TOWARDS SCALABLE PLANNING IN PARTIALLY OBSERVABLE, MULTI-AGENT ENVIRONMENTS

JONATHON SCHWARTZ

A thesis submitted for the degree of
Doctor of Philosophy

School of Computing

September 2024

DECLARATION

This dissertation is an account of research undertaken between September 2020 and May 2024 at the School of Computing, The Australian National University, Canberra, Australia.

The work presented in this thesis is that of the candidate alone, except where indicated by due literature reference and acknowledgements in the text. It has not been submitted in whole or in part for any other degree at this or any other university.

This thesis is based on two papers published in refereed scientific conferences, and one paper which has been submitted for publication and which will be part of an upcoming workshop. The development of ideas and research was undertaken with guidance from my supervisors Hanna Kurniawati and Marcus Hutter and published papers presented in the thesis were written collaboratively with them as well as other coauthors. Below I describe my contributions to the research in each chapter and associated book and papers.

- Chapter 2 is a survey of the scientific literature related to this thesis and has not been published outside of this thesis. My research forms a major contribution to all sections of this chapter.
- Chapter 3 is the paper Schwartz, Zhou, and Kurniawati [SZK22] where my research forms a major contribution to all sections of the paper.
- Chapter 4 is the paper Schwartz, Kurniawati, and Hutter [SKH23], which extends the earlier published work Schwartz and Kurniawati [SK23], where my research forms a major contributions to all sections of both papers.
- Chapters 5 and 6 together form the journal paper Schwartz et al. [Sch+25], with a shorter version published as the workshop paper Schwartz et al. [Sch+24]. My research forms a major contribution to all sections of both papers.

Jonathon Schwartz

July 21, 2025

To my sister Melissa,
your belief in me made me believe in myself.

ACKNOWLEDGEMENTS

I would like to begin by expressing my deepest gratitude to my supervisor Hanna, whose guidance and mentorship have been invaluable throughout my PhD journey. Over the years working together, Hanna provided me the space to explore many areas of computer science which has been fundamental in shaping my interests and the type of researcher I am today. I am forever grateful for her support.

I would also like to thank Marcus Hutter, my associate supervisor, for his valuable advice on various aspects of my thesis. Marcus's feedback on key papers and his emphasis on first-principle thinking had a lasting impact on the way I approached the more fundamental aspects of my research.

My gratitude also goes to Hongdong Li, the final member of my supervisory panel. Although our interactions were limited, I deeply appreciate his feedback during the final presentation and for his role as my final panel member.

This PhD journey would not have been possible without the School of Computing's administrative staff and the faculty within the intelligence systems cluster. Their efforts in creating a supportive and collaborative environment for learning and research were instrumental in my ability to focus on and complete this thesis.

A special thanks goes to my lab colleagues, whose discussions and camaraderie made the research process both intellectually stimulating and enjoyable. I am especially grateful to Rhys Newbury, for being an amazing collaborator and someone I could always bounce ideas off of, and to Ruijia Zhou, for his help on the paper that became the third chapter of this thesis.

I also want to extend my thanks to the GRLA group, particularly Matthew Aitchison and Michael Bennett, for the conversations and motivation provided through our weekly AGI reading group and other discussions. These sessions were a constant source of inspiration and helped keep me engaged and excited throughout my PhD.

I am deeply grateful to my family — my father, Peter, my mother, Marie, and my sisters, Melissa, Neera, Claire, and Christina. Their encouragement and love have sustained me throughout this journey, and I could not have completed this work without their support.

To my friends, Nicholas, for being a constant source of advice and support, and to James, Ash, Alex, Dan, and Liz, whose presence, though limited due to distance, provided essential moral support during some of the most challenging moments of this thesis. I also thank the many other friends I have been fortunate to have in my life.

Finally, my heartfelt thanks go to my wife, Anna. We have been through so much together during the course of this thesis, from sharing a tiny room in a share house during the COVID-19 lockdowns in Canberra, to migrating to Montreal and eventually settling in San Francisco. I am incredibly lucky to have found someone so intelligent, funny, and willing to share this adventure called life with. Your unwavering love and support have been a big part of everything I have achieved during our time together.

ABSTRACT

Artificial intelligence (AI) continues to push the boundaries of what is possible for autonomous systems. However, there remains many tasks which AI is still unable or too unreliable to perform. Many of these require the execution of long sequences of actions in uncertain environments involving partial observability, stochasticity, and the presence of other agents. This thesis tackles critical challenges in this domain, focusing on developing methods for improving planning in partially observable, multi-agent environments.

A leading theoretical framework for modeling the other agents in partially observable environments is the Interactive-Partially Observable Markov Decision Process (I-POMDP). However, the computational demands of I-POMDPs have historically limited their applicability. The first contribution of this thesis is a novel online planning method, *Interactive Nested Tree Monte-Carlo Planning* (I-NTMCP), which improves the scalability and efficiency of planning in I-POMDPs. Our experimental evaluation shows I-NTMCP significantly outperforms existing state-of-the-art offline solvers, and is able to plan effectively to deeper nested reasoning levels than was previously possible.

Another core challenge for multi-agent systems research is the design of autonomous agents that can interact effectively with other agents without prior coordination. Type-based reasoning methods achieve this by maintaining a belief over a set of possible behaviors for the other agent. The second contribution of this thesis is the *Partially Observable Type-based Meta Monte-Carlo Planning* (POTMMCP) algorithm, which uses a novel meta-policy for guiding search during planning. We prove that POTMMCP converges to the optimal solution in the limit, given the model of the other agents is accurate. Furthermore, through empirical evaluations we demonstrate POTMMCP's ability to adapt online to diverse types of other agents across a range of environments, outperforming previous state-of-the-art methods in terms of efficiency and final performance.

An alternative method for tackling sequential decision-making problems is reinforcement learning (RL). Seamless integration of planning and RL promises to bring the best of both model-driven and data-driven worlds to multi-agent decision-making. The third contribution of this thesis is an in-depth empirical investigation of existing state-of-the-art planning methods and a method which combines planning and RL in the type-based reasoning setting. Our findings highlight the benefit of integrating

planning and RL in partially observable, multi-agent domains, while also serving to highlight several important directions for future research.

A key driver of progress in AI research is the availability of high quality, open-source benchmark environments. The final contribution of this thesis is the creation of *POSGGym*: a library for facilitating planning and RL research in partially observable, multi-agent domains. It provides a diverse collection of discrete and continuous environments, complete with their dynamics models and a reference set of policies that can be used for evaluation. POSGGym assimilates the various environments used throughout this thesis, and more, into an integrated Python library with the goal of aiding future research in planning and RL.

In summary, this thesis addresses the problem of optimal decision-making in partially observable, multi-agent environments. It presents algorithmic innovations, improvements, insights, and tools for planning in highly uncertain environments. Much work remains if we are to create autonomous agents with the ability to plan which are capable, robust, and safe. We hope the contribution of this thesis will help along this path.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ABBREVIATIONS

AI	**A**rtificial **I**ntelligence
RL	**R**einforcement **L**earning
MDP	**M**arkov **D**ecision **P**rocess
POMDP	**P**artially **O**bservable **M**arkov **D**ecision **P**rocess
POSG	**P**artially **O**bservable **S**tochastic **G**ame
Dec-POMDP	**D**ecentralized **P**artially **O**bservable **M**arkov **D**ecision **P**rocess
I-POMDP	**I**nteractive **P**artially **O**bservable **M**arkov **D**ecision **P**rocess
MCTS	**M**onte **C**arlo **T**ree **S**earch
UCB	**U**pper **C**onfidence **B**ounds
PUCB	**P**redictor + **U**pper **C**onfidence **B**ounds
UCT	**U**pper **C**onfidence **T**ree
PUCT	**P**robabilistic **U**pper **C**onfidence **T**ree

1

INTRODUCTION

Beyond just satisfying human curiosity, arguably the main ambition of artificial intelligence (AI) research is the creation of autonomous agents that can help humans better achieve their goals, whatever those goals may be. The last decade has seen perhaps more progress towards this end than in any previous time period in human history, due in large part by the coming of age of deep learning [LBH15] and large generative models [Dev+19; Bro+20b]. The capabilities of the current generation of foundational models continue to improve at a rapid rate [Bom+22; Zha+23], making it hard to predict exactly what they will achieve in the future. Despite this, there still remains many tasks which we as humans may like to automate which current AI systems are either unable or are too unreliable to perform.

