulty of this task is that item orders get canceled if they are not collected before 8 timesteps since they appear. Thus, the robot needs to maintain a time counter for each item and decide which one is best to go for.

The dynamics of the problem are represented by the dynamic Bayesian network (DBN) (Pearl, 1988; Boutilier et al., 1999) in Figure 5.1 (right), where L_t denotes the robot's current location in the warehouse, and I_t and P_t are binary variables indicating if the item

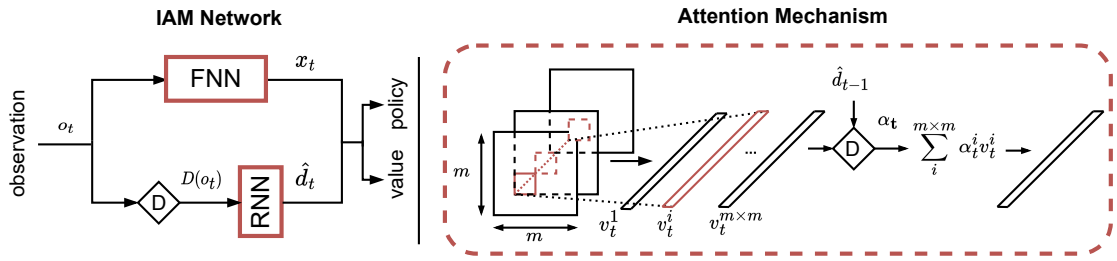


Figure 5.2: Influence-aware Memory network architecture (left). Diagram of the attention mechanism for image data (right)

order is active and whether or not the robot is at the item pick-up location. The hidden variable Y_t is the item's time counter[2], to which the robot has no access. The robot can only infer the time counter based on past observations. To do so, however, it does not need to remember the full history, but only whether or not a given item order was active at a particular timestep. More formally, inspecting the DBN, we see that Y_{t+1} is only indirectly influenced by the agent's past location L_{t-1} via P_{t-1} and the item variable I_t. Therefore, we say that Y_{t+1} is conditionally independent of L_{t-1} given P_{t-1} and I_t,

$$Pr(Y_{t+1} = y_{t+1} | L_{t-1} = l_{t-1}, P_{t-1} = p_{t-1}, I_t = i_t) = Pr(Y_{t+1} = y_{t+1} | P_{t-1} = p_{t-1}, I_t = i_t) \quad (5.1)$$

This means that in order to infer the hidden variable Y at any timestep it is sufficient to condition on the past values of P and I. The history of these two variables, highlighted in green in Figure 5.1, constitutes the d-separating set (d-set; Definition 2.3.4).

5.4 Influence-Aware Memory

The properties outlined in the previous section, are not unique to the warehouse example. In fact, as we show in our experiments, it is often the case in partially observable problems that only a fraction of the observation variables influence the hidden state. This does not necessarily imply that the agent can completely ignore the rest of the information. In the warehouse example, the robot's current location, despite being irrelevant for inferring the hidden state, is in fact crucial for estimating the current action values.

The Bellman equation for the optimal action value function Q^* of a POMDP can be

[2]For simplicity, we only include a single item in the DBN in Figure 5.1. The dynamics of all the other items in the warehouse are analogous.

expressed in terms of the history of actions and observations h_t as

$$Q^*(h_t, a_t) = R(h_t, a_t) + \sum_{o_{t+1}} \Pr(o_{t+1}|h_t, a_t) \max_{a_{t+1}} Q^*(h_{t+1}, a_{t+1}), \tag{5.2}$$

with

$$\Pr(o_{t+1}|h_t, a_t) = \sum_{o_{t+1}, s_{t+1}, y_t} \Pr(o_{t+1}|s_{t+1}) Pr(s_{t+1}|o_t, y_t, a_t) Pr(y_t|h_t) \tag{5.3}$$

where $R(h_t, a_t) = \sum_{s_t} \Pr(s_t|h_t) R(s_t, a_t)$ is the expected immediate reward at time t over the set of possible states s_t given a particular history h_t.

Exploiting d-separation, we can replace the dependence on the full history of actions and observations h_t by a dependence on the d-set d_t,

$$\begin{aligned} &Q^*(\langle d_t, o_t \rangle, a_t) = R(\langle d_t, o_t \rangle, a_t) \\ &+ \sum_{o_{t+1}} \Pr(o_{t+1}|\langle d_t, o_t \rangle, a_t) \max_{a_{t+1}} Q^*(\langle d_{t+1}, o_{t+1} \rangle, a_{t+1}), \end{aligned} \tag{5.4}$$

$$\begin{aligned} &Q^*(\langle d_t, o_t \rangle, a_t) = R(\langle d_t, o_t \rangle, a_t) \\ &+ \sum_{o_{t+1}, y_t} Pr(o_{t+1}|s_{t+1}) \Pr(s_{t+1}|\langle y_t, o_t \rangle, a_t) Pr(y_t|d_t) \max_{a_{t+1}} Q^*(\langle d_{t+1}, o_{t+1} \rangle, a_{t+1}), \end{aligned} \tag{5.5}$$

and $d_{t+1} \triangleq \langle d_t, D(o_{t+1}) \rangle$, where $D(\cdot)$ is the d-set selection operator, which chooses the variables in o_{t+1} that are added to d_{t+1}. Note that, although d_t contains enough information to estimate the hidden state, o_t is still needed to estimate Q. Hence, given the tuple $\langle d_t, o_t \rangle$ we can write

$$Q^*(h_t, a_t) = Q^*(\langle d_t, o_t \rangle, a_t), \tag{5.6}$$

The upshot is that in most POMDPs the combination of d_t and o_t forms a Markov representation that the agent can use to find the optimal policy. Unfortunately, in the RL setting, we are normally not provided a fully specified DBN to determine the exact d-set. Nonetheless, in many problems like in our warehouse example it is not difficult to make an educated guess about the variables containing sufficient information to predict the hidden ones. The network architecture we present in the next section enables us to select beforehand what variables the agent should memorize. This is however not an prerequisite since, as we explain in Section 5.4.2, we can also force the RNN to find such variables by restricting its capacity.

5.4.1 Influence-Aware Memory Network

The Influence-aware Memory (IAM) architecture we propose is depicted in Figure 5.2. The network tries to encode the ideas of IBA as inductive biases with the goal of being able to learn policies and value functions more effectively. Following from (5.6), our architecture implements two separate networks in parallel: an FNN, which processes the entire observation,

$$x_t = F_{\text{fnn}}(o_t), \tag{5.7}$$

and an RNN, which receives only $D(o_t)$ and updates its internal state,

$$\hat{d}_t = F_{\text{rnn}}(\hat{d}_{t-1}, D(o_t)), \tag{5.8}$$

where we use the notation $\hat{d}_t$ to indicate that the d-set is embedded in the RNN's internal memory. The output of the FNN x_t is then concatenated with $\hat{d}_t$ and passed through two separate linear layers which compute values $Q(\langle x_t, \hat{d}_t \rangle, a_t)$ and action probabilities $\pi(\langle x_t, \hat{d}_t \rangle, a_t)$.

IAM vs. standard RNNs: We try to facilitate the task of the RNN by feeding only the information that, in principle, should be enough to uncover hidden state. This is only possible thanks to the parallel FNN channel, which serves as an extra gate through which the information that is useful for predicting action values but that does not need to be stored in memory can flow. This is in contrast to the standard recurrent architectures that are normally used in Deep RL (e.g. LSTM, GRU, etc.), which suffer from the fact that every piece of information that is used for estimating values is inevitably fed back into the network for the next prediction. Intuitively, standard RNNs face a conflict: they need to choose between ignoring those variables that are unnecessary for future predictions, risking worse Q estimates, or processing them at the expense of corrupting their internal memory with irrelevant details. Figure 5.3 illustrates this idea by comparing the information flow in both architectures.

Finally, since the recurrent layers in IAM are freed from the burden of having to remember irrelevant information, they can be dimensioned according to the memory needs of the problem at hand. This translates into networks that combine regular size FNNs together with small RNNs.

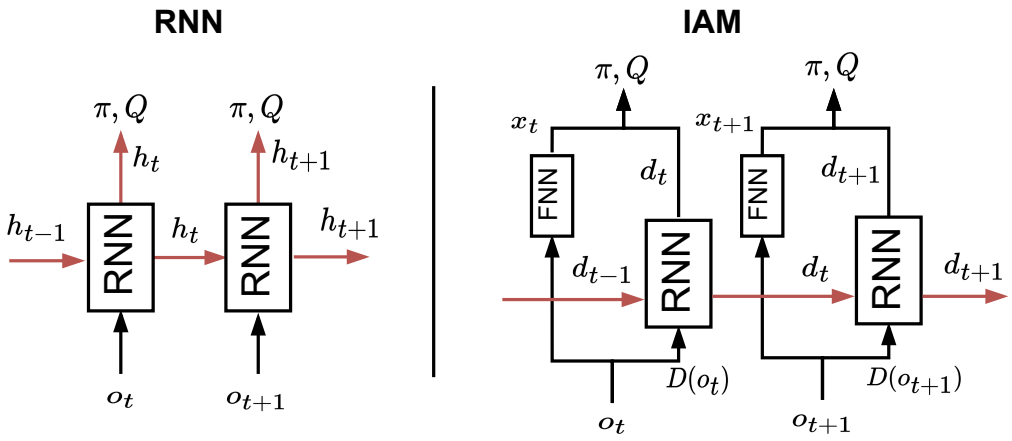


Figure 5.3: Information flow in standard RNNs (left) compared to IAM (right). The diagram on the left shows that the same vector h_t that is used for estimating π and Q is also part of the input for the next prediction (green arrows). On the other hand, in the IAM architecture there is another vector x_t coming out from the FNN, which is only used for estimating π and Q at time t and is not stored in memory. Hence, the RNN in IAM is free to include in $\hat{d}_t$ only the information that the agent needs to remember.

Image data: If our agent receives images rather than feature vectors, we first preprocess the raw observations o with a CNN, $F_{\text{cnn}}(o_t) = \mathbf{v_t}$ and obtain $m \times m$ vectors v of size N, where N is the number of filters in the last convolutional layer and $m \times m$ the dimensions of the 2D output array of each filter (Figure 5.2 right). Fortunately, since the convolution operator preserves the topology of the input, each of these vectors corresponds to a particular region of the input image. Thus, we can still use domain knowledge to choose which vectors should go into the RNN.

5.4.2 Learning Approximate D-sets

Having the FNN channel can help detach the RNN from the task of estimating the current Q values. However, without the d-set selection operator D, nothing prevents the information that does not need to be remembered from going through the RNN. Although, as we show in our first two experiments, it is often possible for the designer to guess what variables directly influence the hidden state information, it might not always be so straightforward. In such cases, rather than manually selecting the d-set, the agent will have to learn D from experience. In particular, we add a linear layer before the RNN, to act as information bottleneck (Tishby and Zaslavsky, 2015) and filter out all that

information that is irrelevant:

$$\hat{D}_A(o_t) = Ao_t \tag{5.9}$$

where $\hat{D}$ indicates that the operator is learned rather than handcrafted and A is a matrix of weights of size $K \times N$, where N is the number of observation variables (the number of filters in the last convolutional layer when using images) and K is a hyperparameter that determines the dimensions of the output. The matrix A needs to be computed differently depending on the nature of the problem:

Static d-sets: If the variables that must go into the d-set do not change from one timestep to another. That is, if D always needs to choose the same subset of observation variables, as occurs in the warehouse example, we just need a fixed matrix A to filter all observations in the same way. A can be implemented as a separate linear layer before the RNN or we can just directly restrict the size the RNN's input layer.

Dynamic d-sets: If, on the other hand, the variables that must go into the d-set do change from one timestep to another, we use a multi-head spatial attention mechanism (Xu et al., 2015; Vaswani et al., 2017) to recompute the weights in every iteration. Thus we write A_t to indicate that the weights can now adapt to o_t and $\hat{d}_{t-1}$. The need for such dynamism can be easily understood by considering the Atari game of breakout. To be able to predict where the ball will be next, the agent does not need to memorize the whole set of pixels in the game screen, but only the ones containing the ball, whose location differs in every observation and hence the need of a varying matrix A_t. Specifically, for each row i in A_t, each element $\alpha_t^{i,j}$ is computed by a two-layer fully connected network that takes as input the corresponding element in the observation vector o_i and $\hat{d}_{t-1}$, followed by a softmax operator. Figure 5.2 is a diagram of how each of the attention heads operates for the case of using as input the output of the CNN $\mathbf{v_t}$ instead of the observation vector o_t. Please refer to Appendix C.1 for more details about the technical implementation of this mechanism.

Note that the above solutions would not be able to filter out the information that is only useful for the current Q estimates without the parallel FNN connection (Figure 5.3). We would also like to stress that these mechanisms are by no means guaranteed to find the optimal d-set. Nonetheless, as shown in our experiments, they constitute an effective inductive bias that facilitates the learning process.

5.5 Experiments

We empirically evaluate the performance of our memory architecture on the warehouse example (Section 5.3), a traffic control task, and the *flickering* version of the Atari video games (Hausknecht and Stone, 2015). The goal of our experiments is:

1. **Learning performance and convergence:** Evaluate whether our model improves over standard recurrent architectures. We compare learning performance, convergence and training time.
2. **High dimensional observation spaces:** Show that our solution scales to high dimensional problems with continuous observation spaces.
3. **Learning approximate d-sets:** Demonstrate the advantages of restricting the input to the RNN and compare the relative performance of learning vs. manually specifying the d-sets.
4. **Architecture analysis:** Analyze the impact of the architecture on the learned representations by inspecting the network hidden activations.

5.5.1 Environments

Below is a brief description of the three domains on which we evaluate our model. Please refer to Appendix C.1 for more details.

Warehouse: This is the same task we describe in our example in Section 5.3. The observations are a combination of the agent's location (one-hot encoded vector) and the 24-item binary variables. In the experiments where d-sets are manually selected, the RNN in IAM only receives the latter variables while the FNN processes the entire vector.

Traffic Control: In this environment (Lopez et al., 2018), the agent must optimize the traffic flow at the intersection in Figure 5.4. The agent can take two different actions: either switching the traffic light on the top to green, which automatically turns the other to red, or vice versa. The observations are binary vectors that encode whether not there is a car at a particular location. Cars are only visible when they enter the red box. There is a 6 seconds delay between the moment an action is taken and the time the lights actually

switch. During this period the green light turns yellow, and no cars are allowed to cross the road. Agents need to anticipate cars entering the red box and switch the lights in time for them to continue without stopping. This forces the recurrent models to remember the location and the time at which cars left the intersection and limits the performance of agents with no memory[3]. In the experiments where d-sets are manually selected, the RNN in IAM receives the last two elements in each of the two vectors encoding the road segments (i.e. 4 bits in total). The location of these elements is indicated by the small grey boxes in Figure 5.4. This information should be sufficient to infer hidden state.

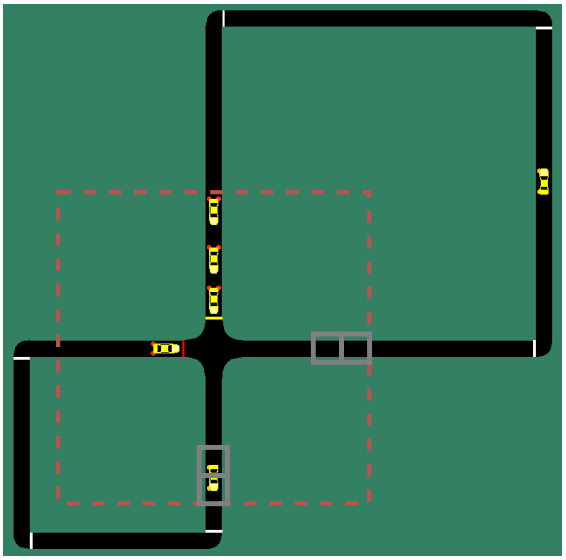


Figure 5.4: Traffic control environment. Cars are only visible when they enter the red box. The small grey boxes show the variables that we feed into the RNN.

Flickering Atari: In this version of the Atari video games (Bellemare et al., 2013) the observations are replaced by black frames with probability $p = 0.5$. This adds uncertainty to the environment and makes it more difficult for the agent to keep track of moving elements. The modification was introduced by Hausknecht & Stone (2015) to test their recurrent version of DQN and has become the standard benchmark for Deep RL in POMDPs (Zhu et al., 2017; Igl et al., 2018).

[3]Videos showing the results of the traffic control experiment can be found at `https://tinyurl.com/wc3jpf4`

5.5.2 Experimental Setup

We compare IAM against two other network configurations: A model with no internal memory that uses frame stacking (FNN) and a standard recurrent architecture (LSTM). All three models are trained using PPO (Schulman et al., 2017). For a fair comparison, and in order to ensure that both types of memory have access to the same amount of information, the sequence length parameter in the recurrent models (i.e. number of time steps the network is unrolled when updating the model) is chosen to be equal to the number of frames that are fed into the FNN baseline. We evaluate the performance of our agents at different time steps during training by calculating the mean episodic return. The results are averaged over three random seeds. A table containing the full list of hyperparameters used for each domain and for each of the three architectures, together with a detailed description of the tuning process, is provided in Appendix C.1.

5.5.3 Learning Performance and Convergence

We first evaluate the performance of our model on the warehouse and traffic control environments. Although the observation sizes are relatively small compared to most deep RL benchmarks (73 and 30 variables respectively), the two tasks are quite demanding memory-wise. In the warehouse environment, the agent is required to remember for how long each of the items has been active. In the traffic domain, cars take 32 timesteps to reappear again in the red box when driving around the big loop (Figure 5.4). Figure 5.5, shows the learning curves of IAM and LSTM in the two environments. While both LSTM and IAM reach similar levels of performance on the traffic control task the LSTM network takes much longer to converge (bottom). On the other hand, in the warehouse environment, IAM clearly outperforms the LSTM baseline (top). The final scores obtained by FNNs with (red) and without memory[4] (black) (i.e. observation stacking) are also included for reference. These results are strong evidence that the parallel feedforward channel in IAM is indeed helping overcome the convergence difficulties of LSTMs, by bypassing the recurrent layers (Section 5.4.1).

Figure 5.6 is a performance comparison of LSTM and IAM for various recurrent layer sizes[5].

[4]Please note that, although the optimal policy in these two environments requires memory, memoryless policies can still reach a decent performance level.

[5]For a fair comparison, the reported results for IAM correspond to networks where $\hat{D}$ is learned and not manually specified. The labels indicate the total number of recurrent neurons. Both LSTMs and IAMs have also feedfoward layers of equivalent size. A detailed description of how these were chosen so as

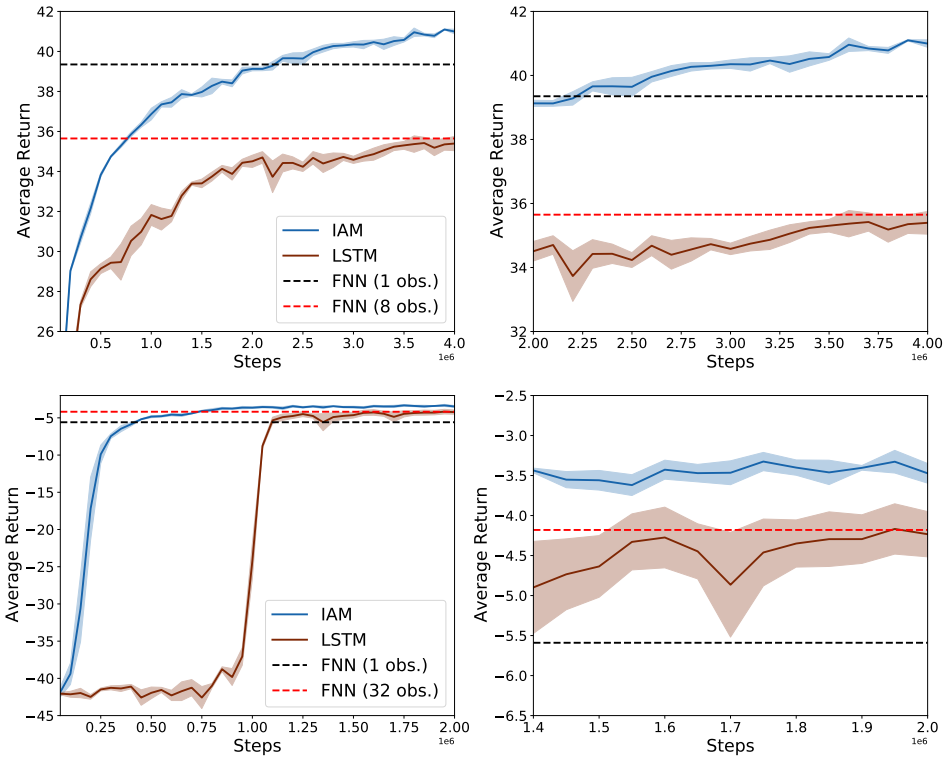


Figure 5.5: Average return and standard deviation during training of IAM and LSTM as a function of the number of timesteps on the warehouse (top) and the traffic (bottom) environments. For ease of visualization, the plots on the right are zoomed in versions of the ones on the left. The dashed horizontal lines indicate the final performance of FNNs with (red) and without memory (black).

While the best recurrent layer size for the LSTM baseline is **128** in both domains, the size of the recurrent model of IAM can be brought down to **64** for the warehouse environment and just **8** recurrent neurons for the traffic task, and still outperform the memoryless FNN baseline (dashed black curve). This, of course, translates into a significant reduction in the total number of weights and computational speedups. A full summary of the average runtime for each architecture, along with a description of the computing infrastructure used, is given in Appendix C.2.

not to interfere with the results is provided in Appendix C.1.

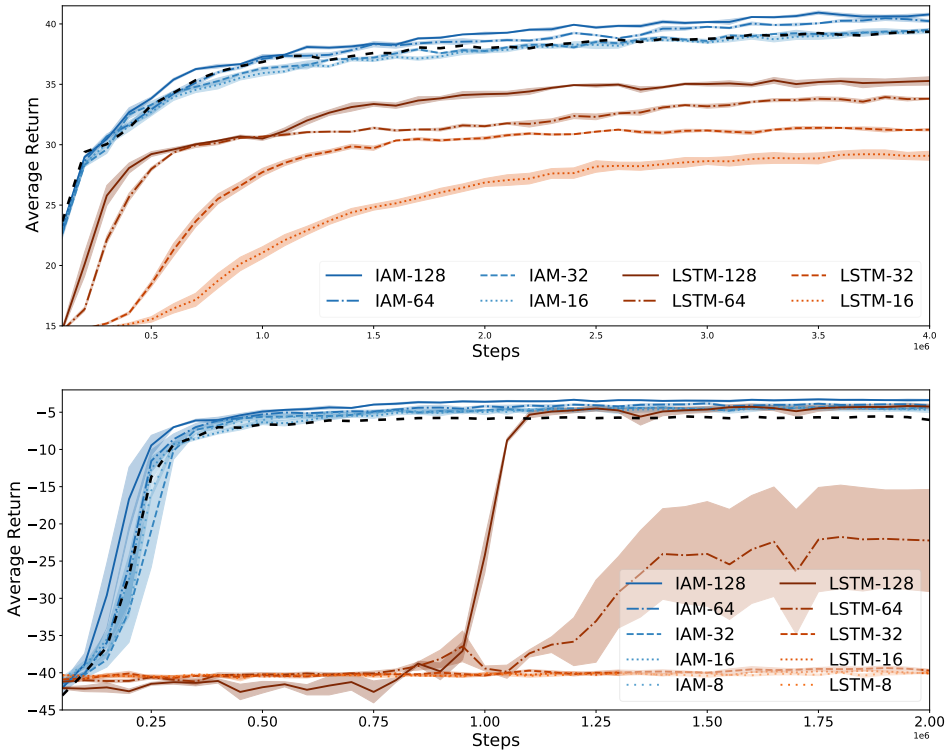


Figure 5.6: Average return and standard deviation during training of IAM (learned $\hat{D}$) and LSTM for various recurrent layer sizes on the warehouse (top) and the traffic (bottom) environments. The dashed black lines are the learning curves of FNNs without memory.

5.5.4 High Dimensional Observation Spaces

The advantage of IAM over LSTMs and FNNs becomes even more apparent as the dimensionality of the problem increases. Table 5.1 compares the average scores obtained in Flickering Atari by the FNN and LSTM baselines with those of IAM. Both IAM and LSTM receive only 1 frame. The sequence length parameter is set to 8 time steps for the two networks. The FNN model, on the other hand, receives the last 8 frames as input. The learning curves are shown in Appendix C.3, together with the results obtained in the original games and the average runtime.

Table 5.1: Average final score on the Flickering Atari games for each of the three network architectures and standard deviation. Bold numbers indicate the best results on each environment.

	FNN (8 frames)	LSTM	IAM
Breakout	26.57 ± 1.51	21.32 ± 0.45	$\mathbf{83.10 \pm 5.29}$
Pong	18.07 ± 0.06	-20.25 ± 0.03	$\mathbf{20.07 \pm 0.11}$
Space Invaders	$\mathbf{854.93 \pm 11.64}$	520.44 ± 9.41	$\mathbf{834.66 \pm 21.23}$
Asteroids	1393.75 ± 11.28	1424.87 ± 5.23	$\mathbf{2281.63 \pm 63.92}$
MsPacman	$\mathbf{2388.03 \pm 167.03}$	1081.11 ± 293.79	$\mathbf{2326.04 \pm 31.53}$

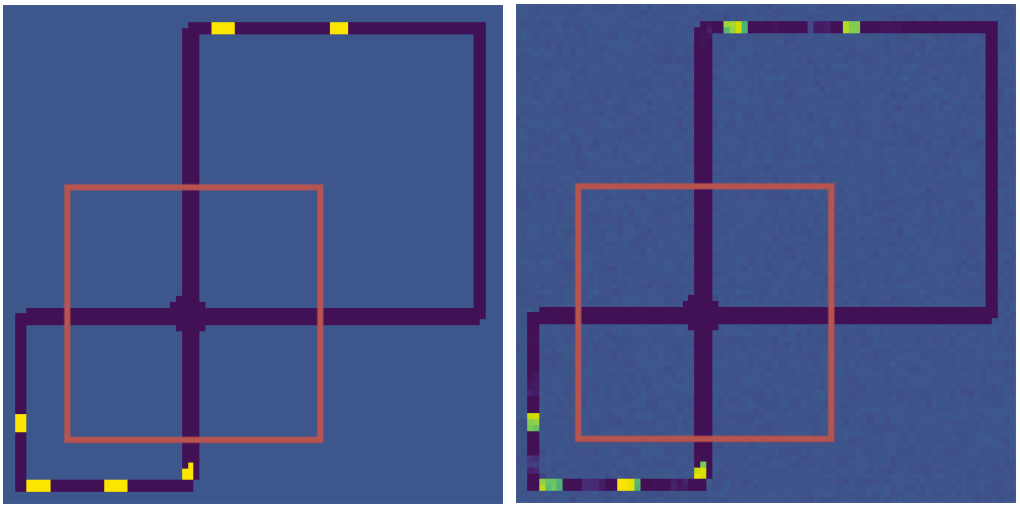


Figure 5.7: Example of a full game screen and the reconstruction made by the memory decoder.

5.5.5 Learning Approximate D-sets.

As explained in Section 5.4.2, if the optimal d-set is static, like in the warehouse and traffic environments, we might be able to learn $\hat{D}$ by simply restricting the size of the RNN. The first two rows in Table 5.2, show a performance comparison between manually selecting our d-set, and forcing the network weights to filter out the irrelevant variables by limiting its capacity. The problem needs to be treated with a bit more care in cases where the variables that influence the hidden state change from one observation to another, as occurs in the Atari games. In such situations, just restricting the size of the RNN is not sufficient since the weights are static, and hence unable to settle for any particular subset of pixels in the game screen (Section 5.4.2). The last two columns in Table 5.2 are the scores obtained in Flickering Atari with static weights (2nd column) and with the dynamic weights computed by the attention mechanism (3rd column). Manual selection is not

Table 5.2: Comparison between manually selecting the d-set and learning it, and between using static and dynamic weights.

	Manual	Learned (static)	Learned (dynamic)
Warehouse	$\mathbf{40.99 \pm 0.20}$	40.76 ± 0.12	-
Traffic Control	-3.76 ± 0.12	$\mathbf{-3.37 \pm 0.09}$	-
Breakout	-	26.17 ± 0.82	$\mathbf{83.10 \pm 5.29}$
Pong	-	$\mathbf{19.72 \pm 0.27}$	$\mathbf{20.07 \pm 0.11}$
Space Invaders	-	583.12 ± 8.76	$\mathbf{834.66 \pm 21.23}$
Asteroids	-	1841.87 ± 26.09	$\mathbf{2281.63 \pm 63.92}$
MsPacman	-	2097.97 ± 34.71	$\mathbf{2326.04 \pm 31.53}$

feasible in the Atari domain.

5.5.6 Architecture Analysis

Decoding the agent's internal memory: We evaluated if the information stored in the agent's internal memory after selecting the d-set and discarding the rest of the observation variables was sufficient to uncover hidden state. To do so, we trained a decoder on predicting the full game screen given the encoded observation x_t and $\hat{d}_t$, using a dataset of images and hidden activations collected after training the policy. The image on the leftmost of Figure 5.7 shows an example of the full game screen, from which the agent only receives the region delimited by the red box. The second image from the right shows the prediction made by the decoder. Note that although everything outside the red box is invisible to the agent, the decoder is able to make a fair reconstruction of the entire game screen based on the d-set encoded in the agent's internal memory $\hat{d}$. This implies that IAM can capture the necessary information and remember how many cars left the intersection and when without being explicitly trained to do so[6].

Analysis of the hidden activations: Finally, we used Canonical Correlation Analysis (CCA) (Hotelling, 1992) to measure the correlation between the network hidden activations when playing Breakout and two important game features: ball velocity and number of bricks destroyed. The projections of the hidden activations onto the space spanned by the canonical variates are depicted in the two plots on the right of Figure 5.7. The scatter plot on the left shows four distinct clusters of hidden memories $\hat{d}_t$. Each of these clusters

[6] A video of this experiment where we use the decoder to reconstruct an entire episode can be found at `https://tinyurl.com/y9cvuz7l`. More examples are provided in Appendix C.5.

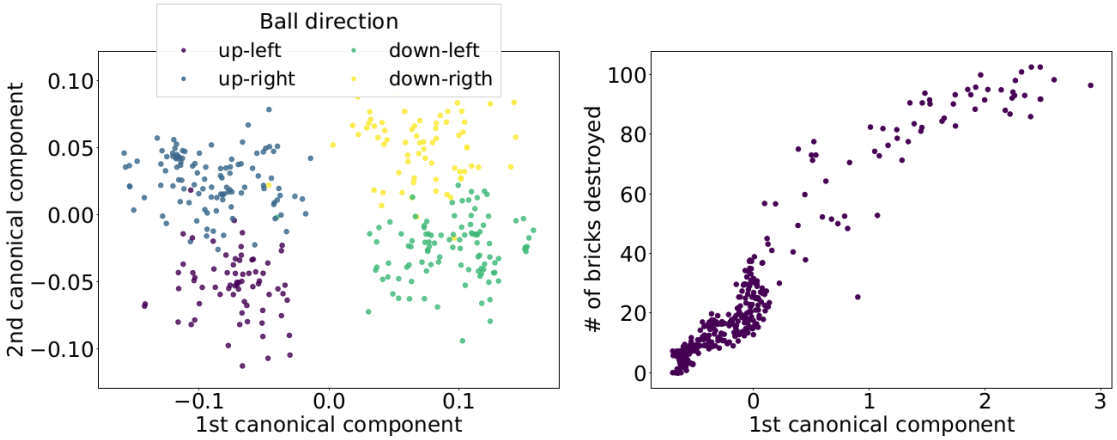


Figure 5.8: **Left:** RNN's internal memories $\hat{d}$ projected onto the two first canonical components, colors indicate the direction of the velocity vector. **Right:** FNN's outputs x projected onto the first canonical component against the number of bricks destroyed.

corresponds directly to one of the four possible directions of the velocity vector. The plot on the right, shows a clear uptrend. High values of the first canonical component of x_t correspond to frames with many missing bricks. While the FNN is taking care of the information that does not need to be memorized (i.e. number of bricks destroyed) the RNN is focused on inferring hidden state variables (i.e. ball velocity). More details about this experiment are given in Appendix C.6.