Many of the tasks which AI systems struggle to perform reliably involve completing a sequence of actions or sub-tasks in uncertain and complex settings, such as the real-world. For example, driving from home to work, or having a robot that can assist people with limited mobility. While significant progress has been made on many such tasks, e.g. [Yur+20; APF22], we are still far from having ubiquitous and trusted autonomous systems helping us with much of our day-to-day work.

Given the recent advances in AI, it is worth asking ourselves: what key obstacles are preventing AI's wider application? While there are many aspects to answering this question –for example, cost, hardware, etc – this thesis focuses on trying to understand and tackle problems associated with operating in real-world environments. In particular, the real-world is inherently uncertain. Autonomous agents, and humans for that matter, operating in the real-world must deal with: limited information, randomness, and the actions of other humans or agents present in the environment. Limited information comes from agents having only partial observability of their environment at any one point in time. For example, while driving your view of the world may be obscured by buildings or other cars on the road. Randomness, or stochasticity, can come in many forms but is typically a property of the world that makes it hard to predict what the next state of the world will be after an agent has performed some action. A classic example of this would be trying to predict the

outcome of flipping a coin, but a more practical example might be how far it will take your vehicle to stop after applying the breaks on a wet or icy road. Finally, operating around other independent agents adds further uncertainty, since the actions of these agents changes the world and can be unpredictable and even change over time given the same situation. Just consider the situation of an agent crossing the road at a busy intersection. In this situation the agent needs to consider the actions of all the vehicles on the road, and there's no guarantee the same vehicle will behave exactly how they did the previous day. These three factors – partial observability, stochasticity, and other agents – compound to introduce high levels of uncertainty to decision-making in many real-world situations and presents a real challenge for the creation of reliable AI agents.

Beyond dealing with uncertainty, planning is another important aspect of many complex and valuable tasks we might want AI to help with. Many workloads rely on an agent's ability to assess their current situation and then, using a model of their environment, assess possible future trajectories in order to decide on the best sequence of actions to take. This is especially important for any general agent trying to perform a new task in the real-world, where mistakes can be costly and earlier actions have a large impact on chances of later success. For example, consider an autonomous vehicle driving and taking a wrong turn onto a long bridge experiencing peak traffic. This mistake will increase the time taken to get to the new destination and also means a new route has to be calculated. The actions the agent takes early on have a big impact on its ability to successfully reach its goal. Importantly, the agent must rely on a model of the world –such as maps, and knowledge of vehicle dynamics, etc– to plan which direction to go in, which lanes to be in for future turns, and which turns to take. These types of planning problems occur in almost any domain of human endeavor. Constructing agents that can reliably plan, even when faced with considerable uncertainty, has been a long standing challenge in AI.

In this thesis we address the problem of designing agents that can plan effectively in uncertain environments involving partial observability, stochasticity, and the presence of other agents. There have been many advances in building the theoretical frameworks for formalizing this problem (covered in Chapter 2), however, finding algorithms that can be tractably applied to practical real-world problems remains a key challenge. The work presented in this thesis makes contributions in the form of algorithmic improvements, which scale planning under uncertainty to larger, more complex environments.

1.1 OUTLINE & CONTRIBUTIONS

This thesis follows a canonical structure, starting with a survey of the relevant concepts and literature, followed by the main contributions. Specifically,

- In Chapter 2 we provide an introduction to planning in multi-agent, and partially observable environments and survey the relevant background and literature.
- Chapter 3 presents the first contribution of this thesis – the *Interactive Nested Tree Monte-Carlo Planning* algorithm – for planning while constructing a model of the other agent using nested reasoning. Based on Schwartz, Zhou, and Kurniawati [SZK22].
- In Chapter 4 we move beyond nested reasoning and instead focus on how best to incorporate uncertainty of the other agents' possible types of behavior into planning and propose the *Partially Observable Type-based Meta Monte-Carlo Planning* algorithm. Based on [SKH23].
- In Chapter 5 we conduct an extensive empirical analysis on the benefits of integrating reinforcement learning (RL) and planning. Based on [Sch+25].
- In Chapter 6 we introduce *POSGGym*: a software library for facilitating planning and RL research in partially observable, multi-agent environments. Based on [Sch+25].
- In Chapter 7 we conclude the thesis with a summary and discussion of potential directions for future research.

2

MULTI-AGENT PLANNING UNDER PARTIAL OBSERVABILITY

Planning is a fundamental component of any autonomous agent whose goal requires performing a sequence of actions, and it has been a core area of study in the artificial intelligence and robotics communities for many years. This chapter aims to give the reader foundational knowledge on planning so they can better understand the context and contributions of this thesis. We begin by first describing what planning is more formally, and then go on to introduce the key framework for planning under uncertainty in single agent settings, which much of the work in this thesis builds upon. Much of the remainder of this chapter is dedicated to the relevant mathematical frameworks and prior results in the area of decision-theoretic planning in partially observable, multi-agent environments, which is the main area of study for this thesis. We end the chapter with a brief background on reinforcement learning and recent work combining it with planning, which is important for the later chapters of this thesis. This chapter is organized so that each section covers a different concept or mathematical framework and starts by first providing the important definitions, before covering the existing literature.

2.1 WHAT IS PLANNING?

Before we dive into defining what planning is, let us first provide some motivation by describing the problem planning is trying to solve. Consider the example of a humanoid robot whose task is to clean a room. To us as humans this task can seem trivial, although arguably unpleasant. However, it actually involves a long sequence of actions across different levels of abstraction. At a high-level, our robot in this case must decide in what order will it perform the sub-tasks necessary to complete the main task of cleaning the room, for example does it start by vacuuming the floor, or picking up the toys. At a lower level of abstraction, the robot must decide what sequence of torques to apply to each of its joints in order to move in the physical environment to

complete its task. At these two levels of abstraction, and all levels between, the robot must decide on a *sequence of actions* in order to achieve some *goal*. Furthermore, it must do so based on what it perceives about the environment from its sensors, such as cameras, which may only provide a partial view of the environment at any one time.

The example we presented is a sequential decision-making problem. More formally, there is a decision-making entity, which we refer to as an *agent*, interacting with its surroundings, which we refer to as the *environment* (Figure 2.1). At each time-step t the agent receives an observation $o_t \in \mathcal{O}$ from the environment, and executes an action $a_t \in \mathcal{A}$. The environment then provides feedback to the agent in the form of a reward $r_{t+1} \in \mathbb{R}$. This cycle of observation, action, reward then repeats. In *continuing*, or *infinite horizon*, tasks this interaction repeats indefinitely. Conversely, in *episodic* or *finite horizon* tasks the interaction continues until some terminating condition is satisfied, e.g. the agent achieves some goal state or a fixed timestep limit is reached. The observation-action-reward sequence defines the agent's *experience*, which we refer to as the agent's *history*, denoted $h_t = o_0, a_0, r_1, o_1, a_1, r_2, ..., o_{t-1}, a_{t-1}, r_t, o_t$. The goal in a sequential decision-making problem is to design an agent that selects actions, given its history, that maximize the total amount of reward it collects during its lifetime. The action selection strategy of the agent is called its *policy* π which in the most general case is a function mapping the agent's history to a distribution over its actions.

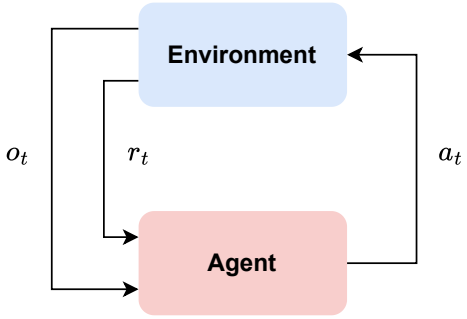


Figure 2.1: The Agent-Environment model for sequential decision-making.

At its core **planning is an approach for solving sequential decision-making problems**. The goal of planning is to find a policy which maximizes the expected total reward for the agent in a given problem. What distinguishes planning from other sequential decision-making approaches, is that computations are done using a model of the environment. A model in this case describes the effect each of the agent's actions will have on the state of the environment and the agent's subsequent observation and reward. Using this model a planning agent can simulate different sequences of actions to generate a policy, before executing that policy in the real environment (Figure 2.2).

Designing agents capable of planning has been a long standing challenge in the pursuit of AI. The ability to generate robust policies from environment models is crucial for any settings where interaction with the real environment is expensive or

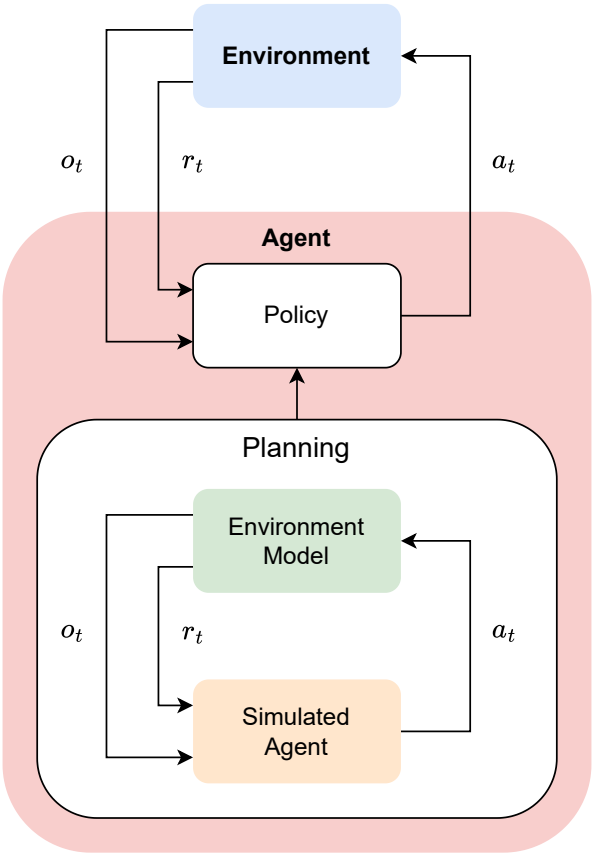


Figure 2.2: The Planning Agent-Environment model for sequential decision-making. The agent computes a policy by simulating interactions with a model of the environment, before executing an action in the real environment.

where mistakes may have large negative consequences, for example in robotics or autonomous driving where mistakes can result in damage to expensive equipment or have the potential to harm bystanders. Furthermore, planning offers a principled approach for improving the ability of agents to generalize to novel situations. In particular, when a planning agent is faced with a new situation, for example a cleaning robot placed in a new room, it can use its environment model to simulate possible trajectories for this new situation in order to find the best sequence of actions to take. Evidence of how planning can help improve robustness and generalization has been demonstrated extensively in board and card games [Sil+16; Sil+18; BS19] where searching (Section 2.2.4.2) over possible future moves was a crucial ingredient for achieving superhuman performance. This ability to generalize robustly to novel situations is becoming increasingly important as humanity builds ever larger, more general, and widely deployed AI systems, such as those built a top frontier language models [Dev+19; Bro+20b].

While planning is a fundamental ingredient of general, autonomous systems, it is far from a solved problem and many questions remain. For example, what model of the environment should be used? It is crucial to have a model that is able to

capture the complexities of the problems we care about, but can still be reasoned about mathematically. Once we have a model there is still the question of how can it be used to find an optimal, or at least good, policy in a way that is tractable in practice? Fortunately, a lot of work has been done over the years trying to answer these questions and more. In the rest of this chapter, we will present the key decision-theory models used in single and multi-agent environments, along with the existing approaches to planning using each model.

2.2 PARTIALLY OBSERVABLE MARKOV DECISION PROCESS

For tasks in environments involving only a single agent the Partially Observable Markov Decision Process (POMDP) [Son71; KLC98] has emerged as the most general and commonly used formal decision-making model. The popularity of the POMDP can be attributed to its ability to capture the important properties and uncertainties of sequential-decision making. Specifically, a set of possible states the environment can be in, the stochastic effect actions have on the state, what the agent can observe about the state, and finally how the agent is rewarded based on their actions (i.e. the agents goal). These four properties make POMDPs relatively easy to reason about, yet general enough for modeling a vast array of problems [Cas98; SPK13; Kur22].

2.2.1 *Definition*

More formally, a POMDP is a tuple $\langle \mathcal{S}, b_0, \mathcal{A}, \mathcal{O}, T, Z, R \rangle$ [Son71; KLC98]. $\mathcal{S}$ is the set of states the environment can be in, $\mathcal{A}$ is the set of actions available to the agent, and $\mathcal{O}$ is the set of possible observations the agent can perceive. Throughout this thesis we use s to denote an environment state in $\mathcal{S}$. $b_0 \in \Delta(\mathcal{S})$ is the initial state distribution defining the probability of a state s being the initial state of the environment.

$$b_0(s) \doteq Pr(s_0 = s). \tag{2.1}$$

$T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition function defining the effect of each action on the environment state, specifically $T(s, a, s')$ is the probability of transitioning to state s' after action a is performed in state s,

$$T(s, a, s') \doteq Pr(s_{t+1} = s' | s_t = s, a_t = a). \tag{2.2}$$

$Z : \mathcal{S} \times \mathcal{A} \times \mathcal{O}$ is the observation function defining what the agent can observe about the environment state, in particular $Z(s', a, o)$ is the probability the agent perceives observation o given the agent ended up in state s' after performing action a,

$$Z(s', a, o) \doteq Pr(o_{t+1} = o | s_{t+1} = s', a_t = a). \tag{2.3}$$

Finally, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ specifies the expected reward the agent will receive after taking action a in state s,

$$R(s, a) \doteq \mathbb{E}\left[r_{t+1} | s_t = s, a_t = a\right]. \tag{2.4}$$

It is also common to see the reward function defined using only the next state, $R(s')$, or the current state, action, and next state, $R(s, a, s')$. Throughout this thesis we use the reward function based on the current state and action, as this is the most common definition used in the literature, and sufficient for our purposes.