5.6 Conclusion

The primary goal of this chapter was to reconcile neural network design choices with the problem of partial observability. We studied the underlying properties of POMDPs and developed a new memory architecture that tries to decouple hidden state inference from value estimation. Influence-aware memory (IAM) connects an FNN and an RNN in parallel. This simple solution allows the RNN to focus on remembering just the essential pieces of information. This is not the case in other recurrent architectures. Gradients in LSTMs and GRUs need to reach a compromise between two, often competing, goals. On the one hand, they need to provide good Q estimates and on the other, they should remove from the internal memory everything that is irrelevant for future predictions. Our model enables the designer to select beforehand what variables the agent should memorize. This is however not an prerequisite since, as shown in our experiments, we can force the RNN

to find such variables by restricting its capacity. We also investigated a solution for those problems in which the variables influencing the hidden state information differ from one observation to another. Our results suggest that while standard architectures have severe convergence difficulties, IAM can even outperform methods that stack multiple frames to remove partial observability. Finally, aside from the clear benefits in learning performance, our analysis of the network hidden activations suggests that the inductive bias introduced in our memory architecture enables the agent to choose what to remember.

Chapter 6

Policy Confounding

> *'How can I tell,' said the man, 'that the past isn't a fiction designed to account for the discrepancy between my immediate physical sensations and my state of mind?'*
>
> –Douglas Adams, The Restaurant at the End of the Universe

Reinforcement learning agents may sometimes develop habits that are effective only when specific policies are followed. After an initial exploration phase during which agents try out different actions in the environment, they eventually converge on a particular policy. At this point, the distribution over state-action trajectories becomes narrower, leading agents to repeatedly experience the same transitions. This repetitive exposure can give rise to spurious correlations. Agents may then pick up on these correlations and develop simple habits that only work well within the specific set of trajectories dictated by their policy. The issue here is that these habits can result in incorrect outcomes if agents are forced to deviate from their typical trajectories due to changes in the environment or in their policies. In this chapter, we provide a mathematical characterization of this phenomenon, which we refer to as policy confounding, and show, through a series of examples, when and how it occurs in practice.[1]

[1] This chapter is based on Suau et al. (2023).

6.1 Introduction

This morning, I went to the kitchen for a coffee. When I arrived,
I forgot why I was there, so I got myself a coffee—

How often do you do something without paying close attention to your actions? Have you ever caught yourself thinking about something else while washing the dishes, making coffee, or cycling? Acting out of habit is a vital human skill as it allows us to concentrate on more important matters while carrying out routine tasks. You can commute to work while thinking about how to persuade your boss to give you a salary raise or prepare dinner while imagining your next holidays in the Alps. However, unlike in the above example, habits can also lead to undesired outcomes when we fail to recognize that the context has changed. You may hop in your car and start driving towards work even though it is a Sunday and you actually want to go to the grocery store, or you may flip the light switch when leaving a room even though the lights are already off.

Here, we show that reinforcement learning (RL) agents also struggle from this same issue. This is due to a phenomenon we term *policy confounding*, which reflects how policies, as a result of influencing both past and future observation variables, may induce spurious correlations (Pearl et al., 2016) among them. These correlations can lead to the development of seemingly sensible but incorrect habits, such as automatically flipping the light switch upon leaving a room, without confirming whether the lights are on. This occurs when agents are forced to deviate from their usual trajectories due to changes in their policies or the environment. We refer to this problem as *out-of-trajectory* (OOT) generalization. It is important to note that OOT generalization differs from the standard RL generalization problem (Kirk et al., 2023) in that the objective is not to generalize to environments with different dynamics and (or) rewards but rather generalize to different trajectories within the same environment.

Contributions This chapter introduces and characterizes the phenomenon of *policy confounding*. To do so, we provide a mathematical framework that helps us describe the different types of state representations, and reveal how, as a result of policy confounding, the agent may learn representations based on spurious correlations that do not guarantee OOT generalization. Moreover, we include a series of clarifying examples that illustrate how this occurs. Unfortunately, we do not have a complete answer for how to prevent policy confounding. However, we suggest a few off-the-shelf solutions that may help

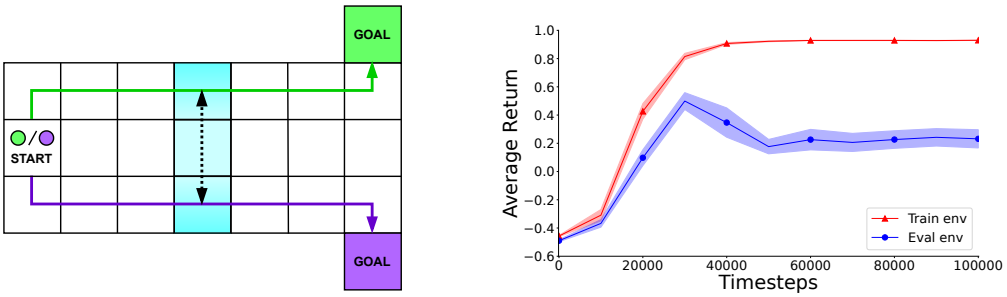


Figure 6.1: Left: An illustration of the Frozen T-Maze environment. Right: Learning curves when evaluated in the Frozen T-Maze environment with (blue curve) and without (red curve) ice.

mitigate its effects. We hope this work will create awareness among the RL community about the risks of policy confounding and inspire further research on this topic.

6.2 Example: Frozen T-Maze

We now provide an example to illustrate the phenomenon and motivate the need for careful analysis. The environment shown in Figure 6.1 is a variant of the popular T-Maze environment (Bakker, 2001). The agent receives a binary signal, green or purple, at the start location. Then, it needs to move to the right and reach the correct goal at the end of the maze (ignore the blue cells and the black vertical arrow in the middle of the maze for now). The agent obtains a reward of $+1$ for moving to the green (purple) goal when having received the green (purple) signal and a reward of -1 otherwise. There is also a -0.1 penalty per timestep to encourage the agent to take the shortest path toward the goal. At first sight, one may think that the only way the agent can solve the task is if, at every cell along its trajectory, it can recall the initial signal. However, once the agent figures out the shortest path to each of the two goals (depicted by the green and purple arrows), the agent may safely forget the initial signal. The agent knows that whenever it is at any of the cells along the green (purple) path, it must have received the green (purple) signal. Hence, it can simply move toward the right goal on the basis of its own location. Sticking to this habit is optimal so long as the agent commits to always taking these two paths.[2] It is also essential that the environment's dynamics remain the same since even the slightest change in the agent's trajectories may erase the spurious correlation induced

[2]Note that the two paths highlighted in Figure 6.1 are not the only optimal paths. However, for the agent to be able to ignore the initial signal, it is important that the paths do not overlap.

by the agent's policy between the agent's location and the correct goal.

To show that this actually occurs in practice, we train agents in the original environment (train env) and evaluate them on a variant of the same (eval env), where some ice (blue) has appeared in the middle of the maze. The ice makes the agent slip from the upper cell to the bottom cell and vice versa. Note that, although the ice does change the environment dynamics, its purpose is to force the agent to take trajectories different from the optimal ones. The way we implemented it, the effect of the ice would be equivalent to forcing the agent to move down twice when in the top cell or move up twice when in the bottom cell. Importantly, these trajectories are feasible in the original environment. The plot on the right of Figure 1 shows the return averaged over 10 trials. The performance drop in the evaluation environment (blue curve) suggests that the agents' policies do not generalize. The ice confuses the agents, who, after being pushed away from their preferred trajectories, can no longer select the right goal. More details about this experiment are provided in Section 6.7.

6.3 Related Work

The presence of spurious correlations in the training data is a well-studied problem in machine learning. These correlations often provide convenient shortcuts that a model can exploit to make predictions (Beery et al., 2018). However, the performance of a model that relies on them may significantly deteriorate under different data distributions (Quionero-Candela et al., 2009; Arjovsky, 2021). Langosco et al. (2022) show that RL agents may use certain environment features as proxies for choosing their actions. These features, which show only in the training environments, happen to be spuriously correlated with the agent's objectives. In contrast, we demonstrate that, as a result of policy confounding, agents may directly take part in the formation of spurious correlations. A few prior works have already reported empirical evidence of particular forms of policy confounding, showing that in deterministic environments, agents can rely on information that correlates with the agent's progress in an episode to determine the optimal actions. This strategy is effective because under fixed policies, features such as timers (Song et al., 2020), agent's postures (Lan et al., 2023), or previous action sequences (Machado et al., 2018) can be directly mapped to the agent's state. These works provide various hypotheses to justify their experimental observations. Here, we contribute an overarching theory that explains the underlying causes and mechanisms behind these results, along with a series of examples

illustrating other types of policy confounding. Please refer to Appendix D.3 for more details on related work.

6.4 Preliminaries

In this section, we introduce the notation used throughout the chapter and provide the mathematical formulation of the problem. We focus again on problems where states are represented by a set of observation variables. (Boutilier et al., 1999).

Definition 6.4.1 (FMDP)**.** A Factored Markov decision process (FMDP) is an MDP (Definition 2.1.1) where the set of states is described by a set of observation variables $\Theta = \{\Theta^1, ..., \Theta^{|\Theta|}\}$. Each variable Θ^i can take any of the values in its domain $\theta^i \in \text{dom}(\Theta^i)$. Hence, every state s corresponds to a different combination of values $\langle \theta^1, ..., \theta^{|\Theta|} \rangle \in \times_i \text{dom}(\Theta^i) = \mathcal{S}$.

The task for the agent involves finding a policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ that maximizes the expected discounted sum of rewards (Sutton and Barto, 2018). Yet, depending on the number of observation variables, learning a policy that conditions on all of them may be infeasible. Fortunately, in many problems, not all variables are strictly relevant; the agent can usually find compact representations of the states, that are sufficient for solving the task.

While, for simplicity, we employ the MDP formulation, the framework we introduce next is not limited to fully observable environments. In cases where the current observation does not satisfy the Markov property, meaning that current observation variables alone are insufficient for predicting state transitions, we address this limitation by augmenting Θ with past action and observation variables. In the extreme case, the agent may need to condition its decisions on the entire history, which itself is guaranteed to satisfy the Markov property (Kaelbling et al., 1998).

6.5 State Representations

As discussed in previous chapters, factored representations are useful because they readily define relationships between states. States can be compared to one another by looking at the individual values the different variables take. Removing some of the variables in Θ has the effect of grouping together those states that share the same values for the

remaining ones. Thus, in contrast with the classical RL framework, which treats states as independent entities, we can define state abstractions at the variable level instead of doing so at the state level (Li et al., 2006).

Definition 6.5.1 (State representation)**.** A state representation is a function $\Phi : \mathcal{S} \to \bar{\mathcal{S}}$, with $\mathcal{S} = \times_i \text{dom}(\Theta^i)$, $\bar{\mathcal{S}} = \times_i \text{dom}(\bar{\Theta}^i)$, and $\bar{\Theta} \subseteq \Theta$.

Intuitively a state representation $\Phi(s_t)$ is a context-specific projection of a state $s \in \mathcal{S} = \times_i \text{dom}(\Theta^i)$ onto a lower dimensional space $\bar{s} = \times_i \text{dom}(\bar{\Theta}^i)$ defined by a subset of its variables, $\bar{\Theta} \subseteq \Theta$. We use $\{s\}^\Phi = \{s' \in \mathcal{S} : \Phi(s') = \Phi(s)\}$ to denote the equivalence class of s under Φ.

6.5.1 Markov State Representations

As noted in Section 6.4, the agent should strive for state representations with few variables. Yet, not all state representations will be sufficient to learn the optimal policy; some may exclude variables that contain useful information for the task at hand.

Definition 6.5.2 (Markov state representation)**.** A state representation $\Phi(s_t)$ is said to be Markov if, for all $s_t, s_{t+1} \in \mathcal{S}$, $a_t \in \mathcal{A}$,

$$R(s_t, a_t) = R(\Phi(s_t), a_t) \quad \text{and} \quad \sum_{s'_{t+1} \in \{s_{t+1}\}^\Phi} T(s'_{t+1} \mid s_t, a_t) = \Pr(\Phi(s_{t+1}) \mid \Phi(s_t), a_t),$$

where $R(\Phi(s_t), a_t) = \{R(s'_t, a_t)\}_{s'_t \in \{s_t\}^\Phi}$ is the reward at any $s'_t \in \{s_t\}^\Phi$.

The above definition is analogous to the notion of bisimulation (Dean and Givan, 1997; Givan et al., 2003) or model-irrelevance state abstraction (Li et al., 2006). Representations satisfying these conditions are guaranteed to be behaviorally equivalent to the original representation. That is, for any given policy and initial state, the expected return (i.e., cumulative reward; Sutton and Barto, 2018) is the same when conditioning on the full set of observation variables Θ or on the Markov state representation Φ.

Definition 6.5.3 (Minimal state representation)**.** A state representation $\Phi^* : \mathcal{S} \to \bar{\mathcal{S}}^*$ with $\bar{\mathcal{S}}^* = \times_i \text{dom}(\bar{\Theta}^{*i})$ is said to be *minimal*, if all other state representations $\Phi : \mathcal{S} \to \bar{\mathcal{S}}$ with $\bar{\mathcal{S}} = \times_i \text{dom}(\bar{\Theta}^i)$ and $\bar{\Theta} \subset \bar{\Theta}^*$, for some $s \in \mathcal{S}$, are not Markov.

In other words, Φ^* is *minimal* when none of the remaining variables can be removed while the representation remains Markov. Hence, we say that a minimal state representation Φ^* is a sufficient statistic of the full set of observation variables Θ.

Definition 6.5.4 (Superfluous variable)**.** Let $\{\bar{\Theta}^*\}_{\cup\Phi^*}$ be the union of variables in all possible minimal state representations. A variable $\Theta^i \in \Theta$ is said to be superfluous, if $\Theta^i \notin \{\bar{\Theta}^*\}_{\cup\Phi^*}$.

6.5.2 π-Markov State Representations

Considering that the agent's policy will rarely visit all states, the notion of Markov state representation seems excessively strict. We now define a relaxed version that guarantees the representation to be Markov when a specific policy π is followed.

Definition 6.5.5 (π-Markov state representation)**.** A state representation $\Phi^\pi(h_t)$ is said to be π-Markov if, for all $s_t, s_{t+1} \in \mathcal{S}^\pi$, $a_t \in \text{supp}(\pi(\cdot \mid s_t))$,

$$R(s_t, a_t) = R^\pi(\Phi^\pi(s_t), a_t) \quad \text{and} \quad \sum_{s'_{t+1} \in \{s_{t+1}\}^\Phi_\pi} T(s'_{t+1} \mid s_t, a_t) = \overset{\pi}{\Pr}(\Phi^\pi(s_{t+1}) \mid \Phi^\pi(s_t), a_t),$$

where $S^\pi \subseteq S$ denotes the set of states visited under π, $R^\pi(\Phi^\pi(s_t), a_t) = \{R(s'_t, a_t)\}_{s'_t \in \{s_t\}^\Phi_\pi}$, $\{s\}^\Phi_\pi = \{s' \in S^\pi : \Phi^\pi(s') = \Phi^\pi(s)\}$, and $\Pr^\pi$ is probability under π.

Definition 6.5.6 (π-minimal state representation)**.** A state representation $\Phi^{\pi*} : \mathcal{S}^\pi \to \bar{\mathcal{S}}^{\pi*}$ with $\bar{\mathcal{S}}^{\pi*} = \times_i \text{dom}(\bar{\Theta}^{\pi*i})$ is said to be *π-minimal*, if all other state representations $\Phi : \mathcal{S}^\pi \to \bar{\mathcal{S}}^\pi$ with $\bar{\mathcal{S}}^\pi = \times_i \text{dom}(\bar{\Theta}^i)$ and $\bar{\Theta} \subset \bar{\Theta}^{\pi*}$, for some $s \in \mathcal{S}^\pi$, are not π-Markov.

6.6 Policy Confounding

We are now ready to describe how and when policy confounding occurs, as well as why we should care, and how we should go about preventing it. The proofs for all theoretical results are deferred to Appendix D.1.

The next result demonstrates that a π-Markov state representation Φ^π requires at most the same variables, and in some cases fewer, than a minimal state representation Φ^*, while still satisfying the Markov conditions for those states visited under π, $s \in S^\pi$.

Proposition 6.6.1. *Let $\mathbf{\Phi}^*$ be the set of all possible minimal state representations, where every $\Phi^* \in \mathbf{\Phi}^*$ is defined as $\Phi^* : \mathcal{S} \to \bar{\mathcal{S}}^*$ with $\bar{\mathcal{S}}^* = \times_i \text{dom}(\bar{\Theta}^{*i})$. For all π and all $\Phi^* \in \mathbf{\Phi}^*$, there exists a π-Markov state representation $\Phi^\pi : \mathcal{S}^\pi \to \bar{\mathcal{S}}^\pi$ with $\bar{\mathcal{S}}^\pi = \times_i \text{dom}(\bar{\Theta}^{\pi i})$ such that for all $s \in \mathcal{S}^\pi$, $\bar{\Theta}^\pi \subseteq \bar{\Theta}^*$. Moreover, there exist cases for which $\bar{\Theta}^\pi$ is a proper subset, $\bar{\Theta}^\pi \neq \bar{\Theta}^*$.*

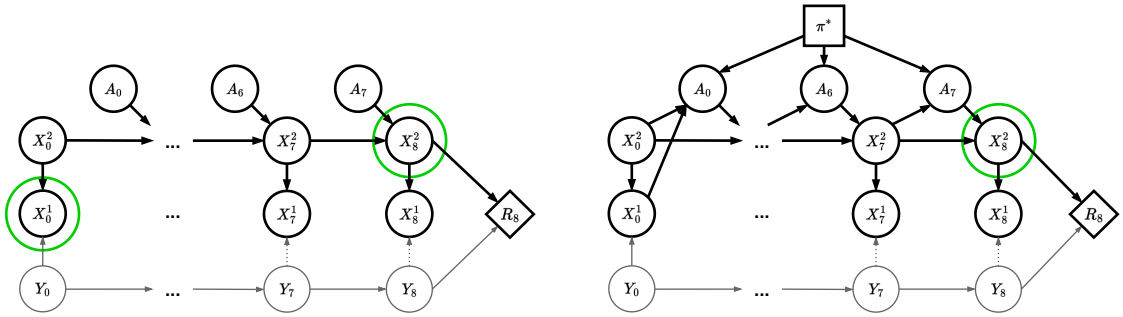


Figure 6.2: Two DBNs representing the dynamics of the Frozen T-Maze environment, when actions are sampled at random (left), and when they are determined by the optimal policy (right).

Although the result above seems intuitive, its truth may appear incidental. While it is clear that Φ^π will never require more variables than the corresponding minimal state representation Φ^*, whether or not Φ^π will require fewer, seems just an arbitrary consequence of the policy being followed. Moreover, since the variables in $\bar{\Theta}^*$ are all strictly relevant for predicting transitions and rewards, one may think that a policy π inducing representations such that $\bar{\Theta}^\pi \subset \bar{\Theta}^*$ can never be optimal. However, as shown by the following example, it turns out that the states visited by a particular policy, especially if it is the optimal policy, tend to contain a lot of redundant information. This is particularly true in environments where future states are heavily influenced by past actions.

Example 6.6.1. (Frozen T-Maze) Let us consider the Frozen T-Maze again (Section 6.2). Figure 6.2 shows two dynamic Bayesian networks (DBN; Dean and Kanazawa, 1989; Murphy, 2002) describing the dynamics of the environment. The nodes labeled as L represent the agent's location from $t = 0$ to $t = 8$. All intermediate nodes between $t = 0$ and $t = 7$ are omitted for simplicity. The nodes labeled as G indicate whether the goal is to go to the green or the purple cell (see Figure 6.1). Note that G always takes the same value at all timesteps within an episode (either green or purple). The information in G is hidden and only passed to the agent at the start location through the node X_0. This makes the environment partially observable. Hence, states are defined as the history of actions, locations, and signal values, $s_t = \langle l_0, x_0, a_0 ..., a_{t-1}, l_t, x_t \rangle$. Most of the information in s_t is irrelevant for predicting transitions and rewards. However, depending on the policy being followed, the agent may be able to ignore more or fewer variables. This can be shown by comparing the two DBNs in Figure 6.2. Let us say that we want to predict the reward R_8 at given the state s_8 at time $t = 8$. On the one hand, if actions are not specified by any particular policy, but simply sampled at random (left DBN), to determine R_8, one needs to know the signal X_0 received at $t = 0$ and the agent's current location

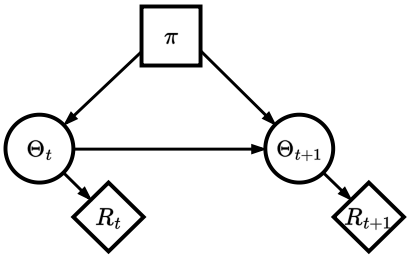


Figure 6.3: A DBN illustrating the phenomenon of policy confounding. The policy opens backdoor path that can affect conditional relations between the variables in Θ_t and Θ_{t+1}.

L_8. These are highlighted by the green circles. This is because the actions $\langle A_0, ..., A_7 \rangle$ appear as exogenous variables and can take any possible value. Hence, the reward could be either -0.1, (per timestep penalty), -1 (wrong goal), or $+1$ (correct goal) depending on the actual values of X_0 and L_8. On the other hand, when actions are sampled from the optimal policy π^* (right DBN), knowing L_8 (green circle) is sufficient to determine R_8. In this second case, π^* makes the action A_0, and thus all future agent locations, dependent on the initial signal X_0. This occurs because, under the optimal policy (green and purple paths in Figure 6.1), the agent always takes the action 'move up' when receiving the green signal or 'move down' when receiving the purple signal, and then follows the shortest path towards each of the goals. As such, we have that, from $t = 1$ onward, $\Phi^{\pi^*}(s_t) = l_t$ is a π-Markov state representation since it constitutes a sufficient statistic of the state s_t under π^*. Analogously, from $t = 1$, actions may also condition only on L.

The phenomenon highlighted by the previous example is the result of a spurious correlation induced by the optimal policy between the initial signal X_0 and the agent's future locations $\langle L_1, ..., L_8 \rangle$. Generally speaking, this occurs because policies act as confounders, opening backdoor paths between future state variables Θ_{t+1} and the variables in the current state Θ_t (Pearl, 2000). This is shown by the DBN depicted in Figure 6.3, where we see that the policy influences both the current observation variables and future observation variables, hence potentially affecting their conditional relationships. For instance, in the above example, $R^{\pi^*}(L_8 = \text{'green goal'}) = +1$ when following π^*, while for an arbitrary π, $R(L_8 = \text{'green goal'}) = \pm 1$.

Definition 6.6.1 (Policy Confounding)**.** A state representation $\Phi : \mathcal{S} \to \bar{\mathcal{S}}$ is said to be confounded by a policy π if, for some $s_t, s_{t+1} \in \mathcal{S}$, $a_t \in \mathcal{A}$,

$$R^{\pi}(\Phi(s_t), a_t) \neq R^{\pi}(\text{do}(\Phi(s_t)), a_t) \quad \text{or} \quad \overset{\pi}{\Pr}(\Phi(s_{t+1}) \mid \Phi(s_t), a_t) \neq \overset{\pi}{\Pr}(\Phi(s_{t+1}) \mid \text{do}(\Phi(s_t)), a_t)$$

The operator $\text{do}(\cdot)$ is known as the do-operator, and it is used to represent physical interventions in a system (Pearl, 2000). These interventions are meant to distinguish cause-effect relations from mere statistical associations. In our case, $\text{do}(\Phi(s_t))$ means setting the variables forming the state representation $\Phi(s_t)$ to a particular value and considering all possible states in the equivalence class, $s'_t \in \{s_t\}^{\Phi}$. That is, independently of what policy is being followed.

It is easy to show that the underlying reason why a π-Markov state representation may require fewer variables than the minimal state representation (as in Example 6.6.1) is indeed policy confounding.

Theorem 6.6.1. *Let $\Phi^* : \mathcal{S} \to \bar{\mathcal{S}}^*$ with $\bar{\mathcal{S}}^* = \times_i \text{dom}(\bar{\Theta}^{*i})$ be a minimal state representation. If, for some π, there is a π-Markov state representation $\Phi^\pi : \mathcal{S}^\pi \to \bar{\mathcal{S}}^\pi$ with $\bar{\mathcal{S}}^\pi = \times_i \text{dom}(\bar{\Theta}^{\pi i})$, such that $\bar{\Theta}^\pi \subset \bar{\Theta}^*$ for some $s \in \mathcal{S}$, then Φ^π is confounded by policy π.*

Finally, it is worth noting that even though, in Example 6.6.1, the variables included in the π-minimal state representation are a subset of the variables in the minimal state representation, $\bar{\Theta}^{\pi*} \subset \bar{\Theta}^*$, this is not always the case, as $\bar{\Theta}^{\pi*}$ may contain superfluous variables (Definition 6.5.4). An example illustrating this situation is provided in Appendix D.2 (Example D.2.1).

Proposition 6.6.2. *Let $\{\bar{\Theta}^*\}_{\cup\Phi^*}$ be the union of variables in all possible minimal state representations. There exist cases where, for some π, there is a π-minimal state representation $\Phi^{\pi*} : \mathcal{S}^\pi \to \bar{\mathcal{S}}^{\pi*}$ with $\bar{\mathcal{S}}^{\pi*} = \times_i \text{dom}(\bar{\Theta}^{\pi*i})$ such that $\bar{\Theta}^{\pi*} \setminus \{\bar{\Theta}^*\}_{\cup\Phi^*} \neq \emptyset$.*

6.6.1 Why Should We Care about Policy Confounding?

Leveraging spurious correlations to develop simple habits can be advantageous when resources such as memory, computing power, or data are limited. Agents can disregard and exclude from their representation those variables that are redundant under their policies. However, the challenge is that some of these variables may be crucial to ensure that the agent behaves correctly when the context changes. In the Frozen T-Maze example from Section 6.2, we observed how the agent could no longer find the correct goal when the ice pushed it away from the optimal trajectory. This is a specific case of a well-researched issue known as out-of-distribution (OOD) generalization (Quionero-Candela et al., 2009; Arjovsky, 2021). We refer to it as *out-of-trajectory* (OOT) generalization to highlight that

the problem arises due to repeatedly sampling from the same policy and thus following the same trajectories. In contrast to previous works (Kirk et al., 2023) that address generalization to environments that differ from the training environment, our objective here is to generalize to trajectories the agent never (or only rarely) takes.

Ideally, the agent should aim to learn representations that enable it to predict future rewards and transitions even when experiencing slight variations in its trajectory. Based on Definition 6.5.2, we know that, in general, only a Markov state representation satisfies these requirements. However, computing such representations is typically intractable (Ferns et al., 2006), and thus most standard RL methods usually learn representations by maximizing an objective function that depends on the distribution of trajectories $P^b(\tau)$ visited under a behavior policy b (e.g., expected return, $\mathbb{E}_{\tau \sim P^b(\tau)}[G(\tau)]$; Sutton and Barto, 2018). The problem is that b may favor certain trajectories over others, which may lead to the exploitation of spurious correlations in the learned representation.

6.6.2 When Should We Worry about OOT Generalization in Practice?

The previous section highlighted the generalization failures of representations that depend on spurious correlations. Now, let us delve into the circumstances in which policy confounding is most prone to cause problems.

Function Approximation Function approximation has enabled traditional RL methods to scale to high-dimensional problems, where storing values in lookup tables is infeasible. Using parametric functions (e.g., neural networks) to model policies and value functions, agents can learn abstractions by grouping together states if these yield the same transitions and rewards. As mentioned before, abstractions occur naturally when states are represented by a set of variables since the functions simply need to ignore some of these variables. However, this also implies that value functions and policies are exposed to spurious correlations. If a particular variable becomes irrelevant due to policy confounding, the function may learn to ignore it and remove it from its representation (Example 6.6.1). This is in contrast to tabular representations, where, every state takes a separate entry, and even though there exist algorithms that perform state (state) abstractions in tabular settings (Andre and Russell, 2002; Givan et al., 2003), these abstractions are normally formed offline before learning (computing) the policy, hence avoiding the risk of policy confounding.

Narrow Trajectory Distributions In practice, agents are less prone to policy confounding when the trajectory distribution $P^b(\tau)$ is broad (i.e., when b encompasses a wide set of trajectories) than when it is narrow. This is because the spurious correlations present in certain trajectories are less likely to have an effect on the learned representations. On-policy methods (e.g., SARSA, Actor-Critic; Sutton and Barto, 2018) are particularly troublesome for this reason since the same policy being updated must also be used to collect the samples. Yet, even when the trajectory distribution is narrow, there is no reason why the agent should pick up on spurious correlations while its policy is still being updated. Only when the agent commits to a particular policy should we start worrying about policy confounding. At this point, lots of the same trajectories are being used for training, and the agent may *'forget'* (French, 1999) that, even though certain variables may no longer be needed to represent the states, they were important under previous policies. This generally occurs at the end of training when the agent has converged to a particular policy. However, if policy confounding occurs earlier during training, it may prevent the agent from further improving its policy (Nikishin et al., 2022; please refer to Appendix D.3 for more details).

6.6.3 What Can We Do to Improve OOT Generalization?

As mentioned in the introduction, we do not have a complete answer to the problem of policy confounding. Yet, here we offer a few off-the-shelf solutions that, while perhaps limited in scope, can help mitigate the problem in some situations. These solutions revolve around the idea of broadening the distribution of trajectories so as to dilute the spurious correlations introduced by certain policies.

Off-policy methods We already explained in Section 6.6.2 that on-policy methods are particularly prone to policy confounding since they are restricted to using samples coming from the same policy. A rather obvious solution is to instead use off-policy methods, which allow using data generated from previous policies. Because the samples belong to a mixture of policies it is less likely that the model will pick up the spurious correlations present on specific trajectories. However, as we shall see in the experiments, this alternative works only when replay buffers are large enough. This is because standard replay buffers are implemented as queues, and hence the first experiences coming in are the first being removed. This implies that a replay buffer that is too small will contain samples coming from few and very similar policies. Since there is a limit on how large replay buffers are

allowed to be, future research could explore other, more sophisticated, ways of deciding what samples to store and which ones to remove (Schaul et al., 2016b).

Exploration and domain randomization When allowed, exploration may mitigate the effects of policy confounding and prevent agents from overfitting their preferred trajectories. Exploration strategies have already been used for the purpose of generalization; to guarantee robustness to perturbations in the environment dynamics (Eysenbach and Levine, 2022), or to boost generalization to unseen environments (Jiang et al., 2022). The goal for us is to remove, to the extent possible, the spurious correlations introduced by the current policy. Unfortunately, though, exploration is not always without cost. Safety-critical applications require the agent to stay within certain boundaries (Altman, 1999; García and Fernández, 2015). When training on a simulator, an alternative to exploration is domain randomization (Tobin et al., 2017; Peng et al., 2018; Machado et al., 2018). The empirical results reported in the next section suggest that agents become less susceptible to policy confounding when adding enough stochasticity to the environment or to the policy. Yet, there is a limit on how much noise can be added to the environment or the policy without altering the optimal policy (Sutton and Barto, 2018, Example 6.6: Cliff Walking).

6.7 Experiments

The goal of the experiments is to: (1) demonstrate that the phenomenon of policy confounding described by the theory does occur in practice, (2) uncover the circumstances under which agents are most likely to suffer the effects of policy confounding and fail to generalize, and (3) evaluate how effective the strategies proposed in the previous section are in mitigating these effects.

6.7.1 Experimental setup

Agents are trained with an off-policy method, DQN (Mnih et al., 2015) and an on-policy method, PPO (Schulman et al., 2017). We represent policies and value functions as feedforward neural networks and use a stack of past observations as input in the environments that require memory. We report the mean return as a function of the number of training steps. Training is interleaved with periodic evaluations on the original

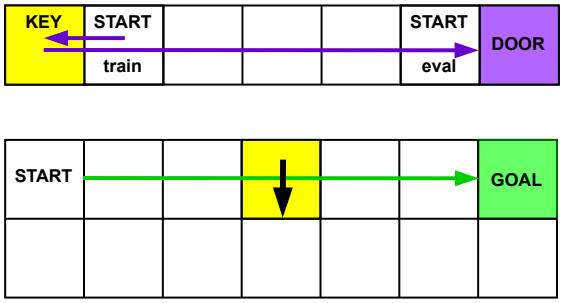


Figure 6.4: Illustrations of the Key2Door (top) and Diversion (bottom) environments.

environments and variants thereof used for validation. The results are averaged over 10 random seeds. Please refer to Appendix D.6 for more details about the setup.