The agent's goal in a POMDP is to maximize the total amount of reward it receives over time from the environment. More formally, the agent seeks to maximize its *expected return*, where the return G_t is defined as some specific function of the reward sequence. The most commonly used function is the *discounted return*:

$$G_t \doteq \sum_{k=0}^{\infty} \gamma^k r_{k+t} \tag{2.5}$$

where γ is a parameter called the *discount rate* with value $0 \leq \gamma < 1$. The discount rate determines how the agent weights near-term versus long-term rewards. If $\gamma = 0$, the agent's goal will be to maximize immediate rewards only. As γ approaches 1, the agent's objective extends to maximizing rewards further into the future. The main advantage of discounting is that the return is finite (due to geometric discounting), even for infinite horizon tasks where there is not step limit. In finite horizon tasks it is also principally sound to have the return be simply the sum of rewards received from the current time t until the final time step T, $G_t \doteq \sum_{k=t}^{T} r_k$, though this is used less commonly, as it depends on the setting.

The current state s_t of a POMDP is a *sufficient statistic* of the entire interaction history of the agent. That is, knowing the current environment state is sufficient for making accurate predictions about future states, observations, and rewards. As such, environment states have the *Markov property* and are *Markov states* [Son71].

In general, in a POMDP the agent is unable to observe the current state directly, $o_t \neq s_t$, for this reason we say that a POMDP environment is *partially observable*. An important subclass of POMDPs is the set of environments where each observation is the state, $o_t = s_t$, such environments are considered *fully observable* and can be modeled using the **Markov Decision Process** (MDP) formalism [Bel57; Put14].

MDPs are a widely used model for single-agent sequential decision making, and are a strict subclass of POMDPs. More formally, MDPs are a POMDP where the state and observation spaces are the same $\mathcal{S} = \mathcal{O}$, and the observation function is simply the identity function for the next state: $Z(s', a, o) = 1$ if $o = s'$ otherwise 0. As such, when discussing MDPs it is convention to drop the observation space and function, and even the initial state distribution, so that an MDP is defined as the tuple $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$. We follow this convention in this thesis when discussing MDPs.

For the more general case of POMDPs where the agent does not observe the underlying environment state, a single observation does not have the Markov property. The implication of this is that in order to make decisions that maximize the expected return, the agent must consider not only its last observation but its entire history of observations and actions. To this end a POMDP agent must maintain some internal representation of its past observations and actions. In Figure 2.3 we show a general POMDP agent-environment model. The agent maintains an *internal state* c_t which is a representation of its past experience. At each step the agent's internal state is updated using an update function $c_t = U(c_{t-1}, a_{t-1}, o_t)$ given its last action, current observation, and previous internal state. The agent then selects its next action using a policy $\pi(a_t|c_t)$ which is a function of its internal state.

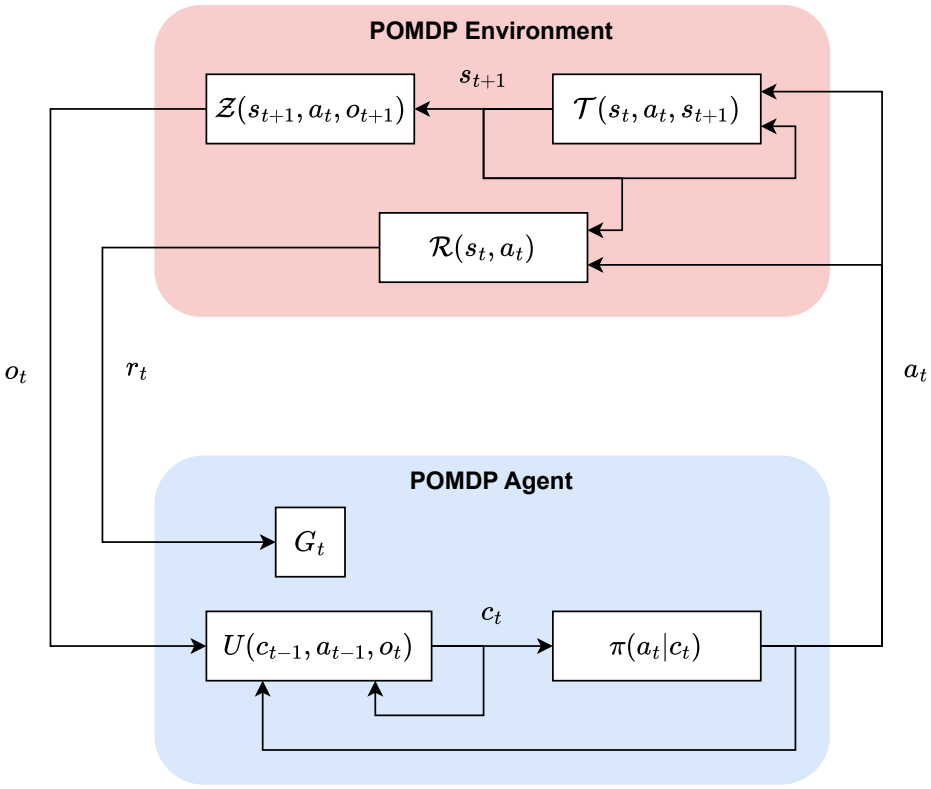


Figure 2.3: The general POMDP Agent-Environment model. The agent maintains internal state c_t, which is updated each step by the update function U, and the agent selects actions using policy π conditioned on c_t.

In Sections 2.2.2 and 2.2.3, we present some specific types of internal states that can be used for POMDP agents. Before that we first must go over what an agent's policy and value functions are.

2.2.1.1 *Policy and Value Functions*

As previously mentioned, the goal of the agent in a POMDP is to maximize the total reward it will receive in expectation. Stated more formally, the solution to a POMDP is a policy π^* that maximizes the agents expected return G from each internal state c_t. In the following we will focus on the case of expected discounted return, as that is the metric we use throughout this thesis

In order to find the optimal policy, we first need a way to measure how good a policy is. This is where the value function come in. The *value function* $V^\pi(c)$ of a policy π defines the agent's expected return when starting from a given internal state c and then following policy π. Similarly, the *action-value function* $Q^\pi(c, a)$ is the expected return starting from internal state c, taking action a, and then following policy π,

$$V^\pi(c) \doteq \mathbb{E}_\pi[G_t | c_t = c] \tag{2.6}$$

$$Q^\pi(c, a) \doteq \mathbb{E}_\pi[G_t | c_t = c, a_t = a] \tag{2.7}$$

where $\mathbb{E}_\pi$ is the expectation over the trajectories generated with policy π.

Finding a policy that maximizes the value function is equivalent to achieving the agent's goal of maximizing its expected return. For this reason we define specific functions related to the maximum possible value achievable by any possible. These are the *optimal value function* V^* and *optimal action-value function* Q^* which are defined as,

$$V^*(c) \doteq \max_\pi V^\pi(c), \forall c \in \mathcal{C} \tag{2.8}$$

$$Q^*(c, a) \doteq \max_\pi Q^\pi(c, a), \forall c \in \mathcal{C}, a \in \mathcal{A}. \tag{2.9}$$

where $\mathcal{C}$ is the space of possible internal states.

An *optimal policy* π^* is any policy which chooses actions that maximizes the action-value function from every internal state,

$$\pi^* \doteq \arg\max_\pi Q^\pi(c, a), \forall c \in \mathcal{C}, a \in \mathcal{A}. \tag{2.10}$$

While the value functions are unique, there may be more than one optimal policy for any given POMDP [Son71]. By definition all optimal policies achieve the same expected return and so are equivalent in terms of maximizing the agent's goal.

A big question that still remains is what representation to use for the internal state of the agent? In the following sections we present the most common approaches to answering this question.

2.2.2 *History Based Agents*

The full history of the agent h_t is a Markov State for a POMDP. By definition, h_t contains the maximum information the agent can know about its past experience, and given the full history the probability of future sequences can be computed [Hut09]. As such, the agent's history can be used as an internal representation for the purpose of selecting actions. Using our general framework of a POMDP agent, the internal state is the agent's current history h_t, the update function simply concatenates the most recent action and observation with the previous history $h_t = U(h_{t-1}, a_{t-1}, o_t) = h_{t-1}a_{t-1}o_t$, and the policy is a function of history $\pi(a_t|h_t)$ (Figure 2.4).

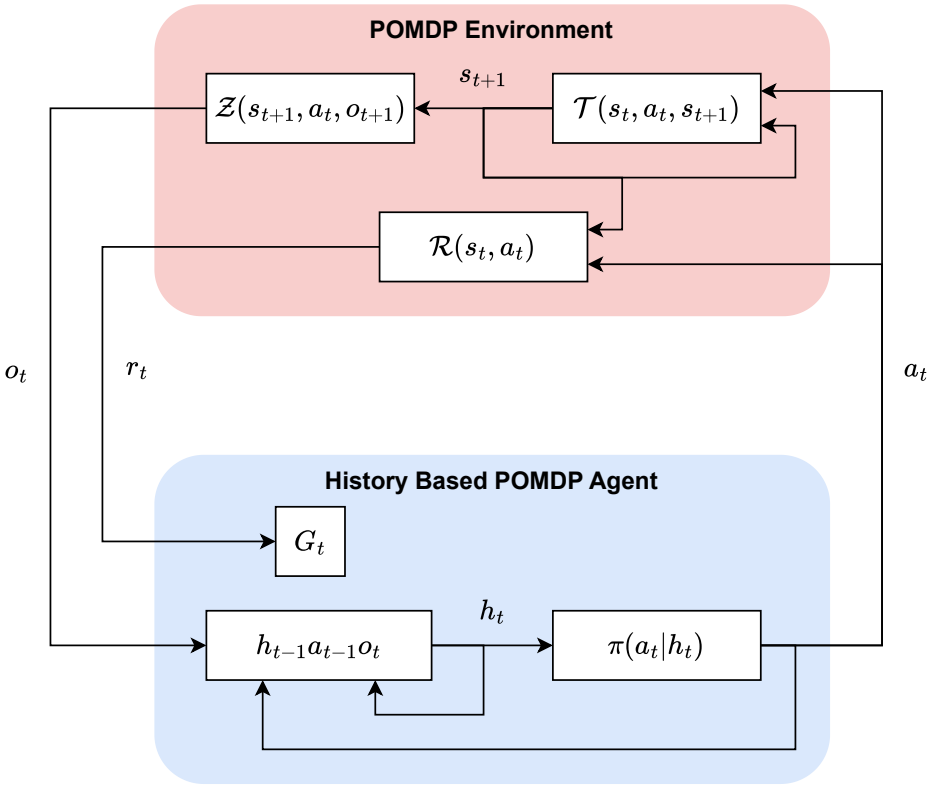


Figure 2.4: The history-based POMDP Agent-Environment model. The agent stores its entire action-observation history h_t for each step. The policy π selects actions conditioned on the agent's history.

The history based approach to POMDPs offers the most general agent framework for POMDPs. In fact, while being beyond the scope of this thesis, it is possible to extend this framework beyond POMDPs to more general environments where no assumptions are made about the existence of a finite underlying state space [Hut04;

Hut09]. Furthermore, it requires very few assumptions to be made about the form of the internal representation.

A fundamental practical limitation of history-based POMDP models is the growth in problem complexity with time horizon, coming from two sources. The first is the linear growth with t of the length of an individual history h, and hence the size of the agent's internal representation. The second, commonly referred to as the *curse of history* [PGT06], is the exponential growth with t of the space of possible time t histories $\mathcal{H}_t$, based on the size of the observation and action spaces $|\mathcal{H}_t| = O((|\mathcal{A}||\mathcal{O}|)^t)$. Both the linear growth of a single history and the exponential growth of the history space, pose difficult challenges to overcome when trying to design practical agents for anything more than very simple POMDPs with short finite-time horizons.

2.2.3 *Belief States*

In POMDPs where the environment is assumed to have a well defined underlying state space $\mathcal{S}$, it is possible to construct a sufficient statistic of the agent's history h_t. This statistic, called the *belief state*, is the distribution over the underlying states given the agent's history and initial belief [Ast65]. Given there are are a finite number of underlying states $\mathcal{S} = \{s_0, s_1, ..., s_n\}$, the belief state at time t is the vector $b_t \in [0,1]^n$ with components

$$b_t(s_i) = Pr(s_t = s_i | h_t, b_0), \tag{2.11}$$

for all states $s_i \in \mathcal{S}$. b_t is a probability distribution over the finite state space so $\sum_{s_i \in \mathcal{S}} b_t(s_i) = 1$.

The belief state b_t is a sufficient statistic for the agent's history h [SS73]. The time t belief state b_t can be computed from the previous belief state b_{t-1}, using the previous action a_{t-1}, and the current observation o_t. In this way the agent's belief state can be incrementally updated using Bayes' rule and the information from the previous time step. Specifically, given belief b, action a, and observation o the new belief b' in state $s' \in \mathcal{S}$ is defined by

$$b'(s') = \frac{Z(s', a, o) \sum_{s \in \mathcal{S}} T(s, a, s') b(s)}{Pr(o|a, b)} \tag{2.12}$$

where

$$Pr(o|a, b) = \sum_{s' \in \mathcal{S}} Z(s', a, o) \sum_{s \in \mathcal{S}} T(s, a, s') b(s) \tag{2.13}$$

is a normalizing factor independent of s'. The new belief state b' is constructed by performing the above calculation for each state $s' \in \mathcal{S}$. This process of updating the belief is commonly called *state-estimation* and is the function, denoted SE, by which the agent's internal state is updated each step, $b_t = SE(b_{t-1}, a_{t-1}, o_t)$ (Figure 2.5).

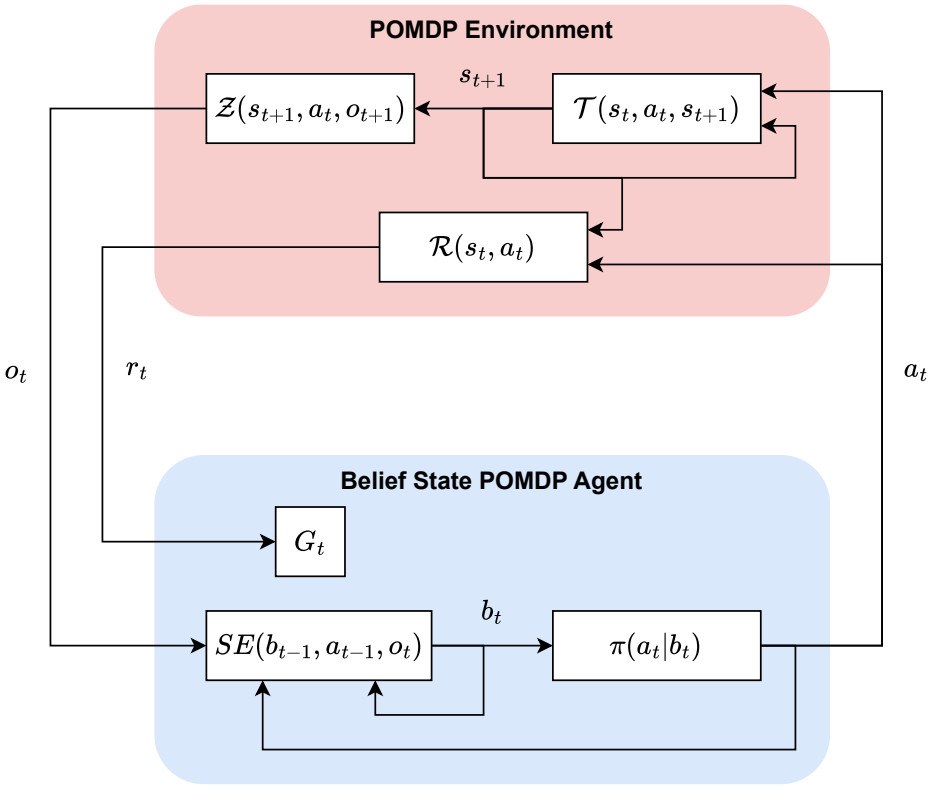


Figure 2.5: The belief state based POMDP Agent-Environment model.