6.7.2 Environments

We ran our experiments on three grid-world environments: the **Frozen T-Maze** from Section 6.2, and the below described **Key2Door**, and **Diversion** environments. We use these as pedagogical examples to clarify the ideas introduced by the theory. Nonetheless, in Appendix D.3, we refer to previous works showing evidence of particular forms of policy confounding in high dimensional domains.

Example 6.7.1. Key2Door. Here, the agent needs to collect a key placed at the beginning of the corridor in Figure 6.4 (left) and then open the door at the end. The current observation variables do not show whether the key has already been collected. The states are thus given by the history of past locations $s_t = \langle l_0, ..., l_t \rangle$. This is because to solve the task in the minimum number of steps, the agent must remember that it already got the key when going to the door. Yet, since during training, the agent always starts the episode at the first cell from the left, when moving towards the door, the agent can forget about the key (i.e., ignore past locations) once it has reached the third cell. As in the Frozen T-Maze example, the agent can build the habit of using its own location to tell whether it has or has not got the key yet. This, can only occur when the agent consistently follows the optimal policy, depicted by the purple arrow. Otherwise, if the agent moves randomly through the corridor, it is impossible to tell whether the key has or has not been collected. In contrast, in the evaluation environment, the agent always starts at the second to last cell, this confuses the agent, which is used to already having the key by the time it reaches said cell. A DBN is provided in Appendix D.4.

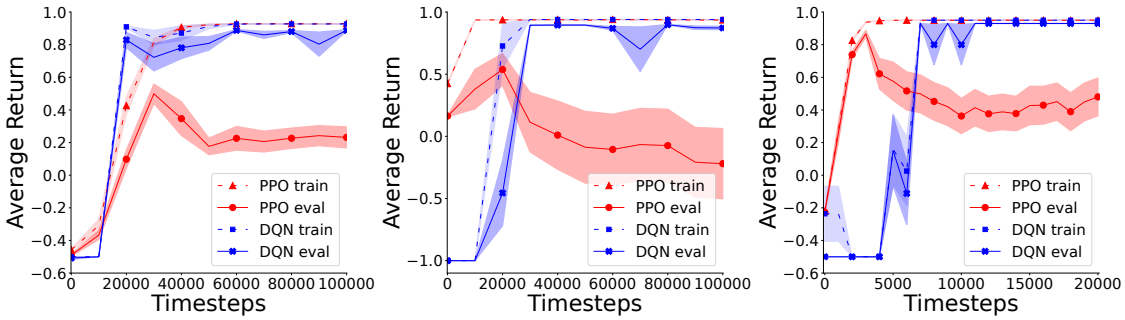


Figure 6.5: DQN vs. PPO in the train and evaluation variants of Frozen T-Maze (left), Key2Door (middle), and Diversion (right).

Example 6.7.2. Diversion. Here, the agent must move from the start state to the goal state in Figure 6.4 (right). The observations are length-8 binary vectors. The first 7 elements indicate the column where the agent is located. The last element indicates the row. This environment aims to show that policy confounding can occur not only when the environment is partially observable, as was the case in the previous examples, but also in fully observable scenarios. After the agent learns the optimal trajectory depicted by the green arrow, it can disregard the last element in the observation vector. This is because, if the agent does not deviate, the bottom row is never visited. Rather than forgetting past information, the agent ignores the last element in the current observation vector for being irrelevant when following the optimal trajectory. We train the agent in the original environment and evaluate it in a version with a yellow diversion sign in the middle of the maze that forces the agent to move to the bottom row. A DBN is provided in Appendix D.4.

Note that, as we did in the Frozen T-Maze environment, in order to assess whether the learned state representations generalize beyond the agent's usual trajectories we must modify the environment dynamics so as to force the agents to take alternative trajectories. Importantly, these alternative trajectories are both possible and probable in the original environments, and thus one would expect well-trained agents to perform well on them.

6.7.3 Results

On-policy vs. off-policy The results in Figure 6.5 reveal the same pattern in all three environments. PPO fails to generalize outside the agent's preferred trajectories. After an initial phase where the average returns on the training and evaluation environments

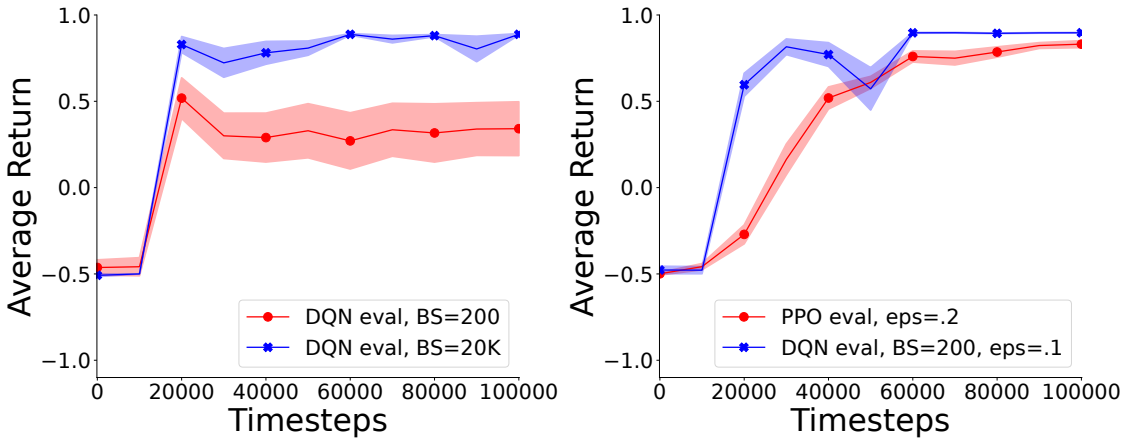


Figure 6.6: Frozen T-Maze. Left: DQN small vs. large buffer sizes. Right: PPO and DQN when adding stochasticity.

increase ('PPO train' and 'PPO eval'), the return on the evaluation environments ('PPO eval') starts decreasing when the agent commits to a particular trajectory, as a result of policy confounding. In contrast, since the training samples come from a mixture of policies, DQN performs optimally in both variants of the environments ('DQN train' and 'DQN eval') long after converging to the optimal policy.[3] A visualization of the state representations learned with PPO, showing that the policy does ignore variables that are necessary for generalization, is provided in Appendix D.5.1.

Large vs. small replay buffers We mentioned in Section 6.6.3 that the effectiveness of off-policy methods against policy confounding depends on the size of the replay buffer. The results in Figure 6.6 (left) confirm this claim. The plot shows the performance of DQN in the Frozen T-Maze environment when the size of the replay buffer contains 100K experiences and when it only contains the last 10K experiences. We see that in the second case, the agents performance in the evaluation environment decreases (red curve left plot). This is because, after the initial exploration phase, the distribution of trajectories becomes too narrow, and the spurious correlations induced by the latest policies dominate the replay buffer. Similar results for the other two environments are provided in Appendix D.5.2.

[3]The small gap between 'DQN train' and 'DQN eval' is due to the -0.1 penalty per timestep. In all three environments, the shortest path is longer in the evaluation environment than in the training environment.

Exploration and domain randomization The last experiment shows that if sufficient exploration is allowed, DQN may still generalize to different trajectories, even when using small replay buffers (blue curve right plot Figure 6.6). In the original configuration, the exploration rate ϵ for DQN starts at $\epsilon = 1$ and decays linearly to $\epsilon = 0.0$ after 20K steps. For this experiment, we set the final exploration rate $\epsilon = 0.1$. In contrast, since exploration in PPO is normally controlled by the entropy bonus, which makes it hard to ensure fixed exploration rates, we add noise to the environment instead. The red curve in Figure 6.6 (right) shows that when we train in an environment where the agent's actions are overridden by a random action with 20% probability, the performance of PPO in the evaluation environment does not degrade after the agent has converged to the optimal policy. This suggests that the added noise prevents the samples containing spurious correlations from dominating the training batches. However, it may also happen that random noise is not sufficient to remove the spurious correlations. As shown in Figure D.7 (Appendix D.5.2), in the Key2Door environment, neither forcing the agent to take random actions 20% of the time nor setting $\epsilon = 0.1$, solves the OOT generalization problem. Similar results for Diversion are provided in Appendix D.5.2.

6.8 Conclusion

This paper described the phenomenon of policy confounding. We showed both theoretically and empirically how as a result of following certain trajectories, agents may pick up on spurious correlations, and build habits that are not robust to trajectory deviations. We also uncovered the circumstances under which policy confounding is most likely to occur in practice and suggested a few ad hoc solutions that may mitigate its effects. We conceive this paper as a stepping stone to explore more sophisticated solutions. An interesting avenue for future research is the integration of tools from the field of causal inference (Pearl et al., 2016; Peters et al., 2017) to aid the agent in forming state representations that are grounded on causal relationships rather than mere statistical associations (Lu et al., 2018; Zhang et al., 2020; Sontakke et al., 2021; Saengkyongam et al., 2023).

Chapter 7

Concluding Remarks

I always do that, get into something, and see how far I can go.

–Richard P. Feynman, 'Surely You're Joking, Mr. Feynman!'

In this chapter, we provide a summary of the main takeaways of this thesis, discuss the limitations of our work, and set the direction for future research.

7.1 Conclusion

This thesis has advocated for the use of factored representations in RL problems. We have provided several examples highlighting the structural insights and efficiency advantages that these representations offer.

In Chapter 3, we showed how by exploiting the structure in certain problems, we could build lightweight local simulators that capture the influence of the global system on the local region through machine learning models. Experimental results showed that agents trained on these local simulators achieve performance comparable to those trained on exact global simulators.

Expanding on this approach, Chapter 4 illustrated how large systems with multiple interacting elements can be factorized into independent regions. The experiments showed that this approach, aside from providing significant computational benefits, also facilitates

the convergence of multi-agent RL.

Chapter 5 showed how, in partially observable environments, we could determine what particular variables the agent should memorize by analyzing the dependencies between the hidden state variables and the variables in the action-observation histories, leading to improved learning speed and convergence.

Lastly, in Chapter 6, we investigated the types of representations necessary for predicting future transitions and rewards under specific policies. We demonstrated that methods focused on maximizing expected returns yield representations insufficient for generalization to trajectories different from those that the agent usually follows.

Despite falling short of fully exploring the potential of factored representations, this thesis contributes to showcasing how we can gain valuable insights and a better understanding of the RL framework by examining the variable dependencies in factored representations.

7.2 Scope

The concepts and methods outlined in this thesis demonstrate good promise in enhancing the efficiency and scalability of RL. There is a compelling basis for optimism regarding the potential improvement of real-world system performance. It is important to note, however, that the assessment of these approaches has been conducted exclusively within synthetic environments characterized by simplified dynamics and structures. While these environments have been designed to emulate real-world challenges in domains such as traffic control and warehouse commissioning, it is essential to emphasize that these simulations only bear a limited resemblance to the complexities and nuances inherent in realistic scenarios (Geroliminis and Daganzo, 2008; Knoop et al., 2015). Hence, the results obtained should not be automatically extrapolated to the real world. The translation of these methods into tangible products for deployment in real systems requires addressing a host of limitations, some of which are discussed in the next section. For a comprehensive review of strategies for real road traffic control, the interested reader is referred to Papageorgiou et al. (2003); Qadri et al. (2020).

7.3 Limitations

Below we outline some of the limitations of our methods. Please note that the list is not exhaustive.

1. **High-level representations:** The insights presented in this thesis require a sufficiently high-level representation of the problem. Our work assumes this is already given, but features or symbols describing the environment may not be available in many cases. Thus, our methods' future impact and applicability to practical problems depend heavily on research in this area (Lopez-Paz et al., 2017; Schölkopf and von Kügelgen, 2022). Moreover, factored representations alone are not sufficient to examine the environment dynamics. In general, we also need to be provided the dependencies between variables in the form of a DBN. Some readers may question the use of DBNs in analyzing RL problems, as it contradicts one of the fundamental premises of RL, namely that the environment dynamics are not known in advance. However, the methods presented here can still be applied even if the dynamics are not fully known (i.e., conditional probabilities are not specified). In general, high-level DBNs showing the dependencies between variables are sufficient. Fortunately, in many cases these can be constructed with some domain knowledge of the environment. Alternatively, one can also resort to methods for learning DBNs (Jonsson and Barto, 2007; Daly et al., 2011).

2. **Well-defined local regions** The main assumption we made in Chapters 3 and 4 was that the agents' observations and rewards depended solely on the subset of state variables within their local regions. This limitation affects their performance in two ways. First, the information the agents receive may be insufficient for making good decisions, resulting in suboptimal policies. Second, the reward function may not align with the true objective due to its narrow focus. In the single-agent scenario, local performance improvements could adversely impact unaccounted regions of the environment. In the multi-agent scenario, it may lead to competition among agents. The solution, of course, is to enlarge the local regions. However, there is a limit to how much the local regions can grow while maintaining the approach's effectiveness. Another related limitation is that, for DBNs to represent partially observable problems (Section 2.1.3), it is essential that the agent's observations depend on the same group of state variables at all possible states. Moreover, interactions between different regions should occur through a limited number of

variables. Fortunately, many domains exhibit this particular structure, including traffic, cooling systems, logistics, and telecommunications.

3. **Stationary dynamics:** As is typically the case in RL research, IALS (Chapter 3) and DIALS (Chapter 4) assume that the dynamics of the global environment are stationary, which is generally not true in most real-world problems. Several external factors may come into play and affect local regions. Unfortunately, neural networks are not the most reactive type of models. They tend to overfit the training data and are often unable to learn new patterns.

4. **Off-Policy Generalization** As seen in Chapter 3, training low-capacity influence predictors on finite datasets with limited support over the state-action space can lead to models that overfit to spurious correlations induced by the exploratory policy used to collect the data (an example of this is provided in A.2). The solution we adopted was to feed into the influence predictor only the necessary information. However, even then, it is not guaranteed that the influence model will perform well when tested under other policies.

5. **Policy confounding and generalization** Finally, our work on policy confounding (Chapter 6) raises important questions that challenge standard RL methods. We discussed the phenomenon from a theoretical perspective and provided examples demonstrating its effects on the learned representations. We also presented a few ad-hoc solutions to this problem, but these were somewhat limited in scope.

7.4 Future Work

7.4.1 Adaptive Simulators for Non-Stationary Environments

As mentioned in the previous section, one of the limitations of our work on influence-augmented local simulators (IALS; Chapter 3) is that the approximate influence predictors (AIPs) are trained offline on a dataset of historical experiences collected off-line with some exploratory policy. This assumes that the dynamics of the global environment are stationary. However, we already saw in Chapter 4 that this was not the case when other agents were learning simultaneously, forcing us to retrain the AIPs every so often. Moreover, not only are the other agents' policies subject to change, but external hidden factors could also affect the dynamics (Feng et al., 2022). For instance, in the traffic

control scenario, traffic densities may vary depending on the time of day, there might be road closures due to construction works, and events such as concerts or sports games may also impact traffic patterns.

One solution is to continuously update the simulator as we gather new data from the environment using cameras or sensors. In our previous work (He et al., 2022), we demonstrated how the AIPs could be learned online while planning in the environment. However, the dynamics were assumed to be stationary, and the new data was merely used to enhance the old AIPs. Under non-stationarity, old AIPs trained with past data become increasingly outdated and biased as time goes by since, strictly speaking, only the most recent data can be trusted. Therefore, we require models that can quickly adapt and capture the new dynamics. One natural way of achieving this is to use Kalman filters (Kalman, 1960) to update the parameter estimates as new data comes in. See Wang and Papageorgiou (2005) and van Lint and Djukic (2012) for examples of traffic state estimation using Kalman filters. More sophisticated methods such as Gaussian processes (Williams and Rasmussen, 2006) may also be useful for this purpose since they make very few assumptions about the underlying distribution, measure the prediction uncertainty, and can be computed swiftly as long as the number of data points is not too large.

Another alternative is to learn various predictive models for different contexts. Naturally, this assumes that data for each context is available during training. If so, one could, for instance, train separate neural networks for each context and use them interchangeably. Yet, assuming that the dynamics stay relatively similar across contexts, rather than training completely independent networks, it may be advantageous to allow some parameter sharing for enhanced training speed and overall accuracy. This can be achieved by utilizing hypernetworks, wherein a network, fed with context-specific information is used to generate some of the weights of another network, known as the target network, which makes the final predictions. This architecture allows the target network to dynamically adapt and respond differently to the same input depending on the context. See Beukman et al. (2023) for an example of the use of hypernetworks in the context of RL and Li et al. (2021a) for an example in the context of traffic forecasting.

7.4.2 Generalization in Dynamics Modeling

Developing predictive models of dynamical systems solely based on historical data raises concerns when utilizing the model for system optimization. To effectively guide system

optimization, the model must accurately capture the consequences of executing actions at every conceivable state. However, conventional machine learning techniques are unable to guarantee consistent performance across regions of the state space that are inadequately represented in the training data. Models tend to show overconfidence in low-data regions even when there is high aleatoric uncertainty (Depeweg et al., 2018; Li et al., 2022). Obtaining a well-balanced dataset that uniformly covers all regions, particularly optimal ones, is impractical, as current policies are not optimal (otherwise, we would not be seeking to improve them).

One solution is to enforce established physical properties of the environment into the models, either through the objective function in the form of penalties, or through the architecture, by introducing inductive biases (Karniadakis et al., 2021). The primary objective is to guide the behavior of the models in the low-data regions. Refer to Van Lint et al. (2005); Huang and Agarwal (2020) for examples of this approach in the context of traffic prediction.

The above techniques will likely mitigate the problem but additional safeguards are necessary. Note that, the underlying challenge originates from deploying the model within a different data distribution than the one on which it was trained. Specifically, the model is trained on the distribution of trajectories induced by the policy followed when generating the data, yet it is queried with trajectories sampled from different policies. Consequently, the model may unintentionally capture spurious correlations and struggle to generalize effectively. This phenomenon, which we named policy confounding, was previously discussed in Chapter 6 in the context of its impact on state representations. Notably, dynamics models are also susceptible to the effects of policy confounding.

If the system structure is known, meaning we can derive DBN describing the variable relationships, one can employ causal inference techniques (Pearl et al., 2016; Peters et al., 2017), such as inverse probability weighting or matching, to control for confounding. The goal is to identify the genuine causal effects of specific actions to ensure that our dynamic models operate accurately when queried about state-action pairs that are not adequately represented in the dataset.

7.4.3 Causal Reinforcement Learning

Preventing policy confounding, while simultaneously exploring the environment and improving the policy, as it is the case in RL, introduces several new challenges but also

opportunities. All of the solutions proposed in Chapter 6 revolve around the idea of broadening the distribution of states so as to dilute the spurious correlations introduced by certain policies. While these solutions might be effective in some cases, they somewhat avoid the actual problem. Going forward, agents should aim to directly capture the causal structure of the environment and be agnostic to the spurious correlations introduced by their policies.

Causal inference approaches are naturally divided into two categories: passive or observational and active or interventional. We describe them below and explain how they can be adjusted to RL.

Passive Passive methods try to uncover the causal structure of a system directly from observational data (Mooij et al., 2016). In this setting, there is normally data from a few different distributions (resulting from different interventions on the system), and the task is to learn a model that generalizes to a set of test distributions. Perhaps the most popular methods in this category are those that exploit prediction invariance (Peters et al., 2016; Arjovsky et al., 2019; Mooij et al., 2020; Krueger et al., 2021). These are based on the observation that causal models should be invariant to interventions on non-causal variables. It turns out that this is also the case in RL; causal state representations should be invariant across policies. Translating this observation into a practical algorithm implies adding constraints or penalties to the objective function to ensure that the learned representations are not biased toward a particular policy (Zhang et al., 2020).

Active In this setting, we are allowed to intervene in the system to determine which variables are causal predictors and which are merely statistically associated (Peters et al., 2017). The RL setting offers a natural way to intervene in the environment through the agent's actions (Seitzer et al., 2021; Sontakke et al., 2021; Lippe et al., 2023). As Professor Schölkopf, argues in a recent paper (Schölkopf et al., 2021), "(in RL), agents can reflect on their decisions and formulate hypotheses that can be empirically verified." Hence, causal inference can be added as one of the agent's exploration objectives. Agents can stress test their representations to ensure they are grounded on causal relations rather than spurious correlations. This is in contrast to the standard RL formulation, where agents simply explore to maximize the return.

Appendices

Appendix A

Influence-Augmented Local Simulators

A.1 Proofs

Theorem 3.3.1. *Let $l_{t-k:t}$ be a truncated version of l_t containing only the last k action-local-state pairs and let $a_{0:t-k}$ be the sequence of past actions from time 0 up to k. Let the agent's policy $\bar{\pi}$ be a function of $h_{t-k:t}$ $\bar{\pi}(a_t|h_{t-k:t})$, where $h_{t-k:t}$ is also a truncated version of h_t. If for all $u_t, a_{0:t-k}$ we have $P(u_t|l_{t-k:t}, a_{0:t-k}) = P(u_t|l_{t-k:t})$, then an influence predictor that conditions only on $l_{t-k:t}$, $\bar{I}(u_t|l_{t-k:t})$, is sufficient to guarantee no loss in value.*[1]

Proof. We will prove that, when conditioning on the last k elements of the AOH, $h_{t-k:t}$, the action-value function $\bar{Q}^{\bar{\pi}}(h_{t-k:t}, a_t)$, can be expressed in terms of an influence predictor that conditions on $l_{t-k:t}$, $\bar{I}(u_t \mid l_{t-k:t})$. If this is true, then Q estimates can be obtained by sampling from the finite memory IALS (with influence predictor $\bar{I}(u_t \mid l_{t-k:t})$) and the agent's optimal policy (in the class of finite memory policies of the form $\bar{\pi}(a_t \mid h_{t-k:t})$) can be found via policy iteration Sutton and Barto (1998). Note that, finite memory policies have been shown to perform arbitrarily bad compared to their full memory counterparts Singh et al. (1994b). Moreover, empirical results have revealed that standard RL methods for MDPs may fail when state representations are not Markovian Littman

[1] Note that this refers only to the value of polices of the form $\bar{\pi}(a_t|h_{t-k:t})$, which might perform arbitrarily worse than policies that condition on the full AOH, $\pi(a_t|h_t)$ (Singh et al., 1994b).

(1994). Nonetheless, here we are only concerned with whether or not we can find the same finite memory policies when using influence predictors that condition only on $h_{t-k:t}$ as those found with influence predictors that condition on the full AOH. That is, independently of the effectiveness of the RL method we use or the absolute performance of the policies we find.

Given the full AOH, the action-value function $Q^\pi(h_t, a_t)$ of a local-FPOMDP satisfies the Bellman equation and can be expressed as

$$Q^\pi(h_t, a_t) = \sum_{x_t} P(x_t \mid h_t)\dot{R}(x_t, a_t) + \sum_{l_t} P(l_t \mid h_t) \sum_{x_{t+1}} P(o_{t+1} \mid l_t, a_t)\Big[\tag{A.1}$$

$$\sum_{a_{t+1}} \pi(a_{t+1} \mid h_{t+1}) Q^\pi(\langle h_t, a_t, o_{t+1}\rangle, a_{t+1})\Big], \tag{A.2}$$

where and $P(x_t \mid h_t)$ is the probability of the local states x_t given the previous AOH, and from (2.8)

$$P(o_{t+1} \mid l_t, a_t) = \sum_{x_{t+1}} \dot{O}(o_{t+1} \mid x_{t+1}) \sum_{u_t} \dot{T}(x_{t+1} \mid x_t, a_t, u_t) I(u_t \mid l_t) \tag{A.3}$$

If, instead, the agent's policy $\bar{\pi}$, and in turn the $\bar{Q}^{\bar{\pi}}$ function, can condition only on the last k elements in the AOH $h_{t-k:t}$, we have

$$\bar{Q}^{\bar{\pi}}(h_{t-k:t}, a_t) = \sum_{h_{0:t-k}} P^{\bar{\pi}}(h_{0:t-k} \mid h_{t-k:t}) Q^{\bar{\pi}}(h_t = \langle h_{0:t-k}, h_{t-k:t}\rangle, a_t),$$

where $h_{0:t-k}$ is the AOH from 0 to k and the above expression is just the expected Q value given $h_{t-k:t}$. Then, using (A.2)

$$\begin{aligned}\bar{Q}^{\bar{\pi}}(h_{t-k:t}, a_t) = &\sum_{h_{0:t-k}} P^{\bar{\pi}}(h_{0:t-k} \mid h_{t-k:t}) \sum_{x_t} P(x_t \mid h_{0:t-k}, h_{t-k:t}) \dot{R}(x_t, a_t) \\ &+ \sum_{h_{0:t-k}} P^{\bar{\pi}}(h_{0:t-k} \mid h_{t-k:t}) \sum_{l_t} P(l_t \mid h_{0:t-k}, h_{t-k:t}) \sum_{o_{t+1}} P(o_{t+1} \mid l_t, a_t)\Big[\\ &\qquad \sum_{a_{t+1}} \bar{\pi}(a_{t+1} \mid h_{-k+1:t+1}) \bar{Q}^{\bar{\pi}}(h_{-k+1:t+1}, a_{t+1})\Big]\end{aligned}$$

and from (A.3) we have

$$\sum_{h_{0:t-k}} P^{\bar{\pi}}(h_{0:t-k} \mid h_{t-k:t}) \sum_{l_t} P(l_t \mid h_{0:t-k}, h_{t-k:t}) \sum_{o_{t+1}} P(o_{t+1} \mid l_t, a_t)$$
$$= \sum_{o_{t+1}} \sum_{x_{t+1}} \dot{O}(o_{t+1} \mid x_{t+1}) \sum_{u_t} \sum_{l_t} \dot{T}(x_{t+1} \mid x_t, a_t, u_t) I(u_t \mid l_t) \Big[\sum_{h_{0:t-k}} P^{\bar{\pi}}(h_{0:t-k} \mid h_{t-k:t}) P(l_t \mid h_{0:t-k}, h_{t-k:t}) \Big].$$

Then, using the rule of conditional probability, the last summation can be simplified as

$$\sum_{h_{0:t-k}} P^{\bar{\pi}}(h_{0:t-k} \mid h_{t-k:t}) P(l_t \mid h_{0:t-k}, h_{t-k:t}) = \sum_{h_{0:t-k}} P^{\bar{\pi}}(l_t, h_{0:t-k} \mid h_{t-k:t}) = P^{\bar{\pi}}(l_t \mid h_{t-k:t})$$

and thus

$$\begin{aligned}
&\sum_{l_t} \dot{T}(x_{t+1} \mid x_t, a_t, u_t) I(u_t \mid l_t) P^{\bar{\pi}}(l_t \mid h_{t-k:t}) \\
&= \sum_{l_{0:t-k}, l_{t-k:t}} \dot{T}(x_{t+1} \mid x_t, a_t, u_t) I(u_t \mid l_{0:t-k}, l_{t-k:t}) P^{\bar{\pi}}(l_{0:t-k}, l_{t-k:t} \mid h_{t-k:t}) \\
&= \sum_{l_{0:t-k}, l_{t-k:t}} \dot{T}(x_{t+1} \mid x_t, a_t, u_t) I(u_t \mid l_{0:t-k}, l_{t-k:t}) P^{\bar{\pi}}(l_{0:t-k} \mid l_{t-k:t}) P^{\bar{\pi}}(l_{t-k:t} \mid h_{t-k:t}) \\
&\quad \text{where } P^{\bar{\pi}}(l_{0:t-k} \mid l_{t-k:t}, h_{t-k:t}) = P^{\bar{\pi}}(l_{0:t-k} \mid l_{t-k:t}) \text{ because} \\
&\quad l_{t-k:t} \text{ is the only parent of } h_{t-k:t} \\
&= \sum_{l_{t-k:t}} \dot{T}(x_{t+1} \mid x_t, a_t, u_t) \sum_{l_{0:t-k}} P^{\bar{\pi}}(u_t, l_{0:t-k} \mid l_{t-k:t}) P^{\bar{\pi}}(l_{t-k:t} \mid h_{t-k:t}) \\
&= \sum_{l_{t-k:t}} \dot{T}(x_{t+1} \mid x_t, a_t, u_t) \bar{I}^{\bar{\pi}}(u_t \mid l_{t-k:t}) P^{\bar{\pi}}(l_{t-k:t} \mid h_{t-k:t})
\end{aligned}$$

Hence, putting all together,

$$\begin{aligned}
\bar{Q}^{\bar{\pi}}(h_{t-k:t}, a_t) = &\sum_{h_{0:t-k}} P^{\bar{\pi}}(h_{0:t-k} \mid h_{t-k:t}) \sum_{x_t} P(x_t \mid h_{0:t-k}, h_{t-k:t}) \dot{R}(x_t, a_t) \\
&+ \sum_{o_{t+1}} \sum_{x_{t+1}} \dot{O}(o_{t+1} \mid x_{t+1}) \sum_{u_t} \sum_{l_{t-k:t}} \dot{T}(x_{t+1} \mid x_t, a_t, u_t) \bar{I}^{\bar{\pi}}(u_t \mid l_{t-k:t}) \\
&P^{\bar{\pi}}(l_{t-k:t} \mid h_{t-k:t}) \sum_{a_{t+1}} \bar{\pi}(a_{t+1} \mid h_{-k+1:t+1}) \bar{Q}^{\bar{\pi}}(h_{-k+1:t+1}, a_{t+1})
\end{aligned}$$

Intuitively, we see that, because the action-value function $\bar{Q}^{\bar{\pi}}(a_t, h_{t-k:t})$ is just an expectation over all possible h_t given $h_{t-k:t}$, the distribution of u_t given the full ALSH l_t, $I(u_t \mid l_t)$, is effectively washed away by the same expectation. Hence, we may as well condition the influence predictor directly on $l_{t-k:t}$, $\bar{I}^{\bar{\pi}}(u_t \mid l_{t-k:t})$.

Finally, using the assumption $P(u_t \mid l_{t-k:t}, a_{0:t-k}) = P(u_t \mid l_{t-k:t})$ we can drop the superscript $\bar{\pi}$ from $\bar{I}^{\bar{\pi}}(u_t \mid l_{t-k:t})$. This assumption is important because the distribution $\bar{I}(u_t \mid l_{t-k:t})$ is generally not well-defined. Hence, if we estimate $\bar{I}^{\bar{\pi}_0}(u_t \mid l_{t-k:t})$ from samples collected while following $\bar{\pi}_0$ but $P(u_t \mid l_{t-k:t}, a_{0:t-k}) \neq P(u_t \mid l_{t-k:t})$, then, when we update the policy, the marginal distribution $P^{\bar{\pi}}(a_{0:t-k})$ will change and the old estimates will be biased.

□

Theorem 3.3.2. *A subset of variables D_t from L_t is guaranteed to elicit an invariant predictor of U_t, across all $\pi \in \Pi$, if (i) D_t constitutes a d-separating set (Definition 2.3.4) and (ii) all policies are functions of the local AOH, $\pi(h_t)$.*

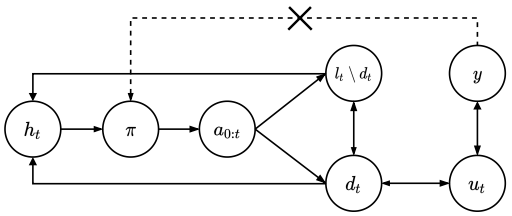


Figure A.1: Graphical causal model of a local-FPOMDP. The bidirectional arrows between two sets of variables (e.g. between d_t and u_t) indicate that there can be variables in one set that are affected by other variables in the other set and vice-versa. As depicted by the two diagonal arrows in the middle, actions $a_{0:t}$ may or may not be included in d_t. The top dashed arrow connecting y and π indicates that π cannot be a function of the non-local variables.

Proof. In this proof, we will make use of the graphical causal model in Figure A.1. We know that the agent's policy π can only influence the environment through its actions $a_{0:t}$. Moreover, from the definition of d-set (Definition 2.3.4), we know that the *direct path* Pearl (2000) that connects the past sequence of actions $a_{0:t}$ with u_t must go through d_t. Otherwise, $(u_t \not\perp\!\!\!\perp d_t \setminus l_t \mid d_t)$ and d_t would not constitute a d-set. Note that, as shown by the diagonal arrows in the middle, actions in $a_{0:t}$ may or may not be included in d_t.