A key feature of belief states is that their size remains constant even as t grows. The size, or dimensionality, of a belief state is purely a function of the size of the state space, which is assumed to be constant for a given POMDP. This is in contrast to history-based POMDP agent whose internal representation, the history h_t, grows linearly with t.

Belief states still present significant computational challenges. The first being that belief states are still affected by the *curse of history*; the space of possible belief states $\mathcal{B}$ grows exponentially with time t. At time t the size of the space of possible belief states is on the order of $O((|\mathcal{A}||\mathcal{O}|)^t)$. The second challenge is the so called *curse of dimensionality*: the dimensionality of belief states grows linearly with the size of the state space [PGT06]. So while the dimensionality of a belief state is constant for a given POMDP, the constant grows with the number of states in the POMDP. In later sections, we discuss approximate methods for overcoming both curses.

2.2.3.1 *Belief MDP*

Since belief states are Markov states for the agent's history they can be used to construct a continuous space *belief MDP* from the original POMDP [KLC98]. Each state in the belief MDP is a belief state, that is a distribution over the underlying state space of the original POMDP. The transition function in the belief MDP defines the probability of transitioning between belief states given an action, similarly the reward function maps from a belief state and an action to the expected reward.

Formally, a belief MDP is a tuple $\langle \mathcal{B}, \mathcal{A}, \tau, \rho \rangle$. $\mathcal{B}$ is the set of belief states which comprise the state space, $\mathcal{A}$ is the set of actions which are unchanged from the original POMDP, $\tau(b, a, b')$ is the transition function, $\rho(b, a)$ is the reward function. τ and ρ are defined from the original POMDP as follows:

$$\tau(b, a, b') = Pr(b'|a, b) = \sum_{o \in \mathcal{O}} Pr(b'|a, o, b) Pr(o|a, b), \tag{2.14}$$

$$\rho(b, a) = \sum_{s \in \mathcal{S}} b(s) R(s, a) \tag{2.15}$$

where

$$Pr(b'|a, o, b) = \begin{cases} 1 & \text{if } SE(b, a, o) = b' \\ 0 & \text{otherwise.} \end{cases} \tag{2.16}$$

Given the state-estimator is correct, an optimal agent policy in the belief MDP will give rise to optimal behavior in the original POMDP [Ast65; Son78].

While not widely used explicitly in practice, the existence of the equivalence between POMDPs and belief MDPs makes it possible to extend many theoretical results from the simpler MDP model to POMDPs [KLC98].

2.2.3.2 *Policy and Value Functions*

The policy and value functions in a POMDP using belief states are analogous to the functions we defined for internal states c, but replacing c with beliefs b. Specifically, a policy π for a belief state based POMDP agent is a function mapping beliefs to a distribution over actions $\pi(a|b)$. While the *value function* $V^\pi(b)$ under a policy π is a mapping from belief states to the expected return when starting from that belief state and then following π: $V^\pi(b) \doteq \mathbb{E}_\pi[G_t|b_t = b]$. Likewise, the *action-value function* $Q^\pi(b, a)$ is the expected return starting from belief b, taking action a, then following policy π: $Q^\pi(b, a) \doteq \mathbb{E}_\pi[G_t|b_t = b, a_t = a]$. The *optimal value function* V_b^* and *optimal action-value function* $Q^*(b, a)$ are the unique value functions that maximize the expected return from every belief.

An *optimal policy* π^* is any policy which selects actions that maximize the action-value function from every belief,

$$\pi^* \doteq \arg\max_{\pi} Q_b^{\pi} a, \forall b \in \mathcal{B}, a \in \mathcal{A}. \tag{2.17}$$

An important property of belief state value functions is that they satisfy recursive relationships between the value of a belief and the values of its successor beliefs. For the value function V^{π}, this relationship is described by the *Bellman equation* [Bel57],

$$V_b^{\pi} \doteq \sum_a \pi(a|b) \left[\sum_s b(s) R(s,a) + \gamma \sum_o Pr(o|a,b) V_{SE(b,a,o)}^{\pi} \right], \tag{2.18}$$

and *Bellman optimality equation* [Bel57],

$$V_b^{*} \doteq \max_a \left[\sum_s b(s) R(s,a) + \gamma \sum_o Pr(o|a,b) V_{SE(b,a,o)}^{*} \right]. \tag{2.19}$$

These equations form the basis of many POMDP planning algorithms. Furthermore, there always exists a deterministic policy that maximizes a value function V^{π} for any belief state [Son78].

2.2.4 *Planning in POMDPs*

Now that we have covered what a POMDP is, we can move onto the various methods for planning in a POMDP. As mentioned previously, the goal of planning in a POMDP is to find a return maximizing policy π given a POMDP model of the environment. Unfortunately, finding the exact optimal policy for a finite-horizon POMDP is PSPACE-complete [PT87], and thus considered intractable for all but the smallest problems. This has spurred researchers to instead focus their attention on approximately optimal planning methods, leading to significant progress being made in this direction over the years.

In the following we will introduce some of the important general components for planning, as well as many of the key algorithms for planning in POMDPs.

2.2.4.1 *Online versus Offline*

Planning approaches can be separated into two main classes: *online planning* and *offline planning*. Online planning algorithms interleave policy construction and policy execution and focus planning on improving the agent's policy for its current state [Ros+08]. Conversely, offline planning algorithms construct a complete policy before

the policy is executed in the environment [PGT06; SPK13]. Figure 2.6 shows a visual comparison of the two approaches.

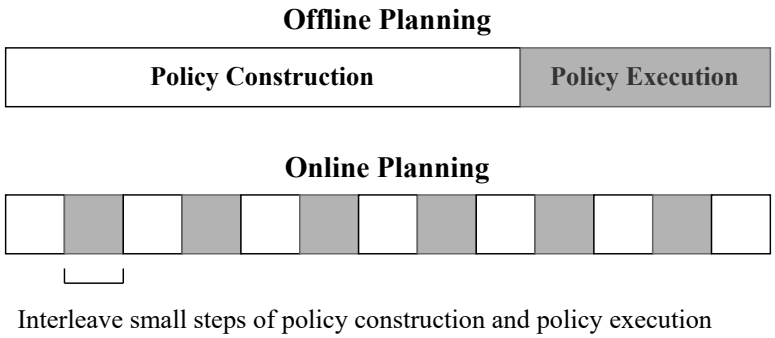


Figure 2.6: Comparison of offline and online planning approaches. *Reproduced from [Ros+08]*

.