Finally, the second condition in Theorem 3.3.2 implies that there cannot be *backdoor paths* Pearl (2000) between π and u_t. Hence, if we control for d_t, π and u_t become conditionally independent,

$$P(u_t \mid d_t) = P^{\pi}(u_t \mid d_t) \quad \text{for all} \quad \pi \in \Pi.$$

Moreover, for the above to hold, the d-set does not necessarily need to be minimal, $d_t \subseteq d_t^*$. That is, as long as the path between $l_t \setminus d_t$ and u_t remains closed, d_t may include as many additional (irrelevant) variables as there are in l_t. Note that this path is only open when d_t is not a d-set. □

Proposition 3.3.1. *Let π_0 be the exploratory policy used to collect the dataset. Then, for all $\pi \in \Pi$, $u_t \in \times_i \mathrm{dom}(U_t^i)$, $l_t \in \times_i \mathrm{dom}(L_t^i)$*

$$KL(P^{\pi_0}(d_t, u_t)||P^{\pi}(d_t, u_t)) \leq KL(P^{\pi_0}(l_t, u_t)||P^{\pi}(l_t, u_t))$$

where KL is the Kullback–Leibler divergence.

Proof.

$$\begin{aligned}
KL(P^{\pi_0}(d_t, u_t) \,||\, P^{\pi}(d_t, u_t)) &= \sum_{d_t, u_t} P^{\pi_0}(d_t, u_t) \log\left(\frac{P^{\pi_0}(d_t, u_t)}{P^{\pi}(d_t, u_t)}\right) \\
&= \sum_{d_t, u_t} P^{\pi_0}(u_t, d_t) \log\left(\frac{P^{\pi_0}(u_t \mid d_t) P^{\pi_0}(d_t)}{P^{\pi}(u_t \mid d_t) P^{\pi}(d_t)}\right) \\
&= \sum_{d_t} \log\left(\frac{P^{\pi_0}(d_t)}{P^{\pi}(d_t)}\right) \sum_{u_t} P^{\pi_0}(u_t, d_t) \\
&\quad \text{since, from Theorem 3.3.2, we know that } P(u_t \mid d_t) \\
&\quad \text{is invariant across all } \pi \in \Pi \\
&\quad = KL(P^{\pi_0}(d_t) \,||\, P^{\pi}(d_t))
\end{aligned}$$

Similarly, because $P(u_t \mid l_t)$ is also invariant across all $\pi \in \Pi$, $P^{\pi_0}(u_t \mid l_t) = P^{\pi}(u_t \mid l_t)$,

$$KL(P^{\pi_0}(l_t, u_t) \,||\, P^{\pi}(l_t, u_t)) = KL(P^{\pi_0}(l_t) \,||\, P^{\pi}(l_t)).$$

Then, since $d_t \subseteq l_t$, we can write

$$P^{\pi}(l_t) = P^{\pi}(d_t, l_t \setminus d_t)$$

and, using the chain rule

$$KL(P^{\pi_0}(l_t) \mid\mid P^{\pi}(l_t)) = KL(P^{\pi_0}(l_t \setminus d_t \mid d_t) \mid\mid P^{\pi}(l_t \setminus d_t \mid d_t)) + KL(P^{\pi_0}(d_t) \mid\mid P^{\pi}(d_t))$$
$$\geq KL(P^{\pi_0}(d_t) \mid\mid P^{\pi}(d_t))$$

where the last inequality holds because the KL divergence is always non-negative. □

Corollary 3.3.3. *Let $a_{0:t}$ be the past sequence of actions from 0 to t. If $P(d_t|do(a_{o:t})) = P(d_t)$ for all d_t and $a_{0:t}$. Then, for any $\pi_1 \neq \pi_2 : \pi_1, \pi_2 \in \Pi$ we have that, for all $u_t \in \times_i \text{dom}(U_t^i)$, $l_t \in \times_i \text{dom}(L_t^i)$, $P^{\pi_1}(d_t, u_t) = P^{\pi_2}(d_t, u_t)$ even though $P^{\pi_1}(l_t, u_t) \neq P^{\pi_2}(l_t, u_t)$.*

Proof. Follows from Proposition 3.3.1. We know that π can only influence the environment through the actions $a_{0:t}$. Hence, for all $\pi_1 \neq \pi_2, d_t, a_{0:t}$,

$$P(d_t \mid do(a_{0:t})) = P(d_t) \Rightarrow P^{\pi_1}(d_t) = P^{\pi_2}(d_t) \iff KL(P^{\pi_1}(d_t) \mid\mid P^{\pi_2}(d_t)) = 0$$

and from the proof of Proposition 3.3.1

$$KL(P^{\pi_1}(d_t, u_t) \mid\mid P^{\pi_2}(d_t, u_t)) = KL(P^{\pi_1}(d_t) \mid\mid P^{\pi_2}(d_t))$$

which means that, for all $\pi_1 \neq \pi_2, d_t, a_{0:t}, u_t$,

$$P(d_t \mid \text{do}(a_{0:t})) = P(d_t) \Rightarrow KL(P^{\pi_1}(d_t, u_t) \mid\mid P^{\pi_2}(d_t, u_t)) = 0 \iff P^{\pi_1}(d_t, u_t) = P^{\pi_2}(d_t, u_t)$$

and, because $l_t \subseteq a_{0:t}$ then, for all $\pi_1 \neq \pi_2, l_t, a_{0:t}, u_t$,

$$KL(P^{\pi_1}(l_t) \mid\mid P^{\pi_2}(l_t)) > 0 \iff KL(P^{\pi_1}(l_t, u_t) \mid\mid P^{\pi_2}(l_t, u_t)) > 0 \iff P^{\pi_1}(l_t, u_t) \neq P^{\pi_2}(l_t, u_t)$$

□

A.2 Example: Spurious Correlations in the Traffic Domain

Figure A.2 shows four screenshots taken from the SUMO simulator Lopez et al. (2018). These capture a sequence consecutive states in the traffic domain (see Section 3.4.2 for more details about this environment). The agent's local region is depicted by the red dashed box. The small cyan box on the right of every screenshot indicates the location of an influence source. As shown in the screenshots, when traffic is dense, lines of cars are

formed along the incoming lanes in front of the traffic lights. This situation leads to the appearance of spurious correlations between the traffic lights and the influence sources. In particular, Figure A.2 reveals that three timesteps after the traffic lights on the horizontal lanes are switched to green a new car appears at the influence source. Although there is clearly no direct relationship between these two events, if this pattern occurs often enough, as it is the case under a uniform random policy, the AIP might pick it up and use it as a shortcut. That is, the AIP might learn to predict that a car will show at the influence source exactly three timesteps after the lights are switched to green. This would indeed be effective at the beginning of training, since traffic jams are common under poor performing policies, but may result in highly inaccurate predictions when policies are further improved. As explained in Section 3.3.2 the problem above can be prevented if d-sets, rather than full AOHs, are fed into the AIP.

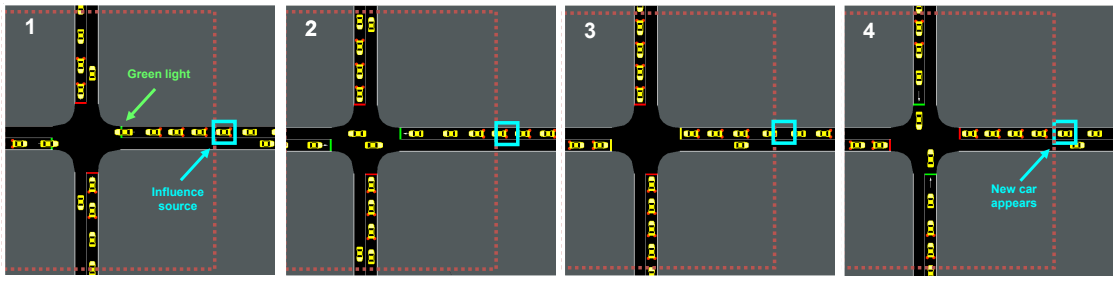


Figure A.2: Four screenshots capturing a sequence of states that may occur under a uniform random policy. The agent's local region is depicted by the red dashed box. The small cyan box on the right of every screenshot indicates the location of an influence source. The screenshots reveal that, when the traffic is dense, three timesteps after the traffic lights on the horizontal lanes are switched to green a new car appears at the influence source. This is clearly a spurious correlation as there is not direct relationship between the traffic lights and the influence sources.

A.3 Runtimes

The table below shows a breakdown of the runtime for the two environments and the three simulators. These were measured on an Intel i7-8650U CPU with 8 cores.

	Simulator	Agents training (h)	Data collection (h)	AIP training (h)	Total (h)
Traffic	GS	6.16	-	-	6.16
	IALS	2.13	0.03	0.01	2.17
	untrained-IALS	2.13	-	-	2.13
Warehouse	GS	13.12	-	-	13.12
	IALS	3.90	0.03	0.06	3.99
	untrained-IALS	3.90	-	-	3.90

A.4 Local Simulators

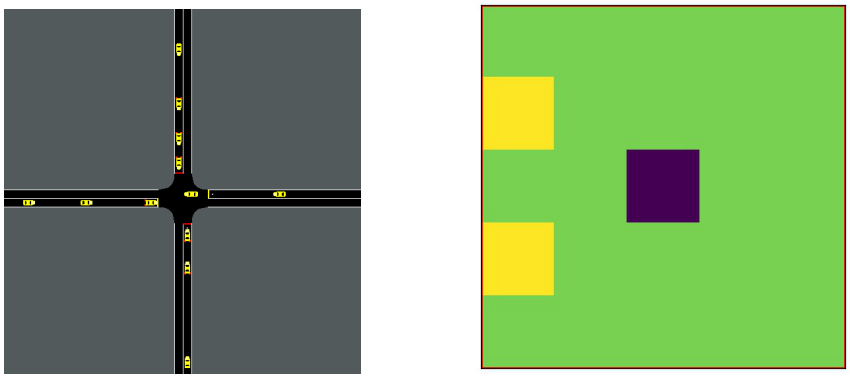


Figure A.3: A screenshot of the local simulators for the traffic control (left) and warehouse (right) environments

A.5 Results

A.5.1 Traffic control intersection 2

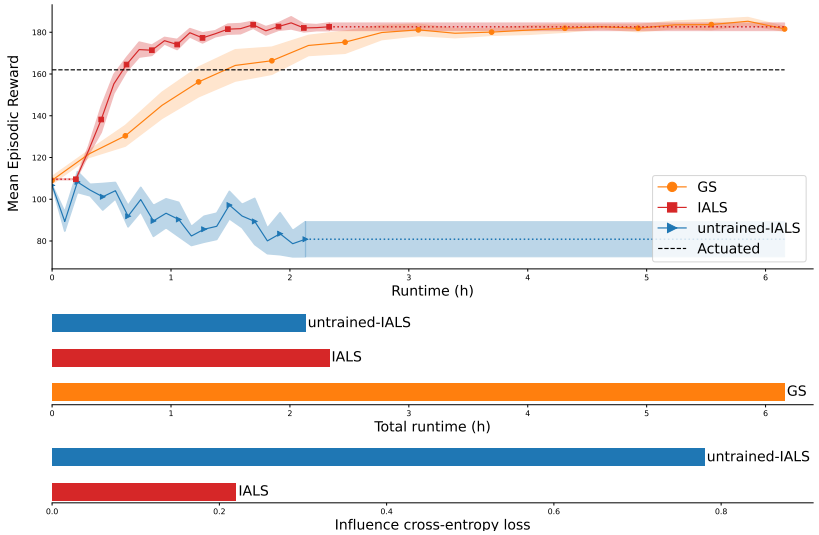


Figure A.4: **Top:** Learning curves of agents trained with the GS, the IALS, and the untrained-IALS on intersection 2 (Figure 3.2) as a function of wall-clock time. The dotted horizontal lines show the final performance of the agents after 2M timesteps of training. The dotted horizontal line at the begining of the red curve corresponds to the AIP training time. **Middle:** Total runtime of training for 2M training steps on the three simulators. **Bottom:** Cross entropy loss for the trained and untrained AIPs.

A.5.2 Comparison on the number of timesteps

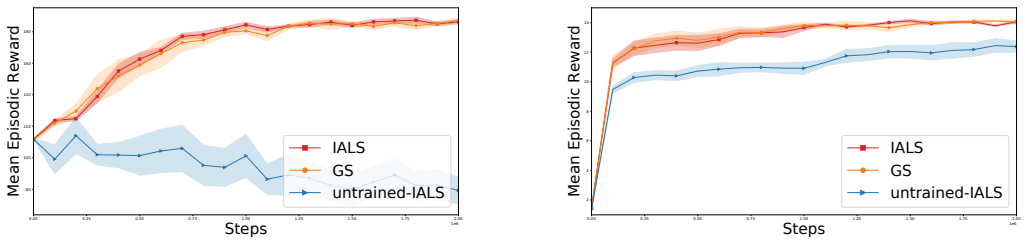


Figure A.5: Learning curves as a function of the number of timesteps for the traffic (left) and warehouse (right) environments.

A.5.3 Sufficiently similar training conditions

Here we further explore our research question in Section 3.3.3. To what extent can agents trained on inaccurate IALS achieve similar performance to those trained on the GS? To shed some light on this this complicated question, we introduce a new type of AIP which represents the influence sources by a fixed marginal probability $P(u_t)$ independent of the ALSH. We call the resulting simulator F-IALS (F for fixed).

Traffic results: Figure 3.3 shows the performance of agents trained on the F-IALS with $P(u_t) = 0.1$ (F-IALS 0.1) and $P(u_t) = 0.5$ (F-IALS 0.5). The bar plot at the bottom of Figure 3.3 shows that the cross-entropy loss is much higher when modeling the influence sources with the marginal distribution $P(u_t)$ (F-IALS 0.1 and F-IALS 0.5) than when modeling them with the learned influence predictor $P(u_t \mid l_t)$ (IALS). This suggests that there is a non-trivial relationship between ALSH and influence sources, and in principle

$$\begin{aligned} KL(I(u_t \mid d_t) \mid\mid \hat{I}_\theta(u_t \mid d_t)) &< KL(I(u_t \mid d_t) \mid\mid P(u_t) = 0.1) \\ &< KL(I(u_t \mid d_t) \mid\mid P(u_t) = 0.5) \end{aligned} \tag{A.4}$$

Nonetheless, agents trained on the F-IALS 0.1 reach the same (intersection 2) or similar (intersection 1) level of performance as those trained on the IALS. This is in line with our hypothesis in section 3.3.3. If real trajectories are somewhat likely under a random influence predictor, an agent with sufficient capacity will be able to learn from them and perform well on the true environment. In fact, given that the probability used for the inflow of vehicles entering the GS is also 0.1, the chances of generating traffic patterns with the F-IALS (0.1) similar to those of the GS are very high. In contrast, when setting the probability of cars entering the LS to a less sensible $P(u_t) = 0.5$ agents trained on the F-IALS (0.5) can no longer match the same performance level.

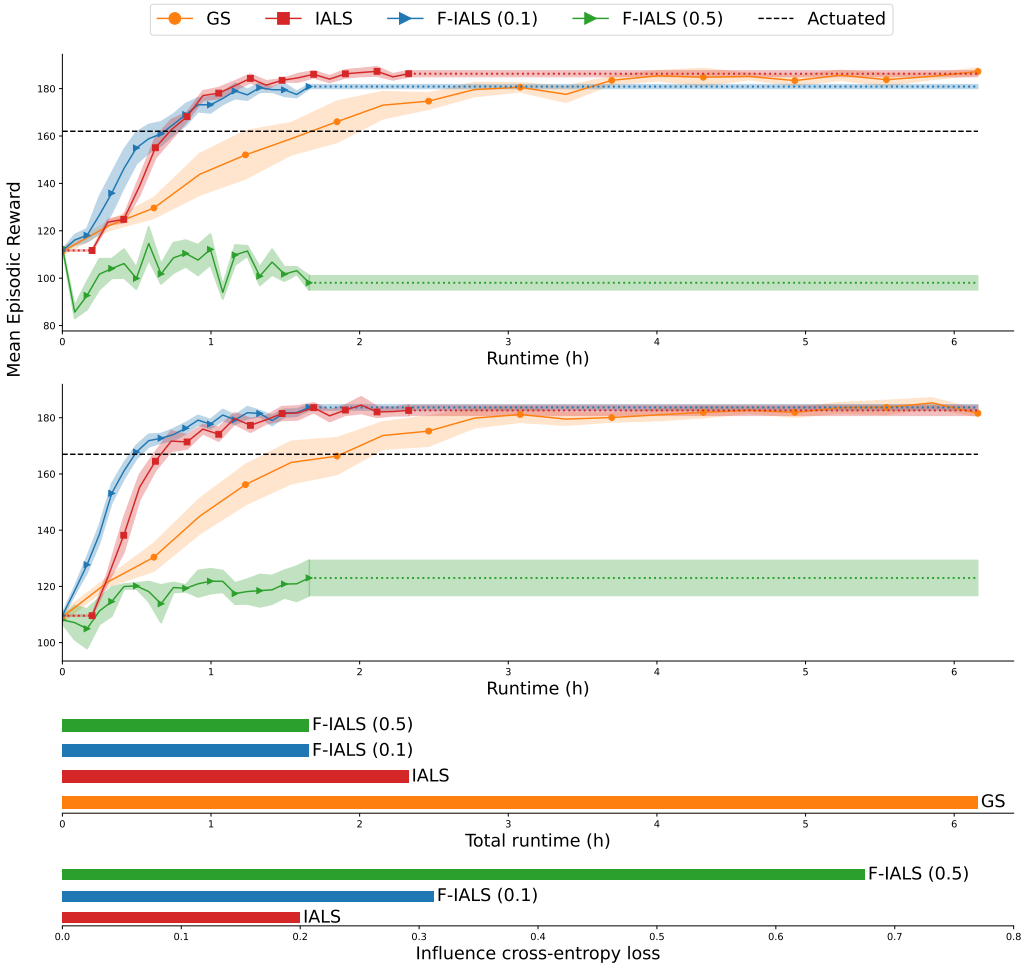


Figure A.6: **Top:** Learning curves of agents trained with the GS, the IALS and the F-IALS on the the two intersections highlighted in Figure 3.2 as a function of wall-clock time. The dotted horizontal lines show the final performance of the agents after 2M timesteps of training. **Second from the bottom:** Total runtime of training for 2M training steps on the three simulators. **Bottom:** Cross entropy loss evaluated at intersection 1 for the three AIPs (values are very similar for intersection 2).

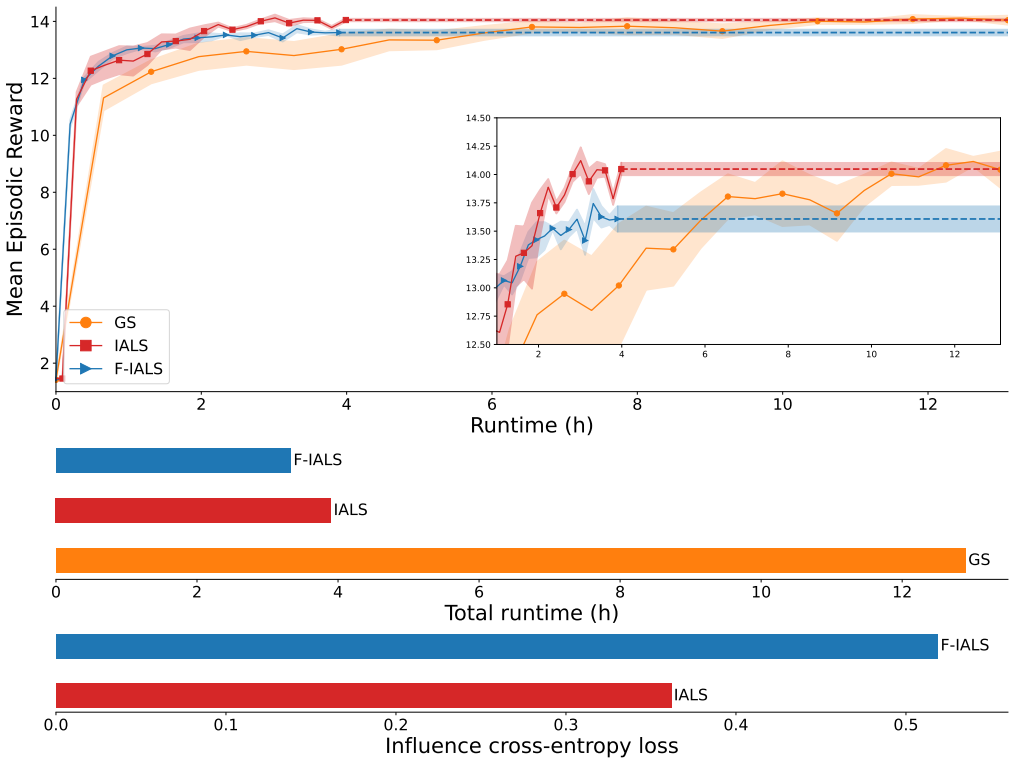


Figure A.7: **Top:** Learning curves of agents trained with the GS, the IALS and the F-IALS on the the warehouse commissioning task as a function of wall-clock time. Zoom in version of the same chart showing the performance difference between IALS and F-IALS. The dotted horizontal lines show the final performance of the agents after 2M timesteps of training. The dotted horizontal line at the beginning of the red curve corresponds to the AIP training time. **Second from the bottom:** Total runtime of training for 2M steps on the three simulators. **Bottom:** Cross entropy loss for the two AIPs.

Warehouse results: In this environment the fixed influence source probabilities is set to an estimate $\hat{P}(u_t)$ of true value $P^{\pi_0}(u_t)$, which we approximated empirically from $N = 10K$ samples collected from the GS while following π_0. Even so, the cross-entropy of the F-IALS is again higher than that of the learned influence predictor $\hat{I}_\theta(u_t \mid l_t)$, indicating again that influence sources u_t and ALSHs are strongly coupled with one another and that there is a non-trivial relationship between them,

$$KL(I(u_t \mid d_t) \mid\mid \hat{I}_\theta(u_t \mid d_t)) < KL(I(u_t \mid d_t) \mid\mid \hat{P}(u_t)) \tag{A.5}$$

Despite being less accurate, agents trained on the F-IALS can also perform well on the true environment. Yet, they do not reach the same level of performance as those obtained with the GS and the IALS (see zoom in version in Figure 3.5). Even though, the basic strategy on how to collect items can be learned from the F-IALS, the simulator does not provide the extra level of detail needed to learn better policies. That is, a consistent pattern that is present in both the GS and the IALS by which items that have been active the longest are more likely to disappear next. With this, agents can learn to not go for an item when the chances that a neighbor robot will get there first are high.

All in all we see that, in certain situations, inaccurate influence predictors can still provide a fair amount of useful experiences for the agent to perform well on the true environment. However, in domains with complicated dynamics, such as our warehouse environment, the best policies can only be obtained when simulations provide the extra level of detail that only accurate influence predictors are able to deliver.

A.6 Implementation Details

Approximate influence predictor: Due to the sequential nature of the problem, rather than feeding the full past history every time we make a prediction, we use a recurrent neural network (RNN) (Hochreiter and Schmidhuber, 1997; Cho et al., 2014) and process observations one at a time,

$$P(u_t \mid l_t) \approx \hat{I}_\theta(u_t \mid \hat{h}_{t-1}, o_t) = F_{\text{rnn}}(\hat{h}_{t-1}, o_t, u_t), \tag{A.6}$$

where we use $\hat{h}$ to indicate that the history h is embedded in the RNN's internal memory.

Given that we generally have multiple influence sources $u_t = \langle u_t^1 \dots u_t^M \rangle$, we need to fit M separate models $\hat{I}_{\theta_m}$ to predict each of the M influence sources. In practice, to reduce the computational cost, we can have a single network with a common representation module for all influence sources and output their probability distributions using M separate heads. This representation assumes that the influence sources are independent of one another,

$$I(u_t \mid l_t) = \prod_{m=0}^{M} P(u_t^m \mid l_t), \tag{A.7}$$

which is true for the two domains we study in this paper.

Practical implications of Theorem 3.3.1: One important consideration when training RNNs via backpropagation through time (BPTT) is to choose the right length for the

input sequences (Williams and Peng, 1990). This determines the number of steps the network is backpropagated and thus for how long past information can be retained. On the one hand, longer sequences will often provide more information to predict the influence sources; on the other, they will also make it harder to optimize the network. Ideally, we would like to choose just the right sequence length such that the agent cannot perceive any distribution shift in the local transitions. Assuming that in our environment $P(u_t \mid l_{t-k:t}, a_{0:t-k}) = P(u_t \mid l_{t-k:t})$, from Theorem 3.3.1 we know that the sequence length should be (at least) as long as that of the agent's (if these are also modeled by RNNs or as long as the number of stacked observations fed to feedforward neural networks; FNN). If $P(u_t \mid l_{t-k:t}, a_{0:t-k}) \neq P(u_t \mid l_{t-k:t})$ we can still condition the AIP only on $l_{t-k:t}$, but we might need to retrain the AIP every certain number of policy updates to prevent it from becoming stale (see last paragraph in the proof of theorem 3.3.1; Appendix A.1).

Policies: We model policies by FNNs. In the warehouse environment, since the agent needs memory to perform well, policies are fed with a stack of the last 8 observations. This architecture performed better than GRUs. In the traffic control task, on the other hand, policies are fed only with the current observation as this seems to be sufficient to predict the influence sources.

A.7 Algorithms

Algorithm 2 Collect dataset with GS

Input: T, O, π_0 ▷ Global simulator, global observation function, and exploratory policy
for $n \in \langle 0, ..., N/T \rangle$ **do**
  $s_0 \leftarrow$ reset ▷ Reset initial state
  $x_0 \leftarrow s_0$ ▷ Extract local state from global state
  $l_0 \leftarrow x_0$ ▷ Initialize ALSH with initial local state
  $o_0 \sim O(\cdot \mid s_0)$ ▷ Sample observation from O
  $h_0 \leftarrow o_0$ ▷ Initialize AOH with initial observation
  for $t \in \langle 0, ..., T \rangle$ **do**
    $\langle u_t^0, ..., u_t^k \rangle \leftarrow s_t$ ▷ Extract influence sources from global state
    $D \leftarrow \{l_t, \langle u_t^0, ..., u_t^k \rangle\}$ ▷ Append ALSH-influence-source pair to the dataset
    $a_t \sim \pi(\cdot \mid h_t)$ ▷ Sample action
    $s_{t+1} \sim T(\cdot \mid s_t, a_t)$ ▷ Sample next state from GS
    $x_{t+1} \leftarrow s_{t+1}$ ▷ Extract local state from global state
    $l_{t+1} \leftarrow \langle a_t, x_{t+1} \rangle$ ▷ Append action-local-state pair to ALSH
    $o_{t+1} \sim O(\cdot \mid s_{t+1})$ ▷ Sample observation from O
    $h_{t+1} \leftarrow \langle a_t, o_{t+1} \rangle$ ▷ Append actions-observation pair to the AOH
  end for
end for

Algorithm 3 Collect dataset with GS

Input: T, O, π_0 ▷ Global simulator, global observation function, and exploratory policy
for $n \in \langle 0, ..., N/T \rangle$ **do**
  $s_0 \leftarrow$ reset ▷ Reset initial state
  $x_0 \leftarrow s_0$ ▷ Extract local state from global state
  $l_0 \leftarrow x_0$ ▷ Initialize ALSH with initial local state
  $o_0 \sim O(\cdot \mid s_0)$ ▷ Sample observation from O
  $h_0 \leftarrow o_0$ ▷ Initialize AOH with initial observation
  for $t \in \langle 0, ..., T \rangle$ **do**
    $\langle u_t^0, ..., u_t^k \rangle \leftarrow s_t$ ▷ Extract influence sources from global state
    $D \leftarrow \{l_t, \langle u_t^0, ..., u_t^k \rangle\}$ ▷ Append ALSH-influence-source pair to the dataset
    $a_t \sim \pi(\cdot \mid h_t)$ ▷ Sample action
    $s_{t+1} \sim T(\cdot \mid s_t, a_t)$ ▷ Sample next state from GS
    $x_{t+1} \leftarrow s_{t+1}$ ▷ Extract local state from global state
    $l_{t+1} \leftarrow \langle a_t, x_{t+1} \rangle$ ▷ Append action-local-state pair to ALSH
    $o_{t+1} \sim O(\cdot \mid s_{t+1})$ ▷ Sample observation from O
    $h_{t+1} \leftarrow \langle a_t, o_{t+1} \rangle$ ▷ Append actions-observation pair to the AOH
  end for
end for

Appendix B

Distributed Influence-Augmented Local Simulators

B.1 Proofs

Lemma 4.4.1. *Let $\Pi = \times_{i \in N} \Pi^i$ be the product space of joint policies with Π^i being the set of policies for agent i. Moreover, let $\Psi = \times_{i \in N} \Psi^i$ be the product space of joint influences, with Ψ^i being the set of influence distributions for agent i. Every joint policy $\pi \in \Pi$ induces exactly one influence distribution $I^i \in \Psi^i$ for every agent $i \in N$.*

Proof. We will prove it by contradiction. Let us assume there is a single joint policy π that induces two different influence distributions I_1^i and I_2^i on agent i. From the definition of influence (Section 4.1; Oliehoek et al. 2021) we have

$$I_1^i(u_t^i|l_t^i) = \sum_{u_{t-1}^i, y_{t-1}^i, a_{t-1}^{-i}} P_1(u_t^i|x_{t-1}^i, u_{t-1}^i, y_{t-1}^i, a_{t-1}) P_1(u_{t-1}^i, y_{t-1}^i, a_{t-1}^{-i}|l_t^i) \tag{B.1}$$

and

$$I_2^i(u_t^i|l_t^i) = \sum_{u_{t-1}^i, y_{t-1}^i, a_{t-1}^{-i}} P_2(u_t^i|x_{t-1}^i, u_{t-1}^i, y_{t-1}^i, a_{t-1}) P_2(u_{t-1}^i, y_{t-1}^i, a_{t-1}^{-i}|l_t^i). \tag{B.2}$$

First, we see that, because $\langle x_{t-1}^i, u_{t-1}^i, y_{t-1}^i \rangle$ fully determines the Markov state s_{t-1}, the first term in the summation can be computed from the environment's transition function,

and thus

$$\begin{aligned} P_1(u_t^i|x_{t-1}^i, u_{t-1}^i, y_{t-1}^i, a_{t-1}) =& P_2(u_t^i|x_{t-1}^i, u_{t-1}^i, y_{t-1}^i, a_{t-1}) \\ =& \sum_{s_t} \mathbb{1}(u_t, s_t) T(s_t|s_{t-1}, a_{t-1}), \end{aligned} \tag{B.3}$$

where $\mathbb{1}(u_t, s_t)$ is an indicator function that determines if the state s_t is feasible in the context of u_t.