Each approach has its advantages and disadvantages. Offline planning produces a policy for all possible beliefs/internal states which can then be reused as many times as desired. However, this comes at a significant computational cost, with the memory cost in particular quickly becoming impractical for large POMDPs. Conversely, online planners recompute the policy each time the planner is deployed, and may require more time per action during deployment. The advantage of online planners is that memory usage is significantly reduced since only the relevant information for the current state is stored. Additionally, online planning focuses on finding the optimal action for the current state, a much simpler search problem than finding, a global policy as offline planning does. These two features have meant that online planners have been successfully applied to much larger problems than offline planners.

2.2.4.2 *Search*

Search is a fundamental component of many planning algorithms. Similar to planning, the word search is widely used and has various meanings across different field. In this thesis we focus on search as applied in the field of planning under uncertainty [Ros+08].

In the context of planning under uncertainty, search is the process by which an agent, given a model of the environment, explores possible future states starting from a particular root state. Here a root state could be the state of the environment, in MDPs, or an agent's belief or internal state, in the context of POMDPs, or another definition of state depending on the problem.

Most search algorithms work by constructing a search tree from a root state. An example search tree is shown in Figure 2.7. Each node in the tree corresponds to a descendant state of the root state, and edges in the tree correspond to action-observation pairs. The set of nodes in the search tree represents the set of internal

states (e.g. beliefs) reachable from the root state, and traversing from the root node to any node in the tree is a sequence of actions and observations that lead to the state represented by that node.

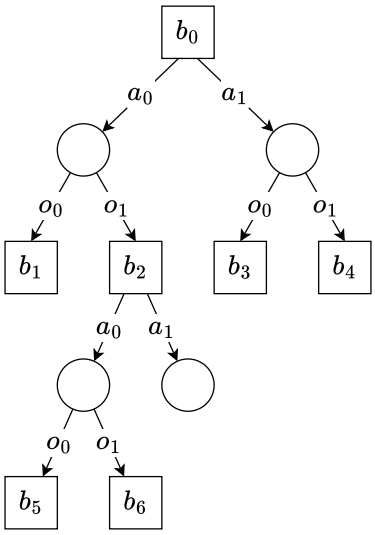


Figure 2.7: Example search tree for a POMDP with two actions and observations. The tree is constructed following different action-observation sequences from an initial internal state. The square nodes represent internal states of the agent (in this case beliefs), while the circles correspond to actions. Edges between state nodes are formed by action-observation pairs.

Search is generally used for two purposes in planning. The first is to explore the parts of the state space that are reachable from an initial state, or collection of initial states. This is used to restrict offline planning computations to only those parts of the space that are useful to the agent. The second use of search is for the selection of an action from a particular root state. This application of search is widely applied in online planning algorithms which focus planning on selecting the best action for the agents current state. Indeed, in MDPs where the environment is fully observed and thus updating the agent's current state is trivial, online planning can be reduced to search.

2.2.4.3 *Monte Carlo Tree Search*

Monte Carlo Tree Search (MCTS) is one of the most widely used algorithm for search in large problems [Cou06]. It uses Monte Carlo simulations to evaluate nodes of a search tree, where each node corresponds to a some state representation (e.g. beliefs, environment state, history, etc) generated following a sequence of actions and observations. Here we will use history h to specify the node corresponding to that agent history, with edges between nodes correspond to actions. During search, as the tree is constructed, for each action $a \in \mathcal{A}$ and node h statistics on the number of

simulations taking that action $N(h,a)$ (also called *visit count*) and the actions estimated value $Q(h,a)$ are tracked. These statistics are then used to guide future simulations.

During planning, simulations start from the root node h_0 and progress in three stages: selection, expansion, and backpropagation.

During the *selection* phase the current search tree is traversed using an *exploration policy* until a leaf node of the tree is reached. The exploration policy balances exploring new regions of the search space with exploiting known high value regions. *Upper Confidence Bounds* (UCB) [ACF02] is the most commonly used exploration policy. It treats action selection as multi-armed bandit problem and selects actions as a function of the actions current estimated value and number of times it has been previously selected. Specifically, the next action a_t from a given node h_t in the tree is chosen using the equation

$$a_t = \arg\max_{a \in \mathcal{A}} Q(h_t, a) + c\frac{\sqrt{2\log N(h_t)}}{N(h_t a)} \tag{2.20}$$

where $N(h_t) = \sum_{a \in \mathcal{A}} N(h_t a)$, and $c \in \mathbb{R}$ is a hyperparameter controlling exploration. Combining tree search with UCB in this way gives rise to the *Upper Confidence Trees* (UCT) algorithm [KS06].

Upon reaching a leaf node, the *expansion* phase evaluates and expands the leaf node by adding it to the tree and adding edges for each action. Evaluation involves estimating the nodes value. This is done using either Monte Carlo rollouts or pre-computed value functions. Monte Carlo rollouts continue the simulated trajectory until some terminal state or step limit is reached, with actions selected using a *rollout policy*, for example a uniform random policy. The rewards gained during the rollout are summed (and discounted, if using discounting), giving the value estimate for the new node. Conversely, if using a pre-computed value function this function is called for the leaf nodes state to get the value estimate. This can be much faster, and often produces better value estimates, but requires additional work to generate the value function prior to running the MCTS.

In the final *backpropagation* phase the nodes and edges visited along the simulated trajectory are updated by propagating the value estimate from the leaf node up to the root node of the tree. In particular, the visit count and value estimate of each selected edge (h_t, a_t) is updated as

$$N(h_t, a_t) \leftarrow N(h_t, a_t) + 1 \tag{2.21}$$

$$Q(h_t, a_t) \leftarrow Q(h_t, a_t) + \frac{R - Q(h_t, a_t)}{N(h_t, a_t)} \tag{2.22}$$

where R is the sample return achieved along the simulated trajectory starting from edge (h_t, a_t), taking into account the value estimate of the leaf node.

Having discussed some of the distinguishing components of planning in POMDPs, we can now move on to presenting key existing methods.