Further, we know that

$$P_1(u_{t-1}^i, y_{t-1}^i, a_{t-1}^{-i}|l_t^i) = \sum_{h_{t-1}^{-i}} \pi^{-i}(a_{t-1}^{-i}|h_{t-1}^{-i}) P_1(u_{t-1}^i y_{t-1}^i, h_{t-1}^{-i}|l_t^i) \tag{B.4}$$

$$P_2(u_{t-1}^i, y_{t-1}^i, a_{t-1}^{-i}|l_t) = \sum_{h_{t-1}^{-i}} \pi^{-i}(a_{t-1}^{-i}|h_{t-1}^{-i}) P_2(u_{t-1}^i y_{t-1}^i, h_{t-1}^{-i}|l_t^i) \tag{B.5}$$

where $P_1(u_{t-1}^i y_{t-1}^i, h_{t-1}^{-i}|l_t)$ and $P_2(u_{t-1}^i y_{t-1}^i, h_{t-1}^{-i}|l_t)$ can be computed recursively as

$$\begin{aligned} &P_1(u_{t-1}^i, y_{t-1}^i, h_{t-1}^{-i}|l_t^i) = \\ &\qquad \sum_{h_{t-2}^{-i}, o_{t-1}^{-i}} O(o_{t-1}^{-i}|x_{t-1}^i, u_{t-1}^i, y_{t-1}^i) \pi^{-i}(a_{t-2}^{-i}|h_{t-2}^{-i}) P_1(u_{t-1}^i, y_{t-1}^i, h_{t-2}^{-i}|l_t^i), \end{aligned} \tag{B.6}$$

and

$$\begin{aligned} &P_2(u_{t-1}^i, y_{t-1}^i, h_{t-1}^{-i}|l_t^i) = \\ &\qquad \sum_{h_{t-2}^{-i}, o_{t-1}^{-i}} O(o_{t-1}^{-i}|x_{t-1}^i, u_{t-1}^i, y_{t-1}^i) \pi^{-i}(a_{t-2}^{-i}|h_{t-2}^{-i}) P_2(u_{t-1}^i, y_{t-1}^i, h_{t-2}^{-i}|l_t^i), \end{aligned} \tag{B.7}$$

with $h_{t-1}^{-i} = \langle h_{-i,t-2}, a_{-i,t-2}, o_{t-1}^{-i} \rangle$. Then, if we further unroll equations (B.6) and (B.7) up to timestep 0, we see that all probability distributions in both cases are equivalent and we reach a contradiction. Hence,

$$I_1^i(u_t^i|l_t^i) = I_2^i(u_t^i|l_t^i) \tag{B.8}$$

□

Proposition 4.4.1. *The space of joint policies $\Pi = \times_{i \in N} \Pi^i$ is necessarily greater than or equal to the space of joint influences $\Psi = \times_{i \in N} \Psi^i$, $|\Pi| \geq |\Psi|$. Moreover, there exist local-form FPOSGs for which the inequality is strict.*

Proof. From Proposition 4.4.1 it follows that the space of joint influences Ψ is at most as large as the space of joint policies Π, $|\Psi| \not> |\Pi|$. Hence, we just need to show that in some cases Π is strictly greater than Ψ, $|\Pi| > |\Psi|$. □

A clear example is that where each agent's local region X^i is independent of the other agents' policies π^{-i} (Becker et al., 2003). That is, the actions of other agents a^{-i} have no effect on agent i's local state transitions. From the definition of IALM (Definition 4.3.3) we know that, in our setting, a^{-i} can only affect the local state transitions through u^i. Therefore, for the local transitions to be independent the following should hold

$$P(u_t^i|x_{t-1}^i, u_{t-1}^i, y_{t-1}^i, a_{t-1}) = P(u_t^i|x_{t-1}^i, u_{t-1}^i, y_{t-1}^i, a_{t-1}^i) \tag{B.9}$$

The equation above reflects that only agent i can affect u_t^i. Thus, in the event of local transition independence, we have that

$$\forall i \in N : \exists! I^{i*}(u_t^i|l_t^i) \in \Psi^i : \forall \pi \in \Pi \left(I^i(u_t^i|l_t^i, \pi) = Ii*(u_t^i|l_t^i)\right) \tag{B.10}$$

That is, for any joint policy $\pi \in \Pi$ there is a unique influence distribution $I^{i*} \in \Psi^i$ for every agent $i \in N$, and thus, in this particular case, $|\Pi| \gg |\Psi| = 1$.

To prove Lemma 4.4.3 we will use the following lemma.

Lemma B.1.1. *Let $I_1^i(u_t^i|l_t^i)$ and $I_2^i(u_t^i|l_t^i)$ be two different influence distributions with M_1^i and M_2^i being the IALMs induced by each of them respectively. Moreover, let $P_1(h_{t+1}^i|h_t^i, a_t^i)$ and $P_2(h_{t+1}^i|h_t^i, a_t^i)$ denote the resulting local AOH transitions for M_1^i and M_2^i respectively. The following inequality holds*

$$\sum_{x_{t+1}^i} \left|P_1(h_{t+1}^i|h_t^i, a_t^i) - P_2(h_{t+1}^i|h_t^i, a_t^i)\right| \leq \sum_{l_t^i, u_t^i} P(l_t^i|h_t^i) \left|I_1(u_t^i|l_t^i) - I_2(u_t^i|l_t^i)\right| \quad \forall h_t^i, a_t^i \tag{B.11}$$

Proof.

$$
\begin{aligned}
&\sum_{h^i_{t+1}} \left|P_1(h^i_{t+1}|h^i_t, a^i_t) - P_2(h^i_{t+1}|h^i_t, a^i_t)\right| \\
=&\sum_{o^i_{t+1}} \Big| \sum_{x^i_{t+1}} O^i(o^i_{t+1}|x^i_{t+1}) \sum_{u^i_t} \dot{T}^i(x^i_{t+1}|x^i_t, u^i_t, a^i_t) \sum_{l^i_t} I_1(u^i_t|l^i_t) P(l^i_t|h^i_t) \\
&- \sum_{x^i_{t+1}} O^i(o^i_{t+1}|x^i_{t+1}) \sum_{u^i_t} \dot{T}^i(x^i_{t+1}|x^i_t, u^i_t, a^i_t) \sum_{l^i_t} I_2(u^i_t|l^i_t) P(l^i_t|h^i_t) \Big| \\
=&\sum_{o^i_{t+1}} \Big| \sum_{x^i_{t+1}} O^i(o^i_{t+1}|x^i_{t+1}) \sum_{u^i_t} \dot{T}^i(x^i_{t+1}|x^i_t, u^i_t, a^i_t) \sum_{l^i_t} P(l^i_t|h^i_t) \big[I_1(u^i_t|l^i_t) - I_2(u^i_t|l^i_t)\big] \Big| \\
=&\Big| \sum_{l^i_t, u^i_t} P(l^i_t|h^i_t) \big[I_1(u^i_t|l^i_t) - I_2(u^i_t|l^i_t)\big] \Big| \\
\leq&\sum_{l^i_t, u^i_t} P(l^i_t|h^i_t) \left|I_1(u^i_t|l^i_t) - I_2(u^i_t|l^i_t)\right|
\end{aligned}
\tag{B.12}
$$

□

Lemma 4.4.3. *Let M^i_1 and M^i_2 be two IALMS differing only on their influence distributions $I^i_1(u^i_t|l^i_t)$ and $I^i_2(u^i_t|l^i_t)$. Let $Q^{\pi^i}_{M^i_1}$ and $Q^{\pi^i}_{M^i_2}$ be the value functions induced by M^i_1 and M^i_2 for the same π^i. If, for all $h^i_t \in H^i_t$, $u^i_t \in \times_j \mathrm{dom}(U^{i,j}_t)$, I^i_1 and I^i_2 satisfy*

$$\sum_{l^i_t, u^i_t} P(l^i_t \mid h^i_t) \left|I^i_1(u^i_t \mid l^i_t) - I^i_2(u^i_t|l^i_t)\right| \leq \xi, \tag{4.2}$$

then

$$\left|Q^{\pi^i}_{M^i_1}(h^i_t, a^i_t) - Q^{\pi^i}_{M^i_2}(h^i_t, a^i_t)\right| \leq \bar{R}\frac{(K-t)(K-t+1)}{2}\xi \tag{4.3}$$

for all $\pi^i \in \Pi^i$, $h^i_t \in H^i_t$, and $a^i_t \in \mathcal{A}^i$, where K is the horizon and $\bar{R} = ||R||_\infty$

Proof. This is a special case of the simulation lemma (Kearns and Singh, 2002). We have that the set of local states and actions is the same for both IALMs. Moreover, the reward function is also the same $R_1(x_t, a_t) = R_2(x_t, a_t)$.

$$\begin{aligned}\left|Q^{\pi^i}_{M^i_1}(h^i_t,a^i_t)-Q^{\pi^i}_{M^i_2}(h^i_t,a^i_t)\right| =& \Bigg|\sum_{x^i_t} P(x^i_t|h^i_t)R(x^i_t,a^i_t)\\ &+\sum_{h^i_{t+1},a^i_{t+1}} P_1(h^i_{t+1}|h^i_t,a^i_t)\pi^i(a^i_{t+1}|h^i_{t+1})Q^{\pi^i}_{M^i_1}(h^i_{t+1},a^i_{t+1})-\sum_{x^i_t} P(x^i_t|h^i_t)R(x^i_t,a^i_t)\\ &-\sum_{h^i_{t+1},a^i_{t+1}} P_2(h^i_{t+1}|h^i_t,a^i_t)\pi^i(a^i_{t+1}|h^i_{t+1})Q^{\pi^i}_{M^i_2}(h^i_{t+1},a^i_{t+1})\Bigg|,\end{aligned} \tag{B.13}$$

where $P_1(h^i_{t+1}|h^i_t,a^i_t)$ and $P_2(h^i_{t+1}|h^i_t,a^i_t)$ are the AOH transitions induced by I_1 and I_2 respectively.

$$\begin{aligned}\left|Q^{\pi^i}_{M^i_1}(h^i_t,a^i_t)-Q^{\pi^i}_{M^i_2}(h^i_t,a^i_t)\right| =& \Bigg|\sum_{h^i_{t+1},a^i_{t+1}} \pi^i(a^i_{t+1}|h^i_{t+1})\Big[\\ &P_1(h^i_{t+1}|h^i_t,a^i_t)Q^{\pi^i}_{M^i_1}(h^i_{t+1},a^i_{t+1})-P_2(h^i_{t+1}|h^i_t,a^i_t)Q^{\pi^i}_{M^i_2}(h^i_{t+1},a^i_{t+1})\Big]\Bigg|\\ =& \Bigg|\sum_{h^i_{t+1},a^i_{t+1}} \pi^i(a^i_{t+1}|h^i_{t+1})\Big[\\ &P_1(h^i_{t+1}|h^i_t,a^i_t)Q^{\pi^i}_{M^i_1}(h^i_{t+1},a^i_{t+1})-P_2(h^i_{t+1}|h^i_t,a^i_t)Q^{\pi^i}_{M^i_1}(h^i_{t+1},a^i_{t+1})\\ &+P_2(h^i_{t+1}|h^i_t,a^i_t)Q^{\pi^i}_{M^i_1}(h^i_{t+1},a^i_{t+1})-P_2(h^i_{t+1}|h^i_t,a^i_t)Q^{\pi^i}_{M^i_2}(h^i_{t+1},a^i_{t+1})\Big]\Bigg|\\ \leq& \Bigg|\bar{R}(K-t)\sum_{h^i_{t+1}}\big(P_1(h^i_{t+1}|h^i_t,a^i_t)-P_2(h^i_{t+1}|h^i_t,a^i_t)\big)\\ &+\sum_{h^i_{t+1},a^i_{t+1}} \pi^i(a^i_{t+1}|h^i_{t+1})P_2(h^i_{t+1}|h^i_t,a^i_t)\Big[Q^{\pi^i}_{M^i_1}(h^i_{t+1},a^i_{t+1})-Q^{\pi^i}_{M^i_2}(h^i_{t+1},a^i_{t+1})\Big]\Bigg|\end{aligned} \tag{B.14}$$

since $Q^{\pi^i}_{M^i_1}(h^i_{t+1})\leq \bar{R}(K-t)$. Then, from Lemma B.1.1 we know that

$$\sum_{h^i_{t+1}}\big(P_1(h^i_{t+1}|h^i_t,a^i_t)-P_2(h^i_{t+1}|h^i_t,a^i_t)\big)\leq \sum_{l^i_t,u^i_t} P(l^i_t|h^i_t)\left|I_1(u^i_t|l^i_t)-I_2(u^i_t|l^i_t)\right|\leq \xi \quad \forall h^i_t,a^i_t. \tag{B.15}$$

Hence,

$$\left|Q^{\pi^i}_{M_1^i}(h_t^i, a_t^i) - Q^{\pi^i}_{M_2^i}(h_t^i, a_t^i)\right| \leq \sum_{k=t}^{K} \bar{R}(K-k)\xi = \bar{R}\frac{(K-t)(K-t+1)}{2}\xi. \tag{B.16}$$

□

Theorem 4.4.4. *Let M_1^i and M_2^i be two IALMS differing only on their influence distributions $I_1^i(u_t^i|l_t^i)$ and $I_2^i(u_t^i|l_t^i)$. M_1^i and M_2^i induce the same optimal policy π^{i*} if, for some Δ,*

$$Q^{\pi^{i*}}_{M_1^i}(h_t^i, \bar{a}_t^i) - Q^{\pi^{i*}}_{M_1^i}(h_t^i, \hat{a}_t^i) > 2\Delta \quad \forall h_t^i, \hat{a}_t^i \neq \bar{a}_t^i \tag{4.4}$$

with

$$\left|Q^{\pi^i}_{M_1^i}(h_t^i, a_t^i) - Q^{\pi^i}_{M_2^i}(h_t^i, a_t^i)\right| \leq \Delta \quad \forall h_t^i, a_t^i, \pi^i, \tag{4.5}$$

where $\bar{a}_t^i = \arg\max_{a_t^i} Q^{\pi^{i*}}_{M_1^i}(h_t^i, a_t^i)$

Proof. We will prove it by contradiction. Let us assume there is a policy π^* that is optimal for M_1^i but not for M_2^i. This implies that, for some h_t^i, there is at least one action $\hat{a}_t^i \neq \bar{a}_t^i$ for which

$$Q^{\pi^*}_{M_2^i}(h_t^i, \bar{a}_t^i) < Q^{\pi^*}_{M_2^i}(h_t^i, \hat{a}_t^i) \tag{B.17}$$

Then, because the maximum gap between $Q_{M_1^i}$ and $Q_{M_2^i}$ is Δ,

$$Q^{\pi^*}_{M_1^i}(h_t^i, \bar{a}_t^i) - \Delta \leq Q^{\pi^*}_{M_2^i}(h_t^i, \bar{a}_t^i) < Q^{\pi^*}_{M_2^i}(h_t^i, \hat{a}_t^i) \leq Q^{\pi^*}_{M_1^i}(h_t^i, \hat{a}_t^i) + \Delta. \tag{B.18}$$

Therefore, we have

$$Q^{\pi^*}_{M_1^i}(h_t^i, \bar{a}_t^i) - Q^{\pi^*}_{M_1^i}(h_t^i, \hat{a}_t^i) < 2\Delta, \tag{B.19}$$

which contradicts the statement

$$Q^{\pi^*}_{M_1^i}(h_t^i, \bar{a}_t^i) - Q^{\pi^*}_{M_1^i}(h_t^i, a_t^i) > 2\Delta \quad \forall h_t^i, a_t^i \tag{B.20}$$

□

B.2 Further Related Work

There is a sizeable body of literature that concentrates on the non-stationarity issues arising from having multiple agents learning simultaneously in the same environment

(Laurent et al., 2011; Hernandez-Leal et al., 2017). Although oftentimes the problem can be simply ignored with virtually no consequences for the agents' performance (Tan, 1993), in general, disregarding changes in the other agents' policies, and assuming individual Q-values to be stationary, can have a catastrophic effect on convergence (Claus and Boutilier, 1998).

The problem of non-stationarity becomes even more severe in the Dec-POMDP setting (Oliehoek and Amato, 2016) since policy changes may not be immediately evident from each agent's AOH. To compensate for this Raileanu et al. (2018) and Rabinowitz et al. (2018) explicitly train models that predict the other agents' goals and behaviors. In contrast, Foerster et al. (2018a) add an extra term to the learning objective that is meant to predict the other agents' parameter updates. This approach is empirically shown to encourage cooperation in general-sum games. In order to better approximate the value function, several works have studied the use of additional information during training to inform each individual agent of changes in the other agents' policies, leading to the ubiquitous centralized training decentralized execution (CTDE) paradigm. The works by Lowe et al. (2017) and Foerster et al. (2018b) exploit this by training a single centralized critic that takes as input the true state and joint action of all the agents. This critic is then used to update the policies of all agents following the actor-critic policy gradient update (Konda and Tsitsiklis, 1999). Even though the use of additional information to augment the critic may help reduce bias in the value estimates, the idea lacks any theoretical guarantees and has been shown to produce the same policy gradient in expectation as those produced by multiple independent critics (Lyu et al., 2021). Moreover, according to Lyu et al., naively augmenting the critic with all other agents' actions and observations can heavily increase the variance of the policy gradients. Both results, however, assume that the critics have converged to the true on-policy value estimates. The authors do admit that, in practice, critics are often used even when they have not yet converged. In such situations, centralized critics might provide more stable policy updates since they are better equipped to follow the true non-stationary Q values. Following a similar perspective, the concurrent work by Spooner et al. (2021) tries to reduce variance by using a per-agent baseline function that removes from the policy gradient the contributions to the joint value estimates of those agents that are conditionally independent, thus effectively providing the agent with more stable updates. The works by de Witt et al. (2020) and Yu et al. (2021) show that the vanilla PPO algorithm (Schulman et al., 2017) works already quite well on several multi-agent tasks. Yu et al. attribute the positive empirical results to the clipping parameter ϵ, which prevents individual policies from changing drastically, and in

turn, reduces the problem of non-stationarity. Li et al. (2021b) further analyze this idea and propose a method to estimate the joint policy divergence, which is then used as a constraint in the optimization objective.

B.3 Algorithms

The two algorithms below describe how to generate the datasets $\{D^i\}_{i \in N}$ with the GS (Algorithm 4) and how to simulate trajectories with each of the IALS (Algorithm 5).

Algorithm 4 Collect datasets $\{D^i\}_{i \in N}$ with GS

Input: T, $\{\dot{O}^i\}_{i \in N}$, $\pi_0 = \{\pi_0^i\}_{i \in N}$ ▷ Global simulator, observation functions, and joint policy
for $n \in \langle 0, ..., N/T \rangle$ **do**
  $s_0 \leftarrow$ reset ▷ Reset initial state
  $\{x_0^i\}_{i \in N} \leftarrow s_0$ ▷ Extract local states from global state
  $\{l_0^i \leftarrow x_0^i\}_{i \in N}$ ▷ Initialize each agent's ALSH with initial local state
  $\{o_0^i \sim O^i(\cdot \mid x_0)\}_{i \in N}$ ▷ Sample each agent's observation from O^i
  $\{h_0^i \leftarrow o_0^i\}_{i \in N}$ ▷ Initialize each agent's AOH with initial observation
  for $t \in \langle 0, ..., T \rangle$ **do**
    $\{u_0^i\}_{i \in N} \leftarrow s_0$ ▷ Extract each agent's influence sources from global state
    $\{D^i \leftarrow (l_t^i, u_t^i)\}_{i \in N}$ ▷ Append ALSH-influence-source pair to the datasets
    $\{a_t^i \sim \pi(\cdot \mid h_t^i)\}_{i \in N}$ ▷ Sample each agent's action from π^i
    $s_{t+1} \sim T(\cdot \mid s_t, a_t = \{a_t^i\}_{i \in N})$ ▷ Sample next state from GS
    $\{x_{t+1}^i\}_{i \in N} \leftarrow s_{t+1}$ ▷ Extract local states from global state
    $\{l_{t+1}^i \leftarrow \langle a_t^i, x_{t+1}^i \rangle\}_{i \in N}$ ▷ Append action-local-state pairs to each agent's ALSH
    $\{o_{t+1}^i \sim \dot{O}^i(\cdot \mid x_{t+1})\}_{i \in N}$ ▷ Sample each agent's observation from $\dot{O}^i$
    $\{h_{t+1}^i \leftarrow \langle a_t^i, o_{t+1}^i \rangle\}_{i \in N}$ ▷ Append actions-observation pairs to each agent's AOH
  end for
end for

B.4 Results

B.4.1 DIALS vs GS

The plots in Figures B.1 and B.2 show the learning curves of agents trained with the GS, DIALS, and untrained-DIALS on the 4 variants of the traffic and warehouse environments

Algorithm 5 Simulate agent i's trajectory with IALS

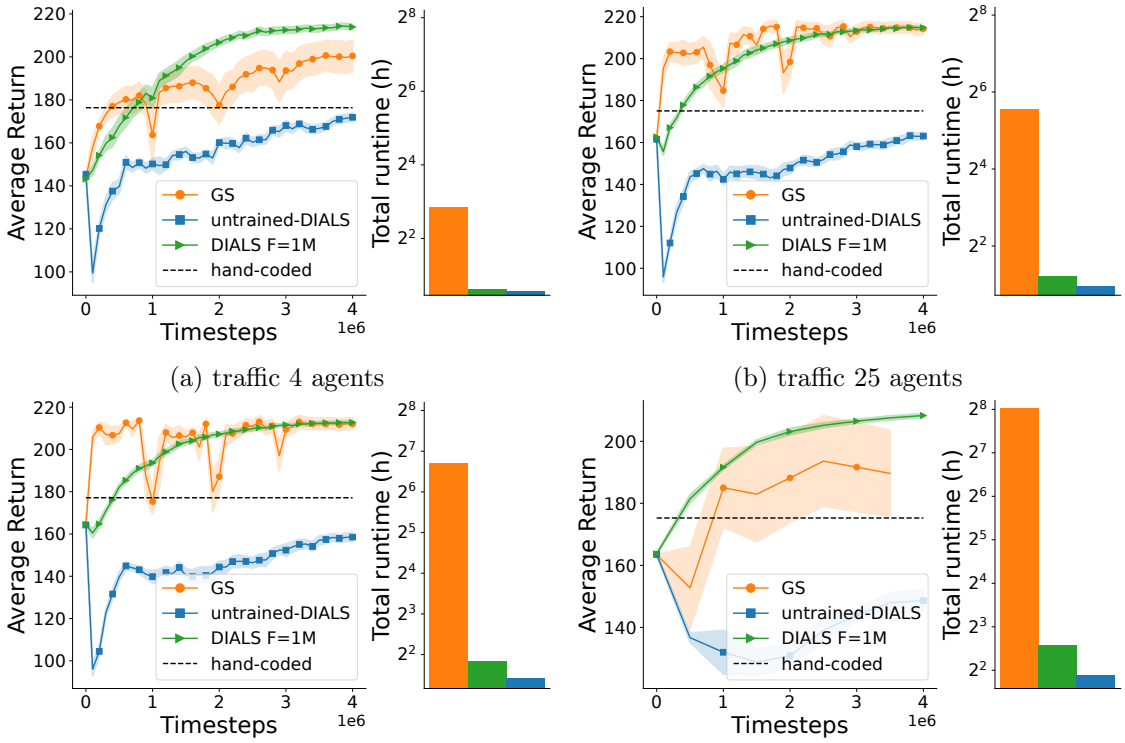
Input: $\dot{T}^i, \dot{R}^i, \dot{O}^i, \pi^i, \hat{I}_{\theta^i}$ ▷ local simulator, local reward and observation functions, policy, AIP
$x_0^i \leftarrow$ reset ▷ Reset initial state
$o_0^i \sim \dot{O}^i(\cdot|x_0^i)$ ▷ Sample observation from $\dot{O}^i$
$h_0^i \leftarrow o_0^i$ ▷ Initialize AOH with initial observation
for $t \in \langle 0, ..., T \rangle$ **do**
    $a_t^i \sim \pi(\cdot \mid h_t^i)$ ▷ Sample action
    $\dot{R}^i(x_t^i, a_t^i)$ ▷ Compute reward
    $u_t^i \sim \hat{I}_{\theta^i}(\cdot \mid l_t^i)$ ▷ Sample influence sources from AIP
    $x_{t+1}^i \sim \dot{T}(\cdot \mid x_t^i, a_t^i, u_t^i)$ ▷ Sample next local state from LS
    $l_{t+1}^i \leftarrow \langle a_t^i, x_{t+1}^i \rangle$ ▷ Append action-local-state pair to ALSH
    $o_{t+1}^i \sim \dot{O}^i(\cdot \mid x_{t+1}^i)$ ▷ Sample observation from O
    $h_{t+1}^i \leftarrow \langle a_t^i, o_{t+1}^i \rangle$ ▷ Append action-observation pair to AOH
end for

(4, 25, 49, and 100 agents). The bar plots show the total runtime of training for 4M timesteps with the three simulators. Shaded areas indicate the standard error of the mean.

The orange curves in Figures B.1d and B.2d stop at 3.5M and 2M timesteps, respectively. This is because the maximum execution time allowed by our computer cluster is 1 week, and training 100 agents with the GS takes longer. A breakdown of the runtimes for the three simulators is provided in Appendix B.7. Note that the runtime measurements were made on the only machine in our computer cluster with more than 100 CPUs. This is so that it would fit DIALS when training on the 100-agent variants. However, the experiments that required less than 100 CPUs were ran on different machines with different CPUs.

The bar plots indicate that DIALS is computationally more efficient and scales much better than GS. Note that the y axis is in $\log_2$ scale. Moreover, agents trained with DIALS seem to converge steadily towards similar high-performing policies in both environments, while agents trained with the GS suffer frequent performance drops and often get stuck in local minima. This is evidenced by the oscillations in the orange curves, the poor mean episodic reward, and large standard errors compared to the green (traffic) and purple (warehouse) curves. The plots also reveal that estimating the influence distributions correctly is important, as indicated by the large gap between DIALS and untrained-DIALS in both environments.

It is worth noting that the gap between GS and DIALS is larger in the warehouse (Figure B.2) than in the traffic environment (Figure B.1). We posit that this is because, in the



(a) traffic 4 agents

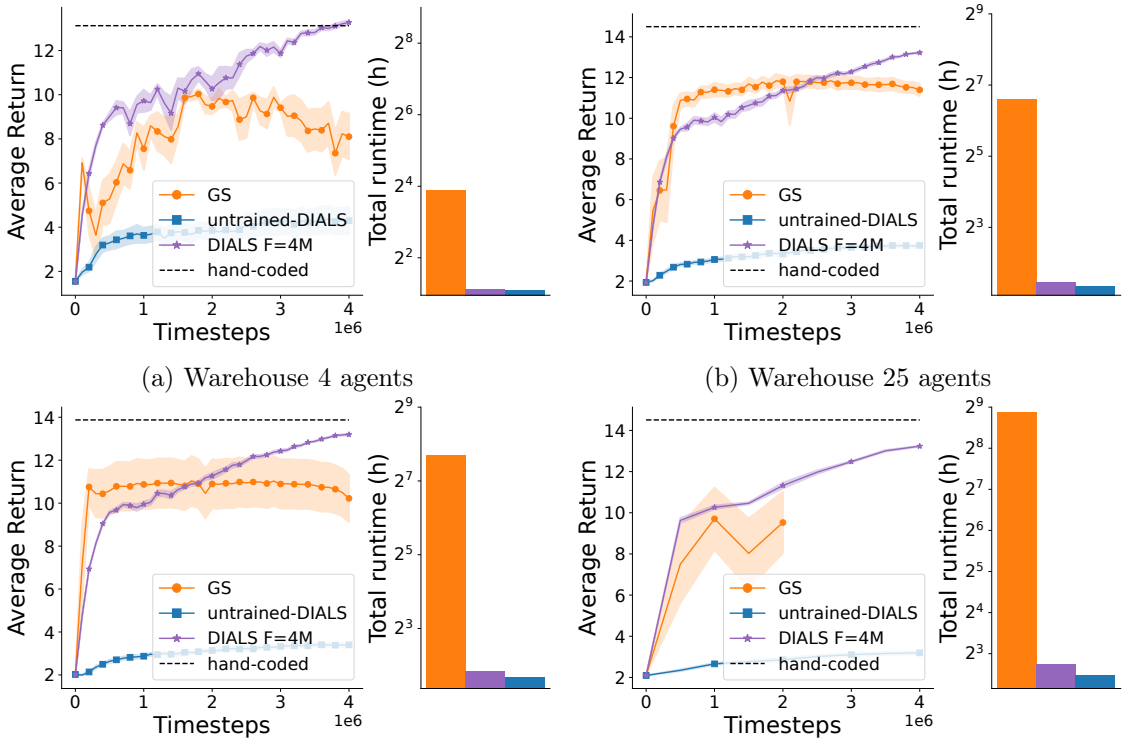
(b) traffic 25 agents

(c) traffic 49 agents

(d) traffic 100 agents

Figure B.1: **Left (a), (b), (c), and (d):** Average return as a function of the number of timesteps with GS, DIALS $F = 1$M, and untrained-DIALS on the traffic environment. **Right (a), (b), (c), and (d):** Total runtime of training for 4M timesteps, y-axis is in $\log_2$ scale.

warehouse environment, agents are more strongly coupled. To see this imagine that, by random chance during training, a robot starts favoring items from one shelf over the three others. The robot's neighbors might exploit this and start collecting items from the unattended shelves. However, as soon as this first robot changes its policy and starts collecting items more evenly from all four shelves, the neighbor robots will experience a sudden drop in the value of their policies, which can have catastrophic effects on the learning dynamics. With the DIALS, however, agents are trained on separate simulators and only become aware of changes in the joint policy when the AIPs are retrained. This prevents them from constantly co-adapting to one another. This is in line with our discussion in Section 4.4.3.



(a) Warehouse 4 agents

(b) Warehouse 25 agents

(c) Warehouse 49 agents

(d) Warehouse 100 agents

Figure B.2: **Left (a), (b), (c), and (d):** Average return as a function of the number of timesteps with GS, DIALS $F = 1$M, and untrained-DIALS on the warehouse environment. **Right (a), (b), (c), and (d):** Total runtime of training for 4M timesteps, y-axis is in $\log_2$ scale.

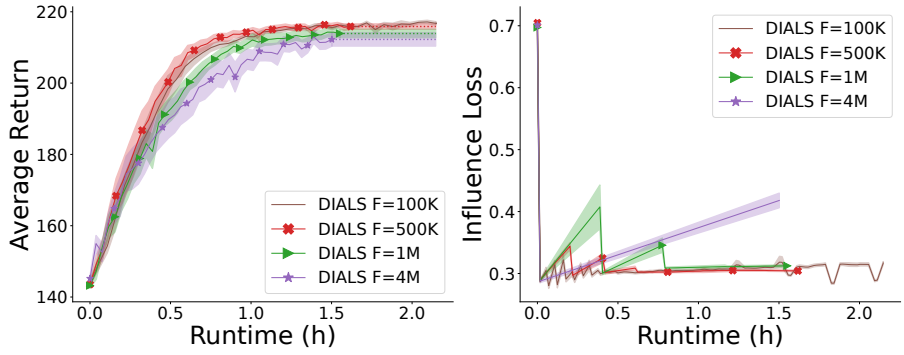
B.4.2 AIPs training frequency

The two plots on the left of Figures B.3 and B.4 show a comparison of the agents' average return as a function of runtime for different values of the AIPs training frequency parameter F (100K, 500K, 1M, and 4M timesteps). For ease of visualization, since DIALS $F = 500$K, $F = 1$M, and $F = 4$M take shorter to finish than DIALS $F = 100$K, the red, green, and purple curves are extended by dotted horizontal lines. Due to computational limitations, we ran these experiments only on the 4, 25, and 49-agent variants of the two environments. We then chose the best-performing values for F ($F = 1$M for traffic and $F = 4$M for warehouse) and used those to run DIALS on the environments with 100 agents.

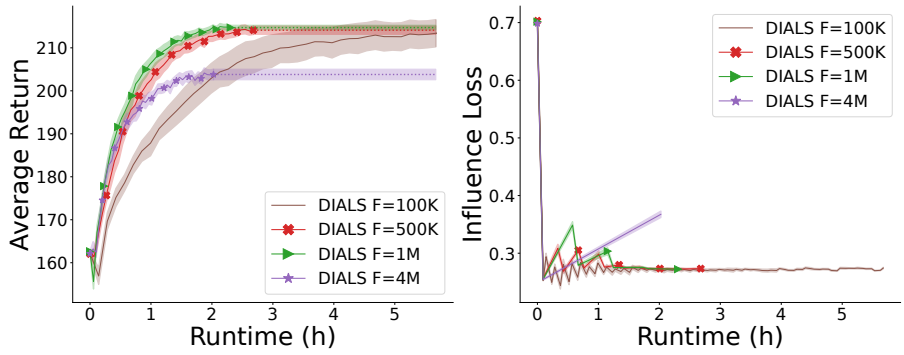
In the traffic domain, the gap between the green and the purple curve (Figure B.3) suggests that it is important to retrain the AIPs at least every 1M timesteps, such that agents

become aware of changes in the other agents' policies. This is consistent on all the three variants (Figures B.3a, B.3b, and B.3c). In contrast, in the warehouse domain (Figure B.4), we see that training the AIPs only once at the beginning (DIALS $F = 4\text{M}$) is sufficient (Figures B.4a, B.4b, and B.4c). In fact, as indicated by the gap between the brown and the rest of the curves, updating the AIPs too frequently (DIALS $F = 100\text{K}$), aside from increasing the runtimes, seems detrimental to the agents' performance. This is consistent with our hypothesis in Section 4.4.3: "by not updating the AIPs too frequently, we get a biased but otherwise more consistent learning signal that the agents can rely on to improve their policies."

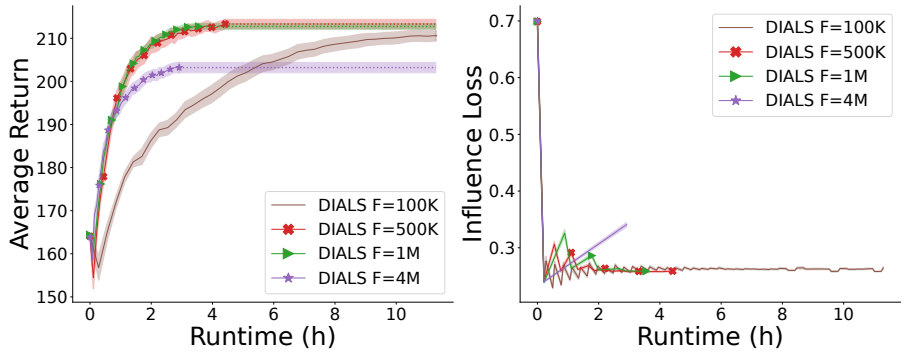
The plots on the right of Figures B.3 and B.4 show the average cross-entropy (CE) loss of the AIPs evaluated on trajectories sampled from the GS. As explained in Section 4.4 since all agents learn simultaneously, the influence distributions $\{I(u_t^i|l_t^i)\}_{i\in N}$ are non-stationary. For this reason, we see that the CE loss changes as the policies of the other agents are updated. We can also see how the CE loss decreases when the AIPs are retrained, which happens more or less frequently depending on the hyperparameter F. Note that the CE not only measures the distance between the two probability distributions but also the absolute entropy. In the warehouse domain (Figure B.4), the neighbor robots' locations become more predictable (lower entropy) as their policies improve. This explains why the CE loss decreases even though the AIPs are not updated. Also note that, in the warehouse environment (Figure B.4), even though by the end of training DIALS $F = 4\text{M}$ is highly inaccurate, as evidenced by the gap between the purple and the other curves, it is still good enough to train policies that match the performance of those trained with DIALS $F = 500\text{K}$ and $F = 1\text{M}$. This is in line with our results in Section 4.4: "Multiple influence distributions may induce the same optimal policy."



(a) Traffic 4 agents

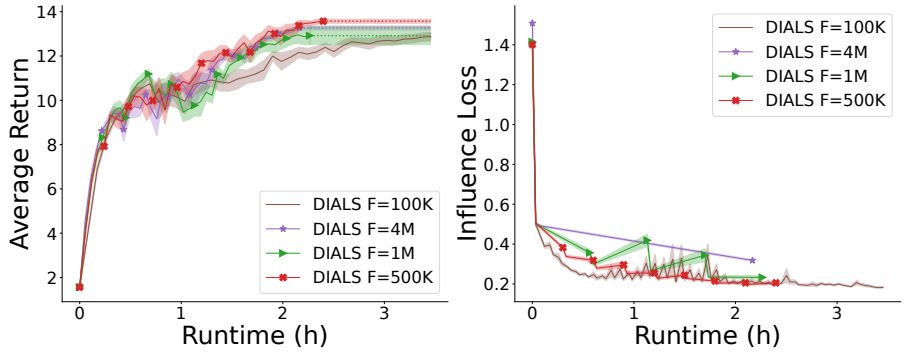


(b) Traffic 25 agents

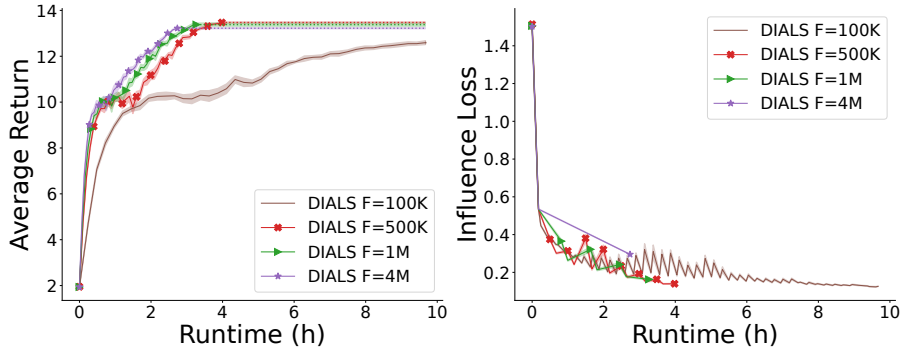


(c) Traffic 49 agents

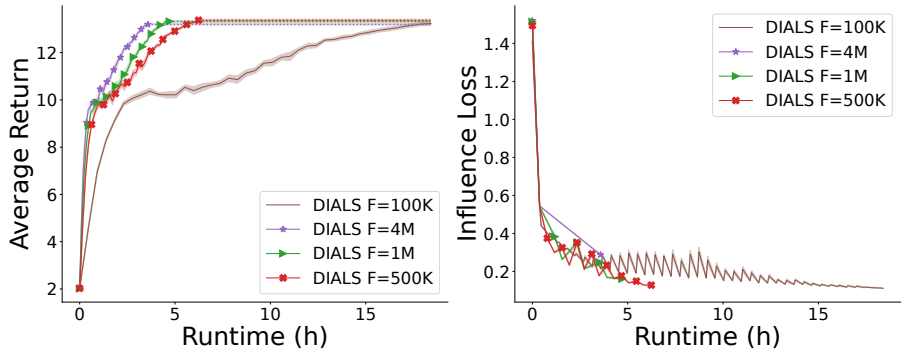
Figure B.3: **Left (a), (b), and (c):** Learning curves for different values of F on the 4, 25, and 49 agent versions of the traffic environment. **Right (a), (b), and (c):** CE loss of the AIPs as a function of runtime.



(a) Warehouse 4 agents



(b) Warehouse 25 agents



(c) Warehouse 49 agents

Figure B.4: **Left (a), (b), (c), and (d):** Average return as a function of the number of timesteps with GS, DIALS $F = 1$M, and untrained-DIALS on the warehouse environment. **Right (a), (b), (c), and (d):** Total runtime of training for 4M timesteps, y-axis is in $\log_2$ scale.

B.5 Implementation Details

B.5.1 Approximate Influence Predictors

Due to the sequential nature of the problem, rather than feeding the full past history every time we make a prediction, we use a recurrent neural network (RNN) (Hochreiter and Schmidhuber, 1997; Cho et al., 2014) and process observations one at a time,

$$P(u_t|l_t) \approx \hat{I}_\theta(u_t|\hat{h}_{t-1}, o_t) = F_{\text{rnn}}(\hat{h}_{t-1}, o_t, u_t), \tag{B.21}$$

where we use $\hat{h}$ to indicate that the history h is embedded in the RNN's internal memory.

Given that we generally have multiple influence sources $u_t = \langle u_t^1, ..., u_t^M \rangle$, we need to fit M separate models $\hat{I}_{\theta_m}$ to predict each of the M influence sources. In practice, to reduce the computational cost, we can have a single network with a common representation module for all influence sources and output their probability distributions using M separate heads. This representation assumes that the influence sources are independent of one another,

$$I(u_t|l_t) = \prod_{m=0}^{M} P(u_t^m|l_t), \tag{B.22}$$

which is true for the two domains we study in this paper.

Finally, although according to the POMDP framework we should condition the AIPs on the full AOH, in many domains, one can exploit the structure of the transitions function to find a subset of variables in the AOH that is sufficient to predict the next observation. This subset is known as the d-separating set (Oliehoek et al., 2021), and as shown in Suau et al. (2022b) conditioning the AIPs on this rather than the full AOH can ease the task of approximating the influence distribution.

B.5.2 Local regions

When choosing the local regions to build the simulators, the only restriction in terms of size is that these should contain all the necessary information to compute local observations and rewards. In our experiments, we use one simulator per agent since, given that the simulators run in parallel, this is the most computationally efficient way of factorizing the environment. Yet, in certain applications, due to hardware limitations (e.g. not enough CPUs or memory available), it might be necessary to partition the environment into fewer local regions than the number of agents in the environment. Moreover, in

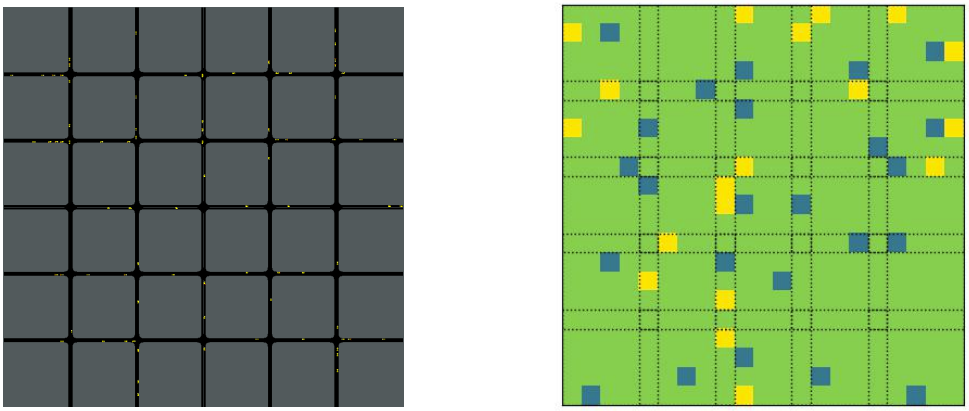


Figure B.5: A screenshot of the global simulators for the 25-agent variants of the traffic control (left) and warehouse (right) environments

some environments (including the two we explore here) better results may be obtained by grouping some of them together in the same simulator. In fact, one could potentially treat the agents in the same group/simulator as a single agent and train a policy to control all of them simultaneously. Note, however, that this is orthogonal to our work as we are mainly concerned with computational speedups.

B.6 Simulators

Figure B.5 shows two screenshots of the global simulator (GS) for the traffic (left) and warehouse (right) environments with 25 agents each. Figure B.6 shows two screenshots of the local simulator (LS) for the traffic (left) and warehouse environments (right). Since all local regions are the same (i.e. $\dot{T}^i$, $\dot{R}^i$, and $\dot{O}^i$ do not change) in the two environments, we use the same LS for all of them. However, because depending on where these are located they are influenced differently by the rest of the system, we train separate AIPs, $\{\hat{I}_{\theta^i}\}_{i \in N}$, for each of them. Note that, we chose the local regions to be the same for simplicity. However, the method can readily be applied to environments with different local transition dynamics $\dot{T}^i$, different local observations $\dot{O}^i$, and/or different local rewards $\dot{R}^i$ for every agent $i \in N$.

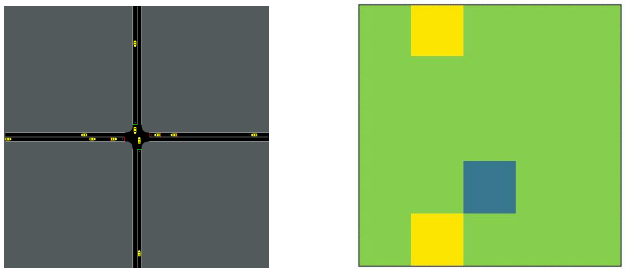


Figure B.6: A screenshot of the local simulators for the traffic (left) and warehouse (right) environments. Since all local regions are the same in the two environments, we use the same LS for all of them.

B.7 Runtimes

The two tables below show a breakdown of the runtimes for the two environments and the three simulators. These were measured on a machine with 128 CPUs of the type AMD EPYC 7452 32-Core Processor. We used this machine for all our measurements because it is the only one in our computer cluster that can fit DIALS when training on the 100-agent variants of the environments. However, the experiments that required less than 100 CPUs were actually run on different machines.

Table B.1: Runtimes for the traffic control environment

	Agents training (h)				Data collection + influence training (h)				Total (h)			
Number of agents	2	25	49	100	2	25	49	100	2	25	49	100
GS	7.24	46.96	105.41	261.06	-	-	-	-	7.24	46.96	105.41	261.06
DIALS F=100K	1.48	1.93	2.70	3.70	0.66	3.74	8.60	22.38	2.14	5.67	11.30	26.08
DIALS F=500K	1.48	1.93	2.70	3.70	0.13	0.75	1.72	4.48	1.61	2.68	4.42	8.18
DIALS F=1M	1.48	1.93	2.70	3.70	0.07	0.37	0.86	2.24	1.55	2.30	3.56	5.94
DIALS F=4M	1.48	1.93	2.70	3.70	0.02	0.09	0.21	0.56	1.50	2.02	2.91	4.26
untrained-DIALS	1.48	1.93	2.70	3.70	-	-	-	-	1.48	1.93	2.70	3.70

Table B.2: Runtimes for the warehouse environment

	Agents training (h)				Data collection + influence training (h)				Total (h)			
Number of agents	2	25	49	100	2	25	49	100	2	25	49	100
GS	14.84	97.04	208.18	468.46	-	-	-	-	14.84	97.04	208.18	468.46
DIALS F=100K	2.13	2.56	3.19	5.55	1.32	7.11	15.19	45.45	4.45	9.67	18.38	51.00
DIALS F=500K	2.13	2.56	3.19	5.55	0.26	1.42	3.04	9.09	2.39	3.98	6.23	14.64
DIALS F=1M	2.13	2.56	3.19	5.55	0.13	0.71	1.52	4.54	2.26	3.27	4.71	10.09
DIALS F=4M	2.13	2.56	3.19	5.55	0.03	0.18	0.38	1.13	2.16	2.74	3.57	6.68
untrained-DIALS	2.13	2.56	3.19	5.55	-	-	-	-	2.13	2.56	3.19	5.55

B.8 Memory Usage

The table below shows the peak memory usage of the GS and the DIALS. For the latter we provide the memory usage per process and in total. The memory needed for the GS seems to grow logarithmically with the number of agents, whereas for DIALS the memory usage per process stays relatively constant. However, the total amount of memory needed to run DIALS (aggregate of all processes) increases linearly with the number of agents and is considerably larger than that of the GS.

Table B.3: Peak Memory Usage in Megabytes (MB)

Environment		Traffic				Warehouse			
Number of agents		4	25	49	100	4	25	49	100
GS		375.3	392.7	412.5	457.4	339.3	391.8	469.6	607.4
DIALS	Per process	219.5	221.0	225.8	228.7	195.6	201.9	203.7	207.5
	Total	878.0	5525.0	11064.2	22870.0	782.4	5047.5	9981.3	20750.0

B.9 Hyperparameters

The hyperparameters used for the AIPs are reported in Table B.4. Since feeding past local states did not seem to improve the performance of the AIPs in the traffic environment we modeled them with FNNs. In contrast, adding the past ALSHs does decrease the CE loss in the warehouse environment, and thus we used GRUs (Cho et al., 2014) instead. The

size of the networks was chosen as a compromise between low CE loss and computational efficiency. On the one hand, we need accurate AIPs to properly capture the influence distributions. On the other, we also want them to be small enough such that we can make fast predictions. The hyperparameter named seq. length determines the number of timesteps the GRU is backpropagated. This was chosen to be equal to the horizon such that episodes did not have to be truncated. The rest of the hyperparameters in Table B.4, which refer to the training setup for the AIPs, were manually tuned.

Table B.4: Hyperparameters for approximate influence predictors (AIPs).

	Architecture	Num. layers	Num. neurons	Seq. length	Learning rate	Dataset size	Batch size	Num. epochs
Traffic	FNN	2	128 and 128	-	1e−4	1e4	128	100
Warehouse	GRU	2	64 and 64	100	1e−4	1e4	32	300

The hyperparmeters used for the policy networks are given in Table B.5. We chose again GRUs for the warehouse environment and FNNs for the traffic domain, since feeding the previous AOHs did not seem to improve the agents' performance in the latter. The network size and the sequence length parameter for the GRUs were manually tuned on the smallest scenarios with 4 agents.

Table B.5: Hyperparmeters for policy networks.

	Architecture	Num. layers	Num. neurons	Seq. length
Traffic	FNN	2	256 and 128	-
Warehouse	GRU	2	256 and 128	8

As for the hyperparameters specific to PPO (Table B.6), we used the same values reported by (Schulman et al., 2017), and only tuned the parameter T, which depends on the rewards and the episode length. T determines for how many timesteps the value function is rollout before computing the value estimates.

Table B.6: PPO hyperparameters.

Rollout steps T	16 traffic and 8 warehouse
Learning rate	2.5e-4
Discount γ	0.99
GAE λ	0.95
Memory size	128
Batch size	32
Num. epoch	3
Entropy β	1.0e-2
Clip ϵ	0.1
Value coeff. c_1	1

Appendix C

Influence-Aware Memory

C.1 Implementation details

Warehouse and traffic control: The FNN and the LSTM baselines that we used in the warehouse and the traffic control environments, have two hidden layers. Only the second layer in the LSTM baseline is recurrent. This seems to work much better than the reverse option (i.e. first layer recurrent and second layer feedforward). Moreover, to ensure a fair comparison, the size of the first layer in IAM is set equal to the size of the first layer in the FNN and the LSTM baselines. That is, adding together the number neurons in the FNN and the RNN channels. For instance, if the first layer of the FNN and the LSTM baselines is 640 neurons, a valid configuration for IAM would be 512 feedfoward and 128 recurrent neurons. Such that the FNN and the RNN combined also add up to 640. When doing observation-stacking the FNN baseline is provided the last 8 observations in the warehouse and 32 observations in the traffic environment. On the other hand, the gradients in the recurrent architectures (LSTM and IAM) are backpropagated for 8 and 32 timesteps (sequence length parameter). We did extensive testing to try to find the best possible network configuration for each model. These are reported in Table C.1 for the warehouse and Table C.2 for the traffic environment. We tried different combinations of $\{128, 256, 512, 640\}$ and $\{32, 64, 128, 256\}$ for the size of the first and second feedfoward layers, respectively. As reported in section 5.3 (Figure 3), we also tried with $\{8, 16, 32, 64, 128, 256\}$ neurons for the recurrent layers in LSTM and IAM.

Table C.1: Hyperparameters used for Warehouse Commissioning.

Warehouse						
Model	FNN		LSTM		IAM	
num frames	8		1		1	
time horizon	16		16		16	
seq length	-		8		8	
FNN	layer 1	layer 2	layer 1	-	layer 1	layer 2
neurons	640	256	640		512	256
RNN	None		layer 2		layer 1	
rec neurons	-		128		128	

Table C.2: Hyperparameters used for Traffic Control.

Traffic Control						
Model	FNN		LSTM		IAM	
num frames	32		1		1	
seq length	-		32		32	
time horizon	128		128		128	
FNN	layer 1	layer 2	layer 1	-	layer 1	layer 2
neurons	256	64	256		128	64
RNN	None		layer 2		layer 1	
rec neurons	-		128		128	

Flickering Atari: As explained in Section 4.1, when the observations are images, instead of feeding them directly into the FNN or the RNN, we first preprocess them with a CNN. The FNN and LSTM baselines use the same architecture reported by Mnih et al. (2015) and Hausknecht and Stone (2015), respectively. The IAM maintains the same CNN configuration and then connects 128 feedforward and 128 recurrent neurons in parallel so that the total number of neurons (256) remains the same as in the FNN and LSTM baselines. Additionally, the attention mechanism, which computes A_t for every $\langle o_t, \hat{d}_t \rangle$, consists of a single head with 256 neurons. The FNN baseline receives the last 8 frames as input, whereas LSTM and IAM only receive 1 frame. To update the network, gradients in the recurrent models are backpropagated for 8 timesteps. The network configurations used for Flickering Atari, are provided in Table C.3.

As for the hyperparameters specific to PPO we used the same values reported by Schulman

Table C.3: Hyperparameters used for Flickering Atari.

Flickering Atari									
Model	FNN			LSTM			IAM		
num frames	8			1			1		
seq length	-			8			8		
time horizon	128			128			128		
CNN	layer 1	layer 2	layer 3	layer 1	layer 2	layer 3	layer 1	layer 2	layer 3
filters	32	64	64	32	64	64	32	64	64
kernel size	8	4	3	8	4	3	8	4	3
strides	4	2	1	4	2	1	4	2	1
FNN	layer 1			None			layer 1		
neurons	512			-			256		
Attention	None			None			layer 1		
neurons	-			-			256		
num heads	-			-			1		
RNN	None			layer 1			layer 1		
rec neurons	-			512			256		

et al. (2017) for all three models and the three domains. These are shown in Table C.4.

Table C.4: PPO hyperparameters.

learning rate	2.5e-4
discount γ	0.99
GAE λ	0.95
memory size	128
batch size	32
num. epoch	3
num. workers	8
entropy β	1.0e-2
clip ϵ	0.1
value coeff. c_1	1

C.2 Runtime performance

Table C.5 is an empirical analysis of the runtime performance of the three architectures. The values are the average wall-clock time in milliseconds of evaluating (forward pass)

and updating (backward pass) the models. The FNN baseline receives a stack of 8, 32 and 8 observations in the Warehouse, Traffic Control and Flickering Atari environments, respectively. On the other hand, gradients in LSTM and IAM are backpropagated for 8, 32 and 8 timesteps when updating the networks in each of the three environments. An evaluation corresponds to a forward pass through the network. One update involves 3 epochs over a batch of 1024 experiences. The test was run on an Intel Xeon CPU E5-1620 v2.

Table C.5: Runtime performance in milliseconds

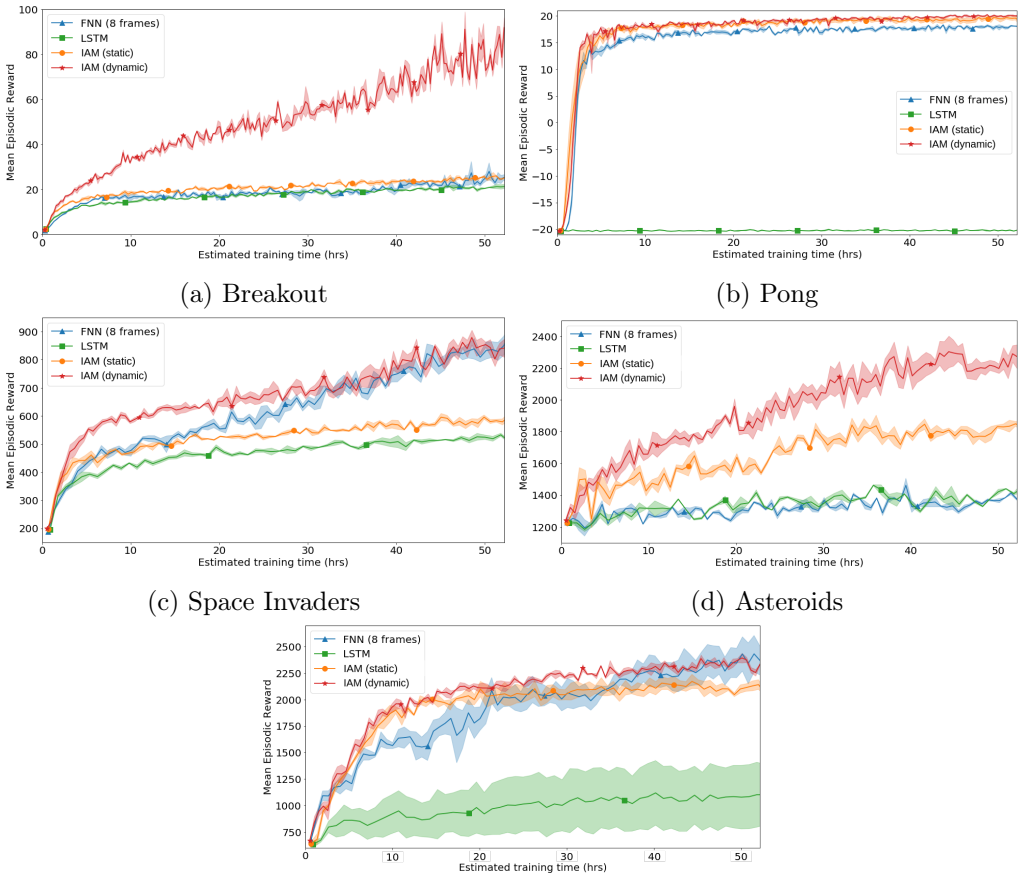
	Warehouse		Traffic		Flickering Atari	
	Evaluation	Update	Evaluation	Update	Evaluation	Update
FNN (obs-stacking)	1.03 ± 0.41	81.96 ± 1.29	0.90 ± 0.16	63.21 ± 0.71	3.82 ± 0.53	215.95 ± 7.54
LSTM	1.27 ± 0.27	151.10 ± 2.98	1.27 ± 0.12	167.36 ± 10.41	5.41 ± 1.13	285.19 ± 24.60
IAM	1.32 ± 0.50	82.60 ± 4.87	1.05 ± 0.13	144.96 ± 13.51	3.47 ± 1.84	163.30 ± 28.19

C.3 Learning curves

Figure C.1 shows the mean episodic reward comparison of the two baselines FNN and LSTM and the two versions of IAM, static and dynamic, as a function of training time on the *flickering* Atari games. Since we could not run all our experiments on the same machine, training time is estimated using the values provided in Table C.5.

C.4 Results on the original Atari games

We also tested IAM on the original Atari games. Although the full game screen is visible at all times, some of the games contain moving elements whose speed and direction can not be measured from just a single frame. This means that the original games are also POMDPs (Hausknecht and Stone, 2015). In DQN (Mnih et al., 2015) this issue is easily solved by stacking the last 4 frames and feeding them into the network. Our experiments show (Table C.6), that even when frame-stacking seems to be the optimal solution, IAM can reach the same or even better performance of the FNN model while clearly outperforming the LSTM baseline.



(a) Breakout

(b) Pong

(c) Space Invaders

(d) Asteroids

(e) MsPacman

Figure C.1: Average score per episode as a function of training time. Shaded areas show the standard deviation of the mean.

C.5 Decoding the agent's internal memory

The memory decoder consists of a two-layer fully connected FNN which takes as input the RNN's internal memory $\hat{d}_{t-1}$ and the output x_t of the FNN (see Figure 3) and outputs a prediction for each of the pixels in the screen. The network is trained on a dataset containing screenshots and their corresponding network activations ($\hat{d}_{t-1}$ and x_t), using the pixel-wise cross entropy loss. The dataset is collected after training the agent's policy. The images are first transformed to grey-scale and then normalized to simplify the task. The results are shown in Figure C.2. The goal of this experiment was to confirm that although the RNN is only fed the last two elements of the binary vectors representing each lane, it can still uncover hidden state. It is important to point out that the agent is only trained to maximize the reward and has no explicit knowledge of the environment

Table C.6: Average score per episode and standard deviation over the last 200K timesteps after 50 hours of training on the original (non-flickering) Atari games. The scores are also averaged over three trials. Bold numbers indicate the best results on each environment. Multiple results are highlighted when the differences are not statistically significant.

	FNN (4 frames)	LSTM	IAM (static)	IAM (dynamic)
Breakout	$\mathbf{326.54 \pm 22.21}$	127.37 ± 26.13	295.73 ± 31.40	$\mathbf{339.42 \pm 21.96}$
Pong	$\mathbf{20.68 \pm 0.07}$	-20.26 ± 0.04	$\mathbf{20.73 \pm 0.08}$	$\mathbf{20.74 \pm 0.04}$
Space Invaders	975.39 ± 70.06	1177.47 ± 24.81	$\mathbf{1326.63 \pm 53.59}$	868.23 ± 37.34
Asteroids	2184.61 ± 62.83	1618.89 ± 41.93	2057.21 ± 109.65	$\mathbf{2792.45 \pm 60.40}$
MsPacman	$\mathbf{3859.91 \pm 751.30}$	667.58 ± 32.39	$\mathbf{4016.78 \pm 273.11}$	3294.29 ± 222.64

dynamics. A video of this experiment can be found at `https://tinyurl.com/y9cvuz7l`.

C.6 Analysis of the hidden activations

We further analyze the information contained in $\hat{d}$ when playing Breakout (non-flickering) by looking at the activation patterns at different timesteps and their correlation with the ball velocity. The velocity vector v is computed by comparing the location of the ball at two subsequent frames. We use Canonical Correlation Analysis (CCA) (Hotelling, 1992) to obtain a lower dimensional representation of $\hat{d}$. In broad terms, CCA is a linear transformation that projects two sets of variables, in this case $\hat{d}$ and v, onto two subspaces that are highly correlated. This is done by iteratively finding linear combinations of $\hat{d}$ and v, known as canonical components, that maximize the correlation between them while being uncorrelated with the previous canonical components. We found that the first two canonical components of $\hat{d}$ are highly correlated with the two coordinates of the velocity vector (**0.98** and **0.90**).

We also show that the FNN's output x, on the other hand, contains information that does not need to be memorized but is relevant for predicting action values. We applied CCA to compare x with the number of bricks destroyed at each frame and obtained a correlation coefficient of **0.96**. The projections of the $\hat{d}$ and x onto the space spanned by their corresponding cannonical components are shown in the two scatter plots on the left of Figure 7 in the paper.

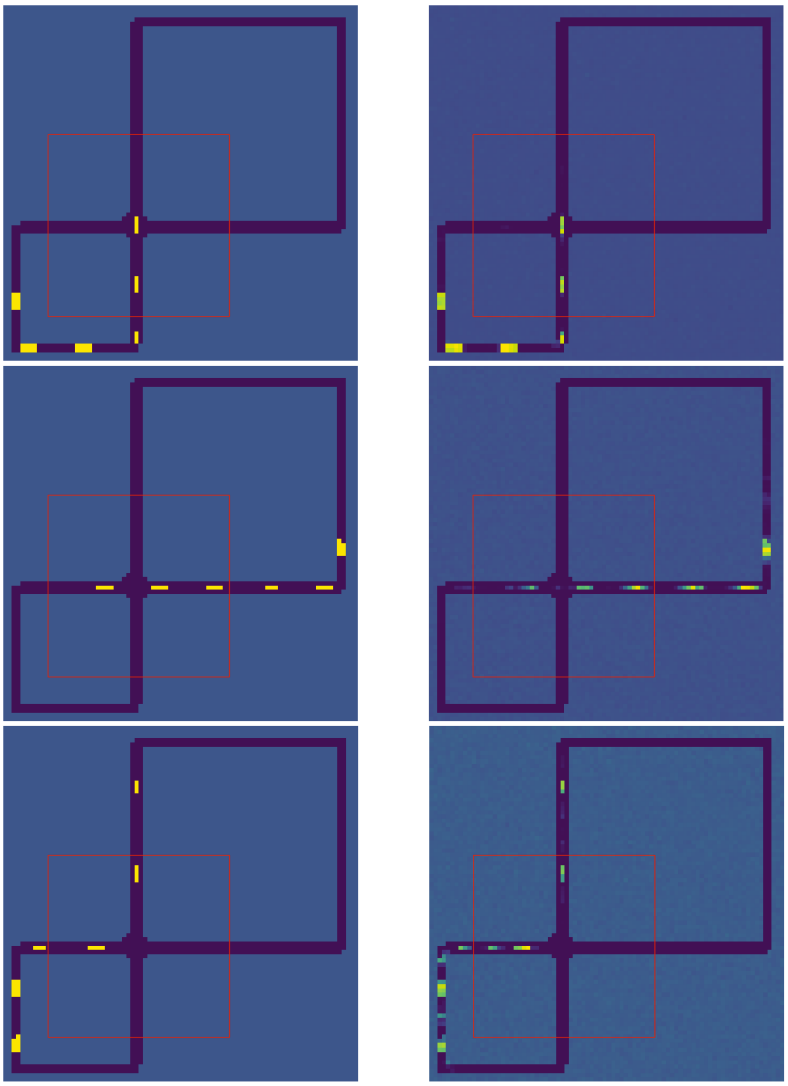


Figure C.2: True state (left) and state predicted by the memory decoder (right)

C.7 Attention maps

Here we include a collection of images which show the weights that the attention mechanism gives to the different regions of the game screen based on the agent's internal memory $\hat{d}$ and the current observation $\hat{o}_t$. The network seems to learn to focus on the regions that are important to memorize.

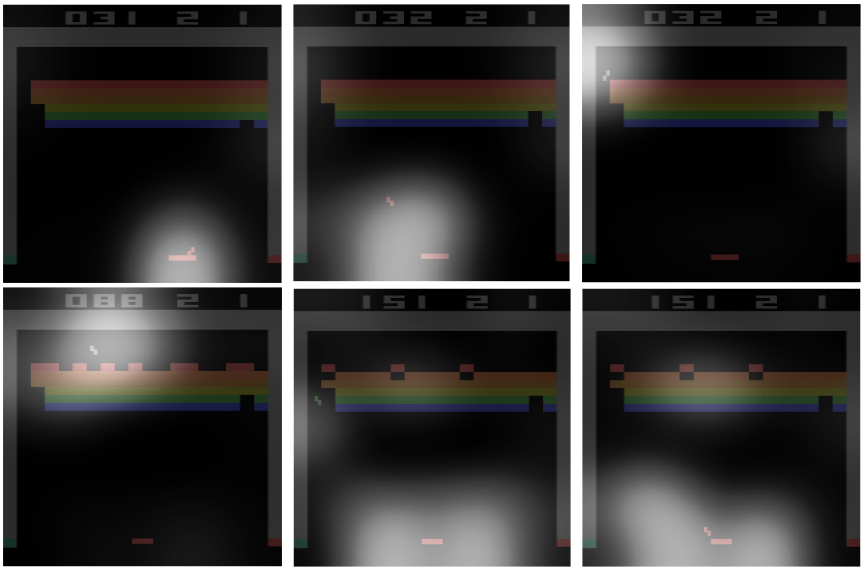


Figure C.3: Breakout

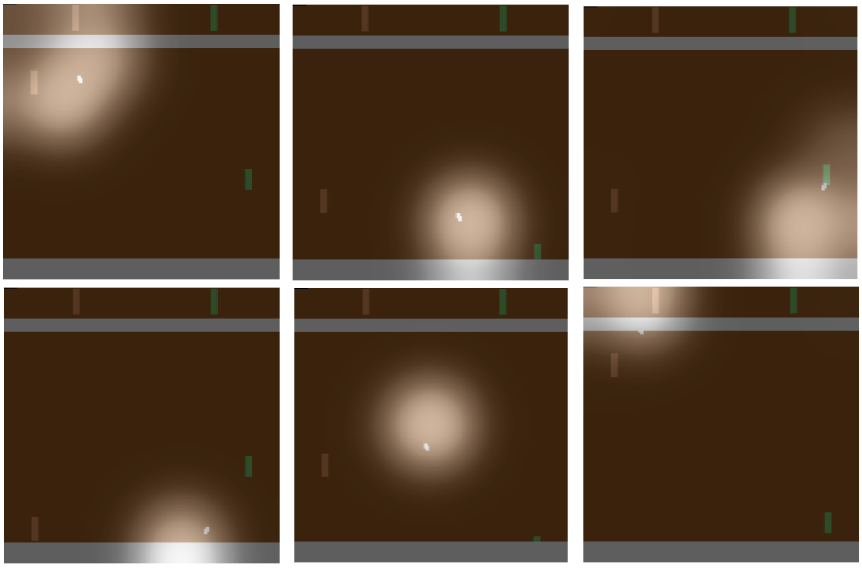


Figure C.4: Pong

Appendix D

Policy Confounding

D.1 Proofs

Lemma D.1.1. *Let $\mathbf{\Phi}^{\pi_1 *}$ be the set of all possible π-minimal state representations under π_1, where every $\Phi^{\pi_1 *} \in \mathbf{\Phi}^{\pi_1 *}$ is defined as $\Phi^{\pi_1 *} : \mathcal{S}^{\pi_1} \to \bar{\mathcal{S}}^{\pi_1 *}$ and $\bar{\mathcal{S}}^{\pi_1 *} = \times_i \mathrm{dom}(\bar{\Theta}^{\pi_1 * i})$, and let π_2 be a second policy such that for all $s_t \in \mathcal{S}^{\pi_1} \cap \mathcal{S}^{\pi_2}$,*

$$\mathrm{supp}\left(\pi_2(\cdot \mid s_t)\right) \subseteq \mathrm{supp}\left(\pi_1(\cdot \mid s_t)\right).$$

For all $\Phi^{\pi_1 *} \in \mathbf{\Phi}^{\pi_1 *}$, there exists a π-Markov state representation under policy π_2, $\Phi^{\pi_2} : \mathcal{S}^{\pi_2} \to \bar{\mathcal{S}}^{\pi_2}$ with $\bar{\mathcal{S}}^{\pi_2} = \times_i \mathrm{dom}(\bar{\Theta}^{\pi_2 i})$, such that $\bar{\Theta}^{\pi_2} \subseteq \bar{\Theta}^{\pi_1 *}$ for all $s_t \in \mathcal{S}^{\pi_1} \cap \mathcal{S}^{\pi_2}$. Moreover, there exist cases where $\bar{\Theta}_t^{\pi_2} \neq \bar{\Theta}_t^{\pi_1 *}$.

Proof. First, it is easy to show that

$$\forall s_t \in \mathcal{S}, \mathrm{supp}\left(\pi_2(\cdot \mid s_t)\right) \subseteq \mathrm{supp}\left(\pi_1(\cdot \mid s_t)\right) \iff \mathcal{S}^{\pi_2} \subseteq \mathcal{S}^{\pi_1},$$

and

$$\forall s_t \in \mathcal{S}, \mathrm{supp}\left(\pi_2(\cdot \mid s_t)\right) = \mathrm{supp}\left(\pi_1(\cdot \mid s_t)\right) \iff \mathcal{S}^{\pi_2} = \mathcal{S}^{\pi_1}.$$

In particular, $\mathcal{S}^{\pi_2} \subset \mathcal{S}^{\pi_1}$ if there is at least one state $s_t' \in \mathcal{S}^{\pi_1} \cap \mathcal{S}^{\pi_2}$ such that

$$\mathrm{supp}\left(\pi_2(\cdot \mid s_t')\right) \subset \mathrm{supp}\left(\pi_1(\cdot \mid s_t')\right)$$

while

$$\text{supp}\left(\pi_2(\cdot \mid s_t)\right) = \text{supp}\left(\pi_1(\cdot \mid s_t)\right)$$

for all other $s_t \in \mathcal{S}^{\pi_1} \cap \mathcal{S}^{\pi_2}$.

In such cases, we know that there is at least one action a' for which $\pi_2(a'_t \mid s'_t) = 0$ but $\pi_1(a'_t \mid s'_t) \neq 0$. Hence, if there was a state (or group of states) that could only be reached by taking action a'_t at s'_t, π_2 would never visit it and thus $\mathcal{S}^{\pi_2} \subset \mathcal{S}^{\pi_1}$.

Further, if $\mathcal{S}^{\pi_2} \subset \mathcal{S}^{\pi_1}$, we know that, for every $\Phi^{\pi_1 *} \in \mathbf{\Phi}^{\pi_1 *}$, there must be a $\Phi^{\pi_2 *}$ that requires, at most, the same number of variables, $\bar{\Theta}_t^{\pi_2} \subseteq \bar{\Theta}_t^{\pi_1 *}$ and, in some cases, fewer, $\bar{\Theta}_t^{\pi_1 *} \neq \bar{\Theta}_t^{\pi_2 *}$ (e.g., Frozen T-Maze example).

□

Proposition 6.6.1. *Let $\mathbf{\Phi}^*$ be the set of all possible minimal state representations, where every $\Phi^* \in \mathbf{\Phi}^*$ is defined as $\Phi^* : \mathcal{S} \to \bar{\mathcal{S}}^*$ with $\bar{\mathcal{S}}^* = \times_i \text{dom}(\bar{\Theta}^{*i})$. For all π and all $\Phi^* \in \mathbf{\Phi}^*$, there exists a π-Markov state representation $\Phi^\pi : \mathcal{S}^\pi \to \bar{\mathcal{S}}^\pi$ with $\bar{\mathcal{S}}^\pi = \times_i \text{dom}(\bar{\Theta}^{\pi i})$ such that for all $s \in \mathcal{S}^\pi$, $\bar{\Theta}^\pi \subseteq \bar{\Theta}^*$. Moreover, there exist cases for which $\bar{\Theta}^\pi$ is a proper subset, $\bar{\Theta}^\pi \neq \bar{\Theta}^*$.*

Proof. The proof follows from Lemma D.1.1. We know that, in general, $\mathcal{S}^\pi \subseteq \mathcal{S}$, and if $\pi(a'_t|s'_t) = 0$ for at least one pair $a'_t \in \mathcal{A}, s'_t \in \mathcal{S}$ for which there is a state (or group of states) that can only be reached by taking action a'_t at s'_t, then $\mathcal{S}^\pi \subset \mathcal{S}$. Hence, for every Φ^* there is a Φ^π such that $\bar{\Theta}^\pi \subseteq \bar{\Theta}^*$, and in some cases, we may have $\bar{\Theta}^\pi \neq \bar{\Theta}^*$ (e.g., Frozen T-Maze example).

□

Theorem 6.6.1. *Let $\Phi^* : \mathcal{S} \to \bar{\mathcal{S}}^*$ with $\bar{\mathcal{S}}^* = \times_i \text{dom}(\bar{\Theta}^{*i})$ be a minimal state representation. If, for some π, there is a π-Markov state representation $\Phi^\pi : \mathcal{S}^\pi \to \bar{\mathcal{S}}^\pi$ with $\bar{\mathcal{S}}^\pi = \times_i \text{dom}(\bar{\Theta}^{\pi i})$, such that $\bar{\Theta}^\pi \subset \bar{\Theta}^*$ for some $s \in \mathcal{S}$, then Φ^π is confounded by policy π.*

Proof. Proof by contradiction. Let us assume that $\bar{\Theta}^\pi \subset \bar{\Theta}^*$, and yet there is no policy confounding. I.e., for all $s_t, s_{t+1} \in \mathcal{S}$, $a_t \in \mathcal{A}$,

$$R^\pi(\Phi^\pi(s_t), a_t) = R^\pi(\text{do}(\Phi^\pi(s_t)), a_t) \tag{D.1}$$

and

$$\overset{\pi}{\Pr}(\Phi^\pi(s_{t+1}) \mid \Phi^\pi(s_t), a_t) = \overset{\pi}{\Pr}(\Phi^\pi(s_{t+1}) \mid \text{do}(\Phi^\pi(s_t)), a_t) \tag{D.2}$$

First, note that the do-operator implies that the equality must hold for *all* s'_t in the equivalence of s_t class under Φ^π, $s'_t \in \{s_t\}^{\Phi^\pi} = \{h'_t \in H_t : \Phi(h'_t) = \Phi(h_t)\}$, i.e., not just those h'_t that are visited under π,

$$R^\pi(\Phi^\pi(s_t), a_t) = R^\pi(\text{do}(\Phi^\pi(s_t)), a_t) = \{R(s'_t, a_t)\}_{s'_t \in \{s_t\}^\Phi} \tag{D.3}$$

which is precisely the first condition in Definition 6.5.2,

$$R(\Phi^\pi(s_t), a_t) = R(s_t, a_t), \tag{D.4}$$

for all $s_t \in \mathcal{S}$ and $a_t \in \mathcal{A}$.

Analogously, we have that,

$$\begin{aligned} \overset{\pi}{\Pr}(\Phi^\pi(s_{t+1}) \mid \Phi^\pi(s_t), a_t) &= \overset{\pi}{\Pr}(\Phi^\pi(s_{t+1}) \mid \text{do}(\Phi^\pi(s_t)), a_t) \\ &= \Pr(\Phi^\pi(s_{t+1}) \mid \Phi^\pi(s_t), a_t) \end{aligned} \tag{D.5}$$

where the second equality reflects that the above must hold independently of π. Hence, we have that for all $s_t, s_{t+1} \in \mathcal{S}$ and $s'_t \in \{s_t\}^\Phi$,

$$\Pr(\Phi^\pi(s_{t+1}) \mid \Phi^\pi(s_t), a_t) = \Pr(\Phi^\pi(s_{t+1}) \mid \Phi^\pi(s'_t), a_t), \tag{D.6}$$

which means that, for all $s_t, s_{t+1} \in \mathcal{S}$ and $s_t \in \mathcal{A}$,

$$\begin{aligned} \Pr(\Phi^\pi(s_{t+1}) \mid \Phi^\pi(s_t), a_t) &= \Pr(\Phi^\pi(s_{t+1}) \mid s_t, a_t) \\ &= \sum_{s'_{t+1} \in \{s_{t+1}\}^{\Phi^\pi}} T(s'_{t+1} \mid s_t, a_t), \end{aligned} \tag{D.7}$$

which is the second condition in Definition 6.5.2.

Equations (D.4) and (D.7) reveal that if the assumption is true (i.e., Φ^π is not confounded by the policy), then Φ^π is not just π-Markov but actually strictly Markov (Definition 6.5.2). However, we know that $\Phi^*(s_t)$ is the minimal state representation, which contradicts the above statement, since, according to Definition 6.5.3, there is no proper subset of $\bar{\Theta}^*$, for all $s_t \in \mathcal{S}$, such that the representation remains Markov. Hence, $\bar{\Theta}^\pi \subset \bar{\Theta}^*$ implies policy

confounding. □

Proposition 6.6.2. *Let* $\{\bar{\Theta}^*\}_{\cup\Phi^*}$ *be the union of variables in all possible minimal state representations. There exist cases where, for some* π*, there is a* π*-minimal state representation* $\Phi^{\pi*} : \mathcal{S}^\pi \to \bar{\mathcal{S}}^{\pi*}$ *with* $\bar{\mathcal{S}}^{\pi*} = \times_i \mathrm{dom}(\bar{\Theta}^{\pi*i})$ *such that* $\bar{\Theta}^{\pi*} \setminus \{\bar{\Theta}^*\}_{\cup\Phi^*} \neq \emptyset$.

Proof (sketch). Consider a deterministic MDP with a deterministic policy. Imagine there exists a variable X that is perfectly correlated with the episode's timestep t, but that is generally irrelevant to the task. The variable X would constitute in itself a valid π-Markov state representation since it can be used to determine transitions and rewards so long as a deterministic policy is followed. At the same time, X would not enter the minimal Markov state representation because it is useless under stochastic policies. Example D.2.1 below illustrates this situation. □

D.2 Example: Watch the Time

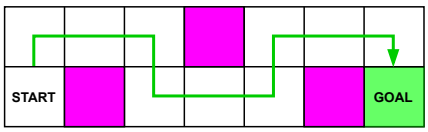


Figure D.1: An illustration of the watch-the-time environment.

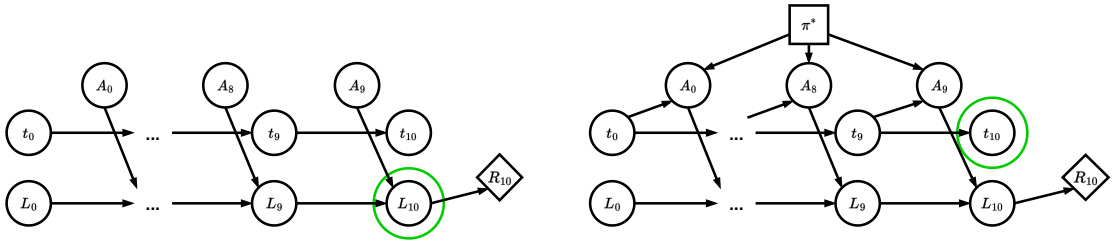


Figure D.2: Two DBNs representing the dynamics of the watch-the-time environment, when actions are sampled at random (left), and when they are determined by the optimal policy (right).

Example D.2.1. (Watch the Time) This example is inspired by the empirical results of Song et al. (2020). Figure D.1 shows a grid world environment., The agent must go from the start cell to the goal cell. The agent must avoid the pink cells; stepping on those yields a -0.1 penalty. There is a is $+1$ reward for reaching the goal. The agent can observe its own location within the maze L and the current timestep t. The two diagrams in

Figure D.2 are DBNs describing the environment dynamics. When actions are considered exogenous random variables (left diagram), the only way to estimate the reward at $t = 10$ is by looking at the agent's location L_{10}. In contrast, when actions are determined by the policy (right diagram), the time variable becomes a proxy for the agent's location. This is because the start location and the sequence of actions are fixed. This implies that t is a perfectly valid π-Markov state representation under π^*. Moreover, as shown by the DBN on the right, the optimal policy may simply rely on t to determine the optimal action.

D.3 Further Related Work

Early evidence of policy confounding Although to the best of our knowledge, we are the first to bring forward and describe mathematically the idea of policy confounding, a few prior works have reported evidence of particular forms of policy confounding. In their review of the Arcade Learning Environment (ALE; Bellemare et al., 2013), Machado et al. (2018) explain that because the games are fully deterministic (i.e., initial states are fixed and transitions are deterministic), open-loop policies that memorize good action sequences can achieve high scores in ALE. Clearly, this can only occur if the policies themselves are also deterministic. In such cases, policies, acting as confounders, induce a spurious correlation between the past action sequences and the environment states. Similarly, Song et al. (2020) showed, by means of saliency maps, how agents may learn to use irrelevant features of the environment that happen to be correlated with the agent's progress, such as background clouds or the game timer, as clues for outputting optimal actions. In this case, the policy is again a confounder for all these, a priori irrelevant, features. Zhang et al. (2018b) provide empirical results showing how large neural networks may overfit their training environments and, even when trained on a collection of procedurally generated environments, memorize the optimal action for each observation. Zhang et al. (2018a) shows how, when trained on a small subset of trajectories, agents fail to generalize to a set of test trajectories generated by the same simulator. Lan et al. (2023) report evidence of well-trained agents failing to perform well on Mujoco environments when starting from trajectories (states) that are out of the distribution induced by the agent's policy. We conceive this as a simple form of policy confounding. Since the Mujoco environments are also deterministic, agents following a fixed policy can memorize the best actions to take for each state instantiation, potentially relying on superfluous features. Hence, they can overfit to unnatural postures that would not occur under different policies. Finally, Nikishin et al. (2022) describe a phenomenon named 'primacy bias', which prevents agents trained on

poor trajectories from further improving their policies. The authors show that this issue is particularly relevant when training relies heavily on early data coming from a fixed random policy. We hypothesize that one of the causes for this is also policy confounding. The random policy may induce spurious correlations that lead to the formation of rigid state representations that are hard to recover from.

Generalization Generalization is a hot topic in machine learning. The promise of a model performing well in contexts other than those encountered during training is undoubtedly appealing. In the realm of reinforcement learning, the majority of research focuses on generalization to environments that, despite sharing a similar structure, differ somewhat from the training environment (Kirk et al., 2023). These differences range from small variations in the transition dynamics (e.g., sim-to-real transfer; Higgins et al., 2017; Tobin et al., 2017; Peng et al., 2018; Zhao et al., 2020), changes in the observations (i.e., modifying irrelevant information, such as noise: Mandlekar et al., 2017; Ornia et al., 2022, or background variables: Zhang et al., 2020; Stone et al., 2021), to alterations in the reward function, resulting in different goals or tasks (Taylor and Stone, 2009; Lazaric, 2012; Muller-Brockhausen et al., 2021). Instead, we focus on the problem of OOT generalization. Keeping the environment unchanged, we aim to ensure that agents perform effectively when confronted with situations that differ from those encountered along their usual trajectories. Note that, in our experiments agents are evaluated in altered environments with different dynamics than those seen during training. These alterations are only intended to force the agent to take different trajectories. Importantly, the trajectories we force the agent to take are possible in the original environment.

State abstraction State abstraction is concerned with removing from the representation all that state information that is irrelevant to the task. In contrast, we are worried about learning representations containing too little information, which can lead to state aliasing. Nonetheless, as argued by McCallum (1995b), state abstraction and state aliasing are two sides of the same coin. That is why we borrowed the mathematical frameworks of state abstraction to describe the phenomenon of policy confounding. Li et al. (2006) provide a taxonomy of the types of state abstraction and how they relate to one another. Givan et al. (2003) introduce the concept of bisimulation, which is equivalent to our definition of Markov state representation (Definition 6.5.2) but for states instead of states. Ferns et al. (2006) proposes a method for measuring the similarity between two states. Castro (2020) notes that this metric is prohibitively expensive and suggests using a relaxed version

that computes state similarity relative to a given policy. This is similar to our notion of π-Markov state representation (Definition 6.5.5). While the end goal of this metric is to group together states that are similar under a given policy, here we argue that this may lead to poor OOT generalization.

D.4 Dynamic Bayesian Networks

Figures D.3 and D.4 show the DBNs representing the dynamics of the Key2Door and Diversion environments respectively.

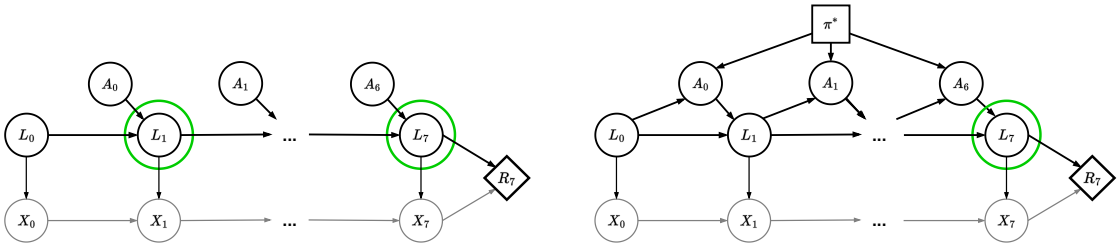


Figure D.3: Two DBNs representing the dynamics of the Key2Door environment, when actions are sampled at random (left), and when they are determined by the optimal policy (right). The nodes labeled as L represent the agent's location, while the nodes labeled as X represent whether or not the key has been collected. The agent can only see L. Hence, when actions that are sampled are random (left), the agent must remember its past locations to determine the reward R_7. Note that only L_1 and L_7 are highlighted in the left DBN. However, other variables in $\langle L_2, ..., L_6 \rangle$ might be needed, depending on when the key is collected. In contrast, when following the optimal policy, only L_7 is needed. In this second case, knowing the location is sufficient to determine whether the key has been collected.

D.5 Experimental Results

D.5.1 Learned state representations

The results reported in Section 3.4 show that the OOT generalization problem exists. However, some may still wonder if the underlying reason is truly policy confounding. To confirm this, we compare the outputs of the policy at every state in the Frozen T-Maze when being fed the same states (observation stack) but two different signals. That is, we permute the variable containing the signal (X in the diagram of Figure 6.2) and leave the

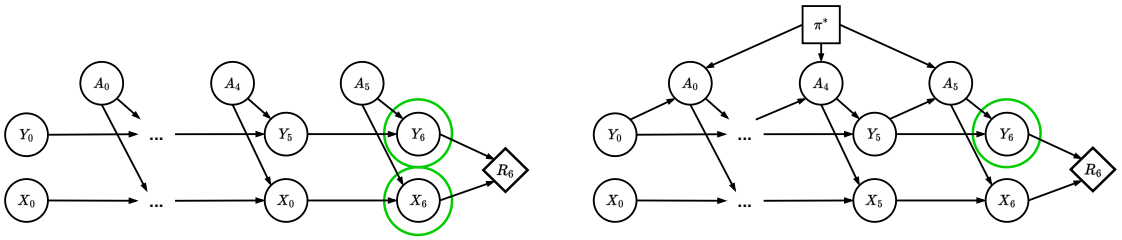


Figure D.4: Two DBNs representing the dynamics of the Diversion environment, when actions are sampled at random (left), and when they are determined by the optimal policy (right). The nodes labeled as X indicate the row where the agent is located; the nodes labeled as Y indicate the column. We see that when actions are sampled at random, both X_6 and Y_6 are necessary to determine r_6. However, when actions are determined by the optimal policy, Y_6 is sufficient, as the agent always stays at the top row.

rest of the variables in the observation stack unchanged. We then feed the two versions to the policy network and measure the KL divergence between the two output probabilities. This metric is a proxy for how much the agent attends to the signal in every state. The heatmaps in Figure D.5 show the KL divergences at various points during training (0, 10K, 30K, and 100K timesteps) when the true signal is 'green' and we replace it with 'purple'. We omit the two goal states since no actions are taken there. We see that initially (top left heatmap), the signal has very little influence on the policy (note the scale of the colormap is 10×-6), after 10K steps, the agent learns that the signal is very important when at the top right state (top right heatmap). After this, we start seeing how the influence of the signal at the top right state becomes less strong (bottom left heatmap) until it eventually disappears (bottom right heatmap). In contrast, the influence of the signal at the initial state becomes more and more important, indicating that after taking the first action, the agent ignores the signal and only attends to its own location. The results for the alternative case, purple signal being replaced by green signal, are shown in Figure D.6.

D.5.2 Buffer size and exploration/domain randomization

Figures D.7 and D.8 report the results of the experiments described in Section 3.4 (paragraphs 2 and 3) for Key2Door and Diversion. We see how the buffer size also affects the performance of DQN in the two environments (left plots). We also see that exploration/domain randomization does improve OOT generalization in Diversion but not in Key2Door.

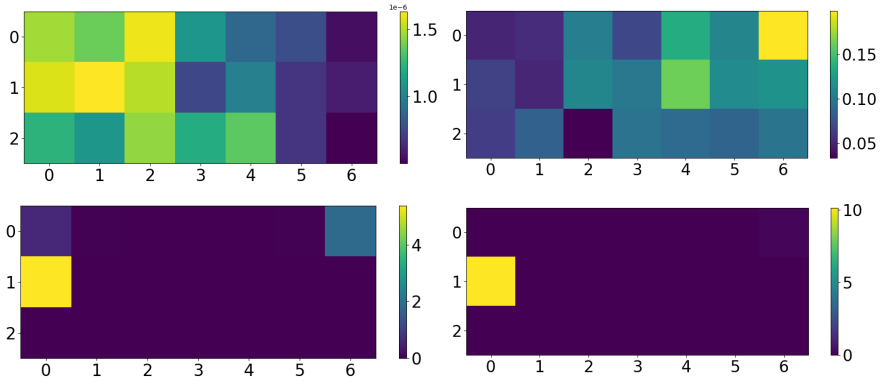


Figure D.5: A visualization of the learned history representations. The heatmaps show the KL divergence between the action probabilities when feeding the policy network a stack of the past 10 observations and when feeding the same stack but with the value of the signal being switched from green to purple, after 0 (top left), 10K (top right), 30K (bottom left), and 100K (bottom right) timesteps of training.

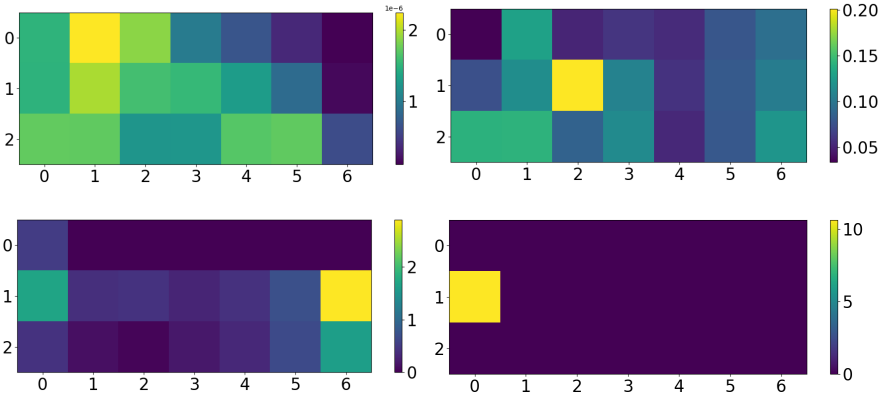


Figure D.6: A visualization of the learned history representations. The heatmaps show the KL divergence between the action probabilities when feeding the policy network a stack of the past 10 observations and when feeding the same stack but with the value of the signal being switched from purple to green, after 0 (top left), 10K (top right), 30K (bottom left), and 100K (bottom right) timesteps of training.

D.6 Further Experimental Details

We ran our experiments on an Intel i7-8650U CPU with 8 cores. Agents were trained with Stable Baselines3 (Raffin et al., 2021). Most hyperparameters were set to their default values except for the ones reported in Tables D.1 (PPO) and D.2 (DQN), which improved the performance over the default values.

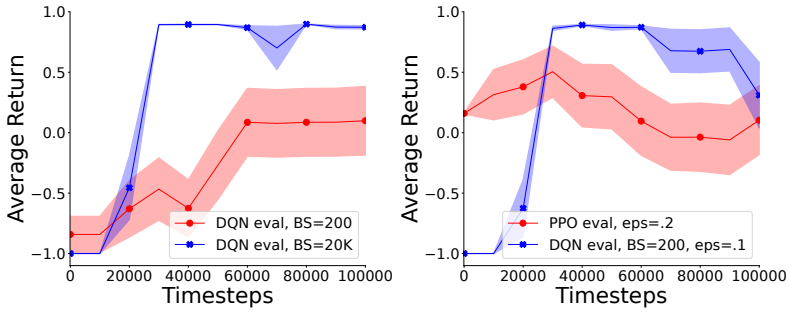


Figure D.7: Key2Door. Left: DQN small vs. large buffer sizes. Right: PPO and DQN when adding stochasticity.

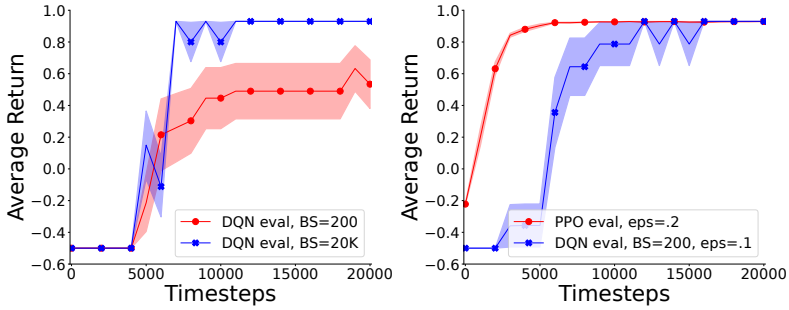


Figure D.8: Diversion. Left: DQN small vs. large buffer sizes. Right: PPO and DQN when adding stochasticity.

Table D.1: PPO hyperparameters.

Rollout steps	128
Batch size	32
Learning rate	2.5e-4
Number epoch	3
Entropy coefficient	1.0e-2
Clip range	0.1
Value coefficient	1
Number Neurons 1st layer	128
Number Neurons 2nd layer	128

Table D.2: DQN hyperparameters.

Buffer size	1.0e5
Learning starts	1.0e3
Learning rate	2.5e-4
Batch size	256
Initial exploration bonus	1.0
Final exploration bonus	0.0
Exploration fraction	0.2
Train frequency	5
Number Neurons 1st layer	128
Number Neurons 2nd layer	128

Bibliography

Abel, D. (2020). *A theory of abstraction in reinforcement learning*. PhD thesis, Brown University.

Abel, D., Hershkowitz, D. E., and Littman, M. L. (2016). Near optimal behavior via approximate state abstraction. In *Proc. of the 33rd International Conference on Machine learning*, pages 2915–2923.

Acid, S. and De Campos, L. M. (1996). An algorithm for finding minimum d-separating sets in belief networks. In *Proceedings of the Twelfth International Conference on Uncertainty in Artificial Intelligence*.

Agarwal, R., Machado, M. C., Castro, P. S., and Bellemare, M. G. (2021). Contrastive behavioral similarity embeddings for generalization in reinforcement learning. In *International Conference on Learning Representations*.

Altman, E. (1999). *Constrained Markov decision processes*, volume 7. CRC press.

Andre, D. and Russell, S. J. (2002). State abstraction for programmable reinforcement learning agents. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pages 119–125.

Arjovsky, M. (2021). Out of distribution generalization in machine learning. *arXiv preprint arXiv:2103.02667*.

Arjovsky, M., Bottou, L., Gulrajani, I., and Lopez-Paz, D. (2019). Invariant risk minimization. *arXiv preprint arXiv:1907.02893*.

Bakker, B. (2001). Reinforcement learning with long short-term memory. *Advances in neural information processing systems*, 14.

Becker, R., Zilberstein, S., Lesser, V., and Goldman, C. V. (2003). Transition-independent decentralized Markov decision processes. In *Proceedings of the Second Joint International Conference on Autonomous Agents and Multiagent Systems*, pages 41–48.

Beery, S., Van Horn, G., and Perona, P. (2018). Recognition in terra incognita. In *Proceedings of the European conference on computer vision (ECCV)*, pages 456–473.

Bellemare, M. G., Candido, S., Castro, P. S., Gong, J., Machado, M. C., Moitra, S., Ponda, S. S., and Wang, Z. (2020). Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588(7836):77–82.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The Arcade Learning Environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press.

Bertsekas, D. P. (2005). *Dynamic Programming and Optimal Control*, volume I. Athena Scientific, 3rd edition.

Beukman, M., Jarvis, D., Klein, R., James, S., and Rosman, B. (2023). Dynamics generalisation in reinforcement learning via adaptive context-aware policies. *arXiv preprint arXiv:2310.16686*.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Botvinick, M., Ritter, S., Wang, J. X., Kurth-Nelson, Z., Blundell, C., and Hassabis, D. (2019). Reinforcement learning, fast and slow. *Trends in cognitive sciences*, 23(5):408–422.

Boutilier, C., Dean, T., and Hanks, S. (1999). Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94.

Boutilier, C., Dearden, R., and Goldszmidt, M. (2000). Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1-2):49–107.

Boutilier, C., Dearden, R., Goldszmidt, M., et al. (1995). Exploiting structure in policy construction. In *IJCAI*, volume 14, pages 1104–1113.

Boutilier, C., Friedman, N., Goldszmidt, M., and Koller, D. (1996). Context-specific independence in bayesian networks. In *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*, pages 115–123.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym.

Castellini, J., Devlin, S., Oliehoek, F. A., and Savani, R. (2020). Difference rewards policy gradients. *arXiv preprint arXiv:2012.11258*.

Castro, P. S. (2020). Scalable methods for computing state similarity in deterministic markov decision processes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 10069–10076.

Chitnis, R. and Lozano-Pérez, T. (2020). Learning compact models for planning with exogenous processes. In *Proceedings of the Conference on Robot Learning*.

Cho, K., van Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing*.

Claes, D., Oliehoek, F., Baier, H., Tuyls, K., et al. (2017). Decentralised online planning for multi-robot warehouse commissioning. In *Proceedings of the 16th international conference on autonomous agents and multiagent systems*, pages 492–500.

Claus, C. and Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. In *Proc. of the National Conference on Artificial Intelligence*, pages 746–752.

Congeduti, E., Mey, A., and Oliehoek, F. A. (2021). Loss bounds for approximate influence-based abstraction. In *Proceedings of the Twentieth International Conference on Autonomous Agents and MultiAgent Systems*.

Daly, R., Shen, Q., and Aitken, S. (2011). Learning bayesian networks: approaches and issues. *The knowledge engineering review*, 26(2):99–157.

de Witt, C. S., Gupta, T., Makoviichuk, D., Makoviychuk, V., Torr, P. H., Sun, M., and Whiteson, S. (2020). Is independent learning all you need in the starcraft multi-agent challenge? *arXiv preprint arXiv:2011.09533*.

Dean, T. and Givan, R. (1997). Model minimization in Markov decision processes. In *Proc. of the National Conference on Artificial Intelligence*, pages 106–111.

Dean, T. and Kanazawa, K. (1989). A model for reasoning about persistence and causation. *Computational intelligence*, 5(2):142–150.

Dearden, R. and Boutilier, C. (1997). Abstraction and approximate decision-theoretic planning. *Artificial Intelligence*, 89(1-2):219–283.

Degrave, J., Felici, F., Buchli, J., Neunert, M., Tracey, B., Carpanese, F., Ewalds, T., Hafner, R., Abdolmaleki, A., de Las Casas, D., et al. (2022). Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419.

Depeweg, S., Hernandez-Lobato, J.-M., Doshi-Velez, F., and Udluft, S. (2018). Decomposition of uncertainty in bayesian deep learning for efficient and risk-sensitive learning. In *International Conference on Machine Learning*. PMLR.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of AI Research*, 13:227–303.

Dulac-Arnold, G., Mankowitz, D., and Hester, T. (2019). Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901*.

Eysenbach, B. and Levine, S. (2022). Maximum entropy RL (provably) solves some robust RL problems. In *International Conference on Learning Representations*.

Farahmand, A.-m. (2011). Action-gap phenomenon in reinforcement learning. *Advances in Neural Information Processing Systems*, 24.

Feng, F., Huang, B., Zhang, K., and Magliacane, S. (2022). Factored adaptation for non-stationary reinforcement learning. In *Advances in Neural Information Processing Systems*.

Ferns, N., Castro, P. S., Precup, D., and Panangaden, P. (2006). Methods for computing state similarity in markov decision processes. In *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence*, UAI'06, page 174–181.

Foerster, J., Chen, R. Y., Al-Shedivat, M., Whiteson, S., Abbeel, P., and Mordatch, I. (2018a). Learning with opponent-learning awareness. In *Proceedings of the Seventeenth International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, pages 122–130, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. (2018b). Counterfactual multi-agent policy gradients. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*.

Freeman, C. D., Frey, E., Raichuk, A., Girgin, S., Mordatch, I., and Bachem, O. (2021). Brax–a differentiable physics engine for large scale rigid body simulation. *arXiv preprint arXiv:2106.13281*.

French, R. M. (1999). Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135.

Ganesh, S., Vadori, N., Xu, M., Zheng, H., Reddy, P., and Veloso, M. M. (2019). Reinforcement learning for market making in a multi-agent dealer market. *ArXiv*, abs/1911.05892.

García, J. and Fernández, F. (2015). A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480.

Geirhos, R., Jacobsen, J.-H., Michaelis, C., Zemel, R., Brendel, W., Bethge, M., and Wichmann, F. A. (2020). Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673.

Geroliminis, N. and Daganzo, C. F. (2008). Existence of urban-scale macroscopic fundamental diagrams: Some experimental findings. *Transportation Research Part B: Methodological*.

Givan, R., Dean, T., and Greig, M. (2003). Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*, 14(1–2):163–223.

Guestrin, C., Koller, D., Parr, R., and Venkataraman, S. (2003). Efficient solution algorithms for factored MDPs. *Journal of AI Research*, 19:399–468.

Gupta, A., Badr, Y., Negahban, A., and Qiu, R. G. (2021). Energy-efficient heating control for smart buildings with deep reinforcement learning. *Journal of Building Engineering*, 34:101739.

Ha, D. and Schmidhuber, J. (2018). Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems*, volume 31, pages 2450–2462.

Hansen, E. A., Bernstein, D. S., and Zilberstein, S. (2004). Dynamic programming for partially observable stochastic games. In *Proc. of the National Conference on Artificial Intelligence*, pages 709–715.

Hausknecht, M. and Stone, P. (2015). Deep recurrent Q-learning for partially observable MDPs. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*.

He, J., Suau, M., Baier, H., Kaisers, M., and Oliehoek, F. A. (2022). Online planning in pomdps with self-improving simulators. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*. Main Track.

He, J., Suau, M., and Oliehoek, F. (2020). Influence-augmented online planning for complex environments. In *Advances in Neural Information Processing Systems*.

Hengst, B. et al. (2002). Discovering hierarchy in reinforcement learning with hexq. In *ICML*, volume 19, pages 243–250. Citeseer.

Hernandez-Leal, P., Kaisers, M., Baarslag, T., and de Cote, E. M. (2017). A survey of learning in multiagent environments: Dealing with non-stationarity. *arXiv preprint arXiv:1707.09183*.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2018). Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*.

Higgins, I., Pal, A., Rusu, A., Matthey, L., Burgess, C., Pritzel, A., Botvinick, M., Blundell, C., and Lerchner, A. (2017). Darla: Improving zero-shot transfer in reinforcement learning. In *International Conference on Machine Learning*, pages 1480–1490. PMLR.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hotelling, H. (1992). Relations between two sets of variates. In *Breakthroughs in statistics*, pages 162–190. Springer.

Huang, A. J. and Agarwal, S. (2020). Physics informed deep learning for traffic state estimation. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE.

Igl, M., Zintgraf, L., Le, T. A., Wood, F., and Whiteson, S. (2018). Deep variational reinforcement learning for POMDPs. In *Proceedings of the 35th International Conference on Machine Learning*, pages 2117–2126.

Iqbal, S. and Sha, F. (2019). Actor-attention-critic for multi-agent reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, pages 2961–2970.

Jaakkola, T., Singh, S. P., and Jordan, M. I. (1995). Reinforcement learning algorithm for partially observable markov decision problems. In *Advances in neural information processing systems*, pages 345–352.

Jaderberg, M., Czarnecki, W. M., Dunning, I., Marris, L., Lever, G., Castaneda, A. G., Beattie, C., Rabinowitz, N. C., Morcos, A. S., Ruderman, A., et al. (2019). Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865.

Jensen, F. V. and Nielsen, T. D. (2007). *Bayesian Networks and Decision Graphs*. Information Science and Statistics. Springer-Verlag New York, 2 edition.

Jiang, Y., Kolter, J. Z., and Raileanu, R. (2022). Uncertainty-driven exploration for generalization in reinforcement learning. In *Deep Reinforcement Learning Workshop NeurIPS 2022*.

Jonsson, A. and Barto, A. (2007). Active learning of dynamic bayesian networks in markov decision processes. In *Abstraction, Reformulation, and Approximation: 7th International Symposium, SARA 2007, Whistler, Canada, July 18-21, 2007. Proceedings 7*, pages 273–284. Springer.

Jonsson, A. and Barto, A. G. (2005). A causal approach to hierarchical decomposition of factored MDPs. In *Proc. of the Twenty-Second International Conference on Machine learning*, pages 401–408.

Juliani, A., Berges, V.-P., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., et al. (2018). Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*.

Kaelbling, L. P., Littman, M., and Moore, A. (1996). Reinforcement learning: A survey. *Journal of AI Research*, 4:237–285.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134.

Kakade, S. M. (2003). *On the sample complexity of reinforcement learning*. PhD thesis, University College London.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems.

Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., and Yang, L. (2021). Physics-informed machine learning. *Nature Reviews Physics*.

Kearns, M. and Koller, D. (1999). Efficient reinforcement learning in factored MDPs. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 740–747.

Kearns, M. and Singh, S. (2002). Near-optimal reinforcement learning in polynomial time. *Machine learning*, 49(2):209–232.

Kirk, R., Zhang, A., Grefenstette, E., and Rocktäschel, T. (2023). A survey of zero-shot generalisation in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 76:201–264.

Knoop, V. L., Van Lint, H., and Hoogendoorn, S. P. (2015). Traffic dynamics: Its impact on the macroscopic fundamental diagram. *Physica A: Statistical Mechanics and its Applications*.

Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

Konda, V. and Tsitsiklis, J. (1999). Actor-critic algorithms. *Advances in neural information processing systems*, 12.

Krueger, D., Caballero, E., Jacobsen, J.-H., Zhang, A., Binas, J., Zhang, D., Priol, R. L., and Courville, A. (2021). Out-of-distribution generalization via risk extrapolation (rex). In *Proceedings of the 38th International Conference on Machine Learning*.

Kumar, A., Zilberstein, S., and Toussaint, M. (2011). Scalable multiagent planning using probabilistic inference. In *Proc. of the International Joint Conference on Artificial Intelligence*, pages 2140–2146.

Lample, G. and Chaplot, D. S. (2017). Playing fps games with deep reinforcement learning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 2140–2146.

Lan, L.-C., Zhang, H., and Hsieh, C.-J. (2023). Can agents run relay race with strangers? generalization of RL to out-of-distribution trajectories. In *The Eleventh International Conference on Learning Representations*.

Langosco, L., Koch, J., Sharkey, L. D., Pfau, J., and Krueger, D. (2022). Goal misgeneralization in deep reinforcement learning. In *International Conference on Machine Learning*, pages 12004–12019. PMLR.

Laurent, G. J., Matignon, L., Fort-Piat, L., et al. (2011). The world of independent learners is not markovian. *International Journal of Knowledge-based and Intelligent Engineering Systems*, 15(1):55–64.

Lazaric, A. (2012). Transfer in reinforcement learning: a framework and a survey. *Reinforcement Learning: State-of-the-Art*, pages 143–173.

Li, G., Knoop, V. L., and Van Lint, H. (2021a). Multistep traffic forecasting by dynamic graph convolution: Interpretations of real-time spatial correlations. *Transportation Research Part C: Emerging Technologies*, 128:103185.

Li, G., Knoop, V. L., and van Lint, H. (2022). Estimate the limit of predictability in short-term traffic forecasting: An entropy-based approach. *Transportation Research Part C: Emerging Technologies*.

Li, L., Walsh, T. J., and Littman, M. L. (2006). Towards a unified theory of state abstraction for MDPs. In *International Symposium on Artificial Intelligence and Mathematics (ISAIM 2006)*.

Li, W., Wang, X., Jin, B., Sheng, J., and Zha, H. (2021b). Dealing with non-stationarity in marl via trust-region decomposition. In *International Conference on Learning Representations*.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In *International Conference on Learning Representations*.

Lin, L.-J. and Mitchell, T. M. (1993). Reinforcement learning with hidden states. *From animals to animats*, 2:271–280.

Lippe, P., Magliacane, S., Löwe, S., Asano, Y. M., Cohen, T., and Gavves, E. (2023). Biscuit: Causal representation learning from binary interactions. *arXiv preprint arXiv:2306.09643*.

Littman, M. L. (1994). Memoryless policies: Theoretical limitations and practical results. In *Proc. of the Third International Conference on Simulation of Adaptive Behavior : From Animals to Animats 3*, pages 238–245.

Lopez, P. A., Behrisch, M., Bieker-Walz, L., Erdmann, J., Flötteröd, Y.-P., Hilbrich, R., Lücken, L., Rummel, J., Wagner, P., and Wießner, E. (2018). Microscopic traffic simulation using SUMO. In *The 21st International Conference on Intelligent Transportation Systems*.

Lopez-Paz, D., Nishihara, R., Chintala, S., Scholkopf, B., and Bottou, L. (2017). Discovering causal signals in images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6979–6987.

Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., and Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. *Neural Information Processing Systems (NIPS)*.

Lu, C., Schölkopf, B., and Hernández-Lobato, J. M. (2018). Deconfounding reinforcement learning in observational settings. *arXiv preprint arXiv:1812.10576*.

Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Lyu, X., Xiao, Y., Daley, B., and Amato, C. (2021). Contrasting centralized and decentralized critics in multi-agent reinforcement learning. *Proceedings of the Twentieth International Conference on Autonomous Agents and MultiAgent Systems*.

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., and Bowling, M. (2018). Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562.

Mandlekar, A., Zhu, Y., Garg, A., Fei-Fei, L., and Savarese, S. (2017). Adversarially robust policy learning: Active construction of physically-plausible perturbations. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3932–3939. IEEE.

Marbach, P. and Tsitsiklis, J. N. (2001). Simulation-based optimization of markov reward processes. *IEEE Transactions on Automatic Control*, 46(2):191–209.

McCallum, A. K. (1995a). Instance-based utile distinctions for reinforcement learning with hidden state. In *Machine Learning Proceedings 1995*, pages 387–395. Elsevier.

McCallum, A. K. (1995b). *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.

Moerland, T. M., Broekens, J., Plaat, A., Jonker, C. M., et al. (2023). Model-based reinforcement learning: A survey. *Foundations and Trends in Machine Learning*, 16(1):1–118.

Mooij, J. M., Magliacane, S., and Claassen, T. (2020). Joint causal inference from multiple contexts. *The Journal of Machine Learning Research*, 21(1):3919–4026.

Mooij, J. M., Peters, J., Janzing, D., Zscheischler, J., and Schölkopf, B. (2016). Distinguishing cause from effect using observational data: methods and benchmarks. *The Journal of Machine Learning Research*, 17(1):1103–1204.

Mott, A., Zoran, D., Chrzanowski, M., Wierstra, D., and Rezende, D. J. (2019). Towards interpretable reinforcement learning using attention augmented agents. In *Advances in Neural Information Processing Systems*, pages 12329–12338.

Muller-Brockhausen, M., Preuss, M., and Plaat, A. (2021). Procedural content generation: Better benchmarks for transfer reinforcement learning. In *2021 IEEE Conference on games (CoG)*, pages 01–08. IEEE.

Murphy, K. P. (2002). *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, UC Berkeley, Computer Science Division.

Murphy, K. P. (2022). *Probabilistic machine learning: an introduction*. MIT press.

Nair, R., Varakantham, P., Tambe, M., and Yokoo, M. (2005). Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In *Proc. of the National Conference on Artificial Intelligence*, pages 133–139.

Ng, A. Y. and Jordan, M. (2000). PEGASUS: A policy search method for large MDPs and POMDPs. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 406–415. Morgan Kaufmann Publishers Inc.

Nikishin, E., Schwarzer, M., D'Oro, P., Bacon, P.-L., and Courville, A. (2022). The primacy bias in deep reinforcement learning. In *International Conference on Machine Learning*, pages 16828–16847. PMLR.

van der Pol, E. and Oliehoek, F. A. (2016). Coordinated Deep Reinforcement Learners for traffic light control. NIPS Workshop on Learning, Inference and Control of Multi-Agent Systems.

Oh, J., Chockalingam, V., Singh, S., and Lee, H. (2016a). Control of memory, active perception, and action in minecraft. In *Proc. of the 33rd International Conference on Machine learning*, pages 2790–2799.

Oh, J., Chockalingam, V., Singh, S., and Lee, H. (2016b). Control of memory, active perception, and action in minecraft. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning*.

Oliehoek, F., Witwicki, S., and Kaelbling, L. (2021). A sufficient statistic for influence in structured multiagent environments. *Journal of Artificial Intelligence Research*, 70:789–870.

Oliehoek, F. A. and Amato, C. (2016). *A Concise Introduction to Decentralized POMDPs*. Springer Briefs in Intelligent Systems. Springer.

Oliehoek, F. A., Witwicki, S. J., and Kaelbling, L. P. (2012). Influence-based abstraction for multiagent systems. In *AAAI12*.

Ornia, D. J., Romao, L., Hammond, L., Mazo Jr, M., and Abate, A. (2022). Observational robustness and invariances in reinforcement learning via lexicographic objectives. *arXiv preprint arXiv:2209.15320*.

Papageorgiou, M., Diakaki, C., Dinopoulou, V., Kotsialos, A., and Wang, Y. (2003). Review of road traffic control strategies. *Proceedings of the IEEE*.

Pearl, J. (1988). *Probabilistic Reasoning In Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

Pearl, J. (2000). *Causality: Models, Reasoning, and Inference*. Cambridge University Press.

Pearl, J., Glymour, M., and Jewell, N. P. (2016). Causal inference in statistics: A primer. 2016. *Internet resource*.

Pearl, J. and Mackenzie, D. (2018). *The book of why: the new science of cause and effect*. Basic books.

Peng, X. B., Andrychowicz, M., Zaremba, W., and Abbeel, P. (2018). Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3803–3810. IEEE.

Peters, J., Bühlmann, P., and Meinshausen, N. (2016). Causal inference by using invariant prediction: identification and confidence intervals. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, pages 947–1012.

Peters, J., Janzing, D., and Schölkopf, B. (2017). *Elements of causal inference: foundations and learning algorithms*. The MIT Press.

Pineau, J., Gordon, G., and Thrun, S. (2003). Point-based value iteration: An anytime algorithm for POMDPs. In *Proc. of the International Joint Conference on Artificial Intelligence*, pages 1025–1032.

Puterman, M. L. (1994). *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. Wiley.

Puterman, M. L. (2014). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Qadri, S. S. S. M., Gökçe, M. A., and Öner, E. (2020). State-of-art review of traffic signal control methods: challenges and opportunities. *European transport research review*.

Quionero-Candela, J., Sugiyama, M., Schwaighofer, A., and Lawrence, N. D. (2009). *Dataset shift in machine learning*. The MIT Press.

Rabinowitz, N., Perbet, F., Song, F., Zhang, C., Eslami, S. A., and Botvinick, M. (2018). Machine theory of mind. In *International conference on machine learning*, pages 4218–4227. PMLR.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8.

Raileanu, R., Denton, E., Szlam, A., and Fergus, R. (2018). Modeling others using oneself in multi-agent reinforcement learning. In *International conference on machine learning*, pages 4257–4266. PMLR.

Rummery, G. A. and Niranjan, M. (1994). *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK.

Saengkyongam, S., Thams, N., Peters, J., and Pfister, N. (2023). Invariant policy learning: A causal perspective. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2016a). Prioritized experience replay. In *4th International Conference on Learning Representations*.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2016b). Prioritized experience replay. In *International Conference on Learning Representations*.

Scheines, R. (n.d.). D-separation. `https://www.andrew.cmu.edu/user/scheines/tutor/d-sep.html`. Accessed: 2023-04-21.

Schmidhuber, J. (1991). Reinforcement learning in Markovian and non-Markovian environments. In *Advances in Neural Information Processing Systems*, pages 500–506.

Schölkopf, B., Locatello, F., Bauer, S., Ke, N. R., Kalchbrenner, N., Goyal, A., and Bengio, Y. (2021). Toward causal representation learning. *Proceedings of the IEEE*, 109(5):612–634.

Schölkopf, B. and von Kügelgen, J. (2022). From statistical to causal learning. *arXiv preprint arXiv:2204.00607*.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. (2020). Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Seitzer, M., Schölkopf, B., and Martius, G. (2021). Causal influence detection for improving efficiency in reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 34.

Shachter, R. D. (1988). Probabilistic inference and influence diagrams. *Operations Research*, 36:589–605.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.

Silver, D. and Veness, J. (2010). Monte-carlo planning in large pomdps. In *Advances in neural information processing systems*, pages 2164–2172.

Singh, S., Jaakkola, T., and Jordan, M. (1994a). Reinforcement learning with soft state aggregation. *Advances in neural information processing systems*, 7.

Singh, S. P., Jaakkola, T., and Jordan, M. I. (1994b). Learning without state-estimation in partially observable Markovian decision processes. In *Proc. of the International Conference on Machine Learning*, pages 284–292.

Song, X., Jiang, Y., Tu, S., Du, Y., and Neyshabur, B. (2020). Observational overfitting in reinforcement learning. In *International Conference on Learning Representations*.

Sontakke, S. A., Mehrjou, A., Itti, L., and Schölkopf, B. (2021). Causal curiosity: Rl agents discovering self-supervised experiments for causal representation learning. In *International conference on machine learning*, pages 9848–9858. PMLR.

Sorokin, I., Seleznev, A., Pavlov, M., Fedorov, A., and Ignateva, A. (2015). Deep attention recurrent q-network. *arXiv preprint arXiv:1512.01693*.

Spooner, T., Vadori, N., and Ganesh, S. (2021). Factored policy gradients: Leveraging structure for efficient learning in MOMDPs. *Advances in Neural Information Processing Systems*, 34.

Stone, A., Ramirez, O., Konolige, K., and Jonschkowski, R. (2021). The distracting control suite–a challenging benchmark for reinforcement learning from pixels. *arXiv preprint arXiv:2101.02722*.

Suau, M., Agapitos, A., Lynch, D., Farrell, D., Zhou, M., and Milenovic, A. (2021). Offline contextual bandits for wireless network optimization. *arXiv preprint arXiv:2111.08587*.

Suau, M., He, J., Celikok, M. M., Spaan, M. T. J., and Oliehoek, F. A. (2022a). Distributed influence-augmented local simulators for parallel MARL in large networked systems. In *Advances in Neural Information Processing Systems*.

Suau, M., He, J., Congeduti, E., Starre, R. A., Czechowski, A., and Oliehoek, F. A. (2022b). Influence-aware memory architectures for deep reinforcement learning in POMDPs. *Neural Computing and Applications. Special Issue on Adaptive and Learning Agents*.

Suau, M., He, J., Spaan, M. T., and Oliehoek, F. A. (2022c). Speeding up deep reinforcement learning through influence-augmented local simulators. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, pages 1735–1737.

Suau, M., He, J., Spaan, M. T. J., and Oliehoek, F. (2022d). Influence-augmented local simulators: a scalable solution for fast deep RL in large networked systems. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 20604–20624. PMLR.

Suau, M., Spaan, M. T. J., and Oliehoek, F. A. (2023). Bad habits: Policy confounding and out-of-trajectory generalization in RL. In *Sixteenth European Workshop on Reinforcement Learning*.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. The MIT Press.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12.

Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337.

Tang, Y., Nguyen, D., and Ha, D. (2020). Neuroevolution of self-interpretable agents. *arXiv preprint arXiv:2003.08165*.

Taylor, M. E. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7).

Tian, J., Paz, A., and Pearl, J. (1998). *Finding minimal d-separators*. Tech. rep. R-254, University of California, Los Angeles.

Tishby, N. and Zaslavsky, N. (2015). Deep learning and the information bottleneck principle. In *2015 IEEE Information Theory Workshop (ITW)*, pages 1–5. IEEE.

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE.

van der Pol, E., Kipf, T., Oliehoek, F. A., and Welling, M. (2020). Plannable approximations to mdp homomorphisms: Equivariance under actions. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*.

Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*.

van Hasselt, H. P., Hessel, M., and Aslanides, J. (2019). When to use parametric models in reinforcement learning? In *Advances in Neural Information Processing Systems*, volume 32.

van Lint, H. and Djukic, T. (2012). Applications of kalman filtering in traffic management and control. In *New Directions in Informatics, Optimization, Logistics, and Production*. informs.

Van Lint, J., Hoogendoorn, S., and van Zuylen, H. J. (2005). Accurate freeway travel time prediction with state-space neural networks under missing data. *Transportation Research Part C: Emerging Technologies*.

Varakantham, P., Marecki, J., Yabu, Y., Tambe, M., and Yokoo, M. (2007). Letting loose a SPIDER on a network of POMDPs: Generating quality guaranteed policies. In *Proc. of the International Conference on Autonomous Agents and Multiagent Systems*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems 17*, pages 5998–6008.

Vinitsky, E., Kreidieh, A., Le Flem, L., Kheterpal, N., Jang, K., Wu, C., Wu, F., Liaw, R., Liang, E., and Bayen, A. M. (2018). Benchmarks for reinforcement learning in mixed-autonomy traffic. In *Conference on robot learning*, pages 399–409. PMLR.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. (2019). Grandmaster level in starcraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354.

Wang, J., Xu, W., Gu, Y., Song, W., and Green, T. (2021). Multi-agent reinforcement learning for active voltage control on power distribution networks. *Advances in Neural Information Processing Systems*, 34.

Wang, Y. and Papageorgiou, M. (2005). Real-time freeway traffic state estimation based on extended kalman filter: a general approach. *Transportation Research Part B: Methodological*.

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

Williams, C. K. and Rasmussen, C. E. (2006). *Gaussian processes for machine learning*. MIT press Cambridge, MA.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256.

Williams, R. J. and Peng, J. (1990). An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural computation*, 2(4):490–501.

Witwicki, S. and Durfee, E. (2010a). From policies to influences: A framework for nonlocal abstraction in transition-dependent Dec-POMDP agents. In *Proc. of the International Conference on Autonomous Agents and Multiagent Systems*, pages 1397–1398.

Witwicki, S. J. and Durfee, E. H. (2010b). Influence-based policy abstraction for weakly-coupled Dec-POMDPs. In *Proc. of the International Conference on Automated Planning and Scheduling*, pages 185–192.

Witwicki, S. J. and Durfee, E. H. (2011). Towards a unifying characterization for quantifying weak coupling in Dec-POMDPs. In *Proceedings of the Tenth International Conference on Autonomous Agents and Multiagent Systems*, pages 29–36.

Wu, C., Kreidieh, A., Parvate, K., Vinitsky, E., and Bayen, A. M. (2017). Flow: A modular learning framework for autonomy in traffic. *arXiv preprint arXiv:1710.05465*.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., and Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. In *Proc. of the 32nd International Conference on Machine learning*, pages 2048–2057.

Yu, C., Velu, A., Vinitsky, E., Wang, Y., Bayen, A., and Wu, Y. (2021). The surprising effectiveness of ppo in cooperative, multi-agent games. *arXiv preprint arXiv:2103.01955*.

Zhang, A., Ballas, N., and Pineau, J. (2018a). A dissection of overfitting and generalization in continuous reinforcement learning. *arXiv preprint arXiv:1806.07937*.

Zhang, A., Lyle, C., Sodhani, S., Filos, A., Kwiatkowska, M., Pineau, J., Gal, Y., and Precup, D. (2020). Invariant causal prediction for block mdps. In *International Conference on Machine Learning*, pages 11214–11224. PMLR.

Zhang, A., McAllister, R. T., Calandra, R., Gal, Y., and Levine, S. (2021). Learning invariant representations for reinforcement learning without reconstruction. In *International Conference on Learning Representations*.

Zhang, C., Abdallah, S., and Lesser, V. (2009). Integrating organizational control into multi-agent learning. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 757–764.

Zhang, C., Lesser, V. R., and Abdallah, S. (2010). Self-organization for coordinating decentralized reinforcement learning. In *AAMAS*, volume 10, pages 739–746.

Zhang, C., Vinyals, O., Munos, R., and Bengio, S. (2018b). A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893*.

Zhao, W., Queralta, J. P., and Westerlund, T. (2020). Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE symposium series on computational intelligence (SSCI)*, pages 737–744. IEEE.

Zhu, P., Li, X., and Poupart, P. (2017). On improving deep reinforcement learning for POMDPs. *ArXiv e-prints*, arXiv:1704.07978.

Zhu, Y., Wong, J., Mandlekar, A., and Martín-Martín, R. (2020). robosuite: A modular simulation framework and benchmark for robot learning. *arXiv preprint arXiv:2009.12293*.

Curriculum Vitae

12-05-1991	Born in Lugo, Spain

Education

2010 - 2014	Bachelor's Degree in Energy Engineering *Technical University of Madrid*
2016 - 2018	Master of Science in Mathematical Modelling and Computation *Technical University of Denmark*

Work Experience

2016	Solutions Assistant *NTT Data*
2017 - 2018	Business Intelligence Analyst *Unity Technologies*
2018 - 2023	Ph.D. Candidate *Delft University of Technology*
2020 - 2022	Lecturer, CSE2530, Computational Intelligence *Delft University of Technology*
2021	Ph.D. Research Intern *Huawei Ireland Research Center*
2022	Ph.D. Research Intern *J.P. Morgan AI Research*
2023 - Now	AI Research Scientist *Phaidra Inc.*

List of Publications

10. M. Suau, M. T. J Spaan, F. A. Oliehoek. Bad Habits: Policy Confounding and Out-of-Trajectory Generalization in RL. European Workshop on Reinforcement Learning, 2023.

9. M. Suau, J. He, M. M. Çelikok, M. T. J Spaan, F. A. Oliehoek. Distributed Influence-Augmented Local Simulators for Parallel MARL in Large Networked Systems. Neural Information Processing Systems, 2022.

8. M. Suau, J. He, M. T. J Spaan, F. A. Oliehoek. Influence-Augmented Local Simulators: A Scalable Solution for Fast Deep RL in Large Networked Systems. International Conference on Machine Learning, 2022.

7. J. He, M. Suau, H. Baier, M. Kaisers, F. A. Oliehoek. Online Planning in POMDPs with Self-Improving Simulators. International Joint Conferences on Artificial Intelligence, 2022.

6. M. Suau, J. He, M. T. J. Spaan, F. A. Oliehoek. Speeding up Deep RL through Influence-augmented Local Simulators. Extended Abstract. Autonomous Agents and Mult-Agents Systems, 2022.

5. M. Suau, J. He, E. Congeduti, J. He, R.A.N. Starre, A. Czechowski, F. A. Oliehoek. Influence-aware Memory Architectures for Deep Reinforcement Learning in POMDPs. Neural Computing and Applications, Springer Journal, 2022.

4. M. Suau, A. Agapitos, D. Lynch, D. Farrell, M. Zhou, A. Milenovic. Offline Contextual Bandits for Wireless Network Optimization. Offline RL workshop, Neural Information Processing Systems, 2021.

3. N. Albers, M. Suau, F.A. Oliehoek. Using Bisimulation Metrics to Analyze and Evaluate Latent State Representations. Benelux Conference on Artificial Intelligence and Machine Learning, 2021.

2. J. He, M. Suau, F.A. Oliehoek. Influence-augmented Online Planning for Complex Environments. Neural Information Processing Systems, 2020.

1. M. Suau, E. Congeduti, R.A.N. Starre, A. Czechowski, F.A. Oliehoek. Influence-based Abstraction in Deep Reinforcement Learning. Workshop on Adaptive Learning Agents, Autonomous Agents and Mult-Agents Systems 2019.