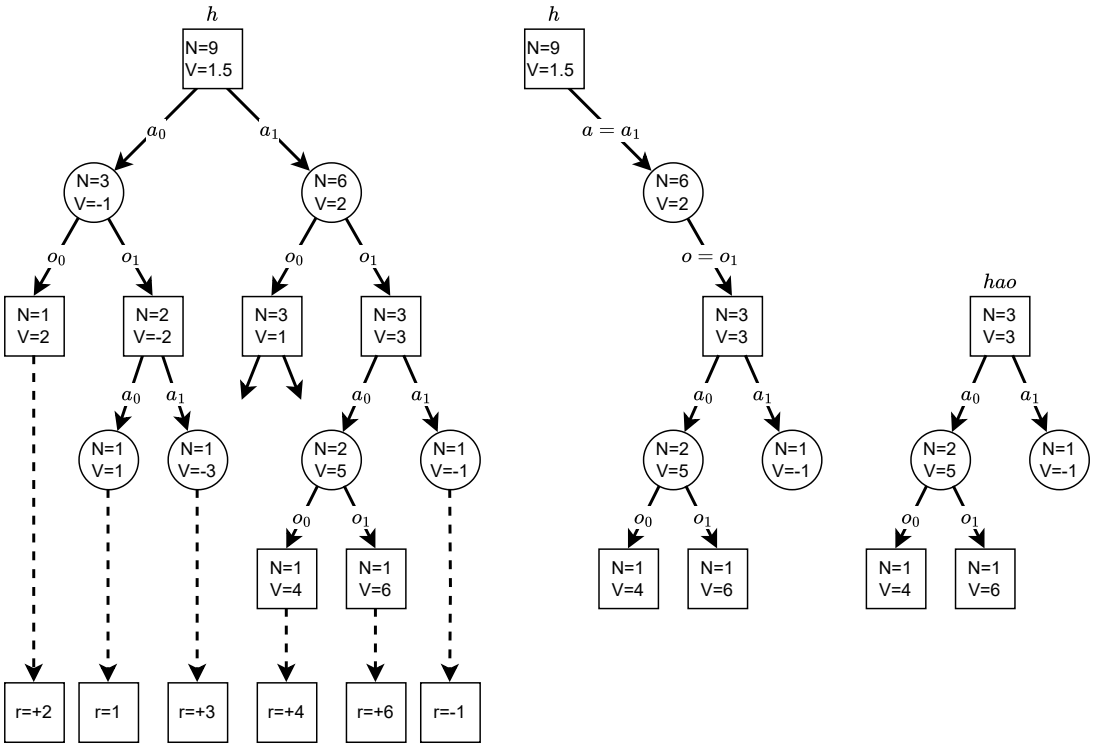


Figure 2.8: An illustration of POMCP in an environment with 2 actions, 2 observations, 50 states, and no intermediate rewards. The agent constructs a search tree from multiple simulations, and evaluates each history by its mean return (left). The agent uses the search tree to select a real action a, and observes a real observation o (middle). The agent then prunes the tree and begins a new search from the updated history hao (right). *Reproduced from [SV10].*

2.2.4.4 *Offline Algorithms*

Much of the early work in POMDP planning focused on developing offline planning algorithms. Initially, this focused on effective pruning methods to reduce memory costs [KLC98], however these methods still suffered from the curses of dimensionality and history and thus could only be applied to POMDPs with only tens or hundreds of states. The subsequent development of point-based algorithms, was able to greatly alleviate these limitations and made it possible to generate approximately optimal solutions to POMDPs with thousands of states [SS04; SS05; PGT06]. Extending point-based methods with search to limit planning to the optimally reachable belief spaces, has now made it possible for offline planning in POMDPs with hundreds of thousands or even millions of states [KHL08]. However, even with this major progress offline

planning still faces significant difficulty in scaling up to very large POMDPs, with longer horizons and larger action and observation spaces.

2.2.4.5 *Online Algorithms*

More recently, online methods for POMDP planning have greatly improved the scaling of planning for very large discrete and even continuous problems [SV10; Som+13; KY16; SK18].

There have been numerous approaches for online planning in POMDPs. At a high-level this has included methods based on heuristic search, branch-and-bound pruning, and Monte Carlo sampling. Heuristic search guides belief tree search via a heuristic [RC07; Ros+08]. While branch-and-bound pruning improves computational efficiency by pruning sub-optimal sub-trees using bounds on the optimal value [PTC05]. Lastly, Monte Carlo sampling uses random sampling to explore the belief space and/or estimate action values [BC99; Yoo+08; KMN02; SV10].

Of the various methods, those based on MCTS on a belief tree have proven to be the most practical for very large POMDPs. Silver and Veness [SV10] first proposed this type of method with the Partially Observable Monte-Carlo Planning (POMCP) algorithm (Figure 2.8). Subsequent works have since improved on POMCP, making it possible to handle dynamic environments [KY16], use sparse sampling [Ye+17], and tackle problems with large action spaces [WKK18] as well as continuous spaces [SK18; HK21]. The scalability of online MCTS is derived from two key features. First, they avoid storing the entire policy and/or value function by only computing the policy for the current belief only. Second, is the use of particle filters [RGT05] to represent beliefs which avoid high memory costs by never explicitly storing the entire state space for each belief.

Formally, a particle-based belief representation approximates the belief using a finite set of sample states or "particles"

$$\hat{b}(s) = \frac{1}{K} \sum_{k=1}^{K} \delta(s, s^k) \tag{2.23}$$

where K is the number of particles, s^k is the k-th particle (sampled state), and $\delta(s, s^k)$ is the Dirac delta function, which is 1 if $s = s^k$, otherwise 0. Each particle represents a possible state, and the belief over any state is proportional to the number of particles in that state.

With these advances planning under uncertainty using the POMDP is fast becoming practical for many real-world tasks, especially in areas such as robotics [Kur22].

2.3 PARTIALLY OBSERVABLE STOCHASTIC GAME

The *Partially Observable Stochastic Game* (POSG) [HBZ04] framework is arguably the most general formal model for sequential decision-making in environments containing multiple agents. POSGs are a strict generalization of POMDPs, that incorporate the actions, observations, and rewards of multiple agents. This makes them ideal for modeling scenarios involving partial observability and multiple decision-making agents with arbitrary incentives – whether competitive, cooperative, or mixed. As such, POSGs are applicable for a vast number of real-world problems where there is prevalent uncertainty.

2.3.1 *Definition*

Formally, a POSG is a tuple $\langle \mathcal{I}, \mathcal{S}, S_0, \vec{\mathcal{A}}, \vec{\mathcal{O}}, T, Z, R \rangle$ [HBZ04], where,

- $\mathcal{I}$ is a finite set of agents indexed 1, ..., N.
- $\mathcal{S}$ is a finite set of states.
- $S_0 \in \Delta(\mathcal{S})$ is the initial state distribution, where $S_0(s)$ denotes the probability that the environment is in state s at time $t = 0$.[1]
- $\mathcal{A}_i$ is the finite set of actions for agent $i \in \mathcal{I}$. $\vec{\mathcal{A}} = \times_{i \in \mathcal{I}} \mathcal{A}_i$ is the set of joint actions, where $\vec{a} = \langle a_1, ..., a_N \rangle$ denotes a joint action.
- $\mathcal{O}_i$ is the finite set of observations for agent $i \in \mathcal{I}$. $\vec{\mathcal{O}} = \times_{i \in \mathcal{I}} \mathcal{O}_i$ is the set of joint observations, where $\vec{o} = \langle o_1, ..., o_N \rangle$ denotes a joint observation.
- $T : \mathcal{S} \times \vec{\mathcal{A}} \times \mathcal{S} \to [0, 1]$ is the Markovian state transition function, where $T(s, \vec{a}, s') = Pr(s_{t+1} = s' | s_t = s, \vec{a}_t = \vec{a})$ denotes the probability that taking joint action $\vec{a}$ in state s results in a transition to state s'.
- $Z : \mathcal{S} \times \vec{\mathcal{A}} \times \vec{\mathcal{O}} \to [0, 1]$ is the Markovian observation function, where $Z(\vec{o}, s', \vec{a}) = Pr(\vec{o}_{t+1} = \vec{o} | s_{t+1} = s', \vec{a}_t = \vec{a})$ denotes the probability that taking joint action $\vec{a}$ in state s' results in joint observation $\vec{o}$.
- $R : \mathcal{S} \times \vec{\mathcal{A}} \to \mathbb{R}^N$ is the reward function defining the reward each agent receives when joint action $\vec{a}$ is performed in state s. $\vec{r} = \langle r_1, ..., r_N \rangle$ denotes a joint reward where r_i is the reward for agent i.

1 We use S_0 here instead of b_0, to make it clear that a belief in a POSG can be more than just a distribution over the environment state.

In Figure 2.9 we show the general POSG agent-environment model for the two agent setting. A POSG unfolds in discrete time steps. At each step, all agents $i \in \mathcal{I}$ simultaneously perform an action a_i and receive an observation o_i and reward r_i. This can continue for a finite or infinite number of steps, where the number of steps is called the *horizon*. A sequence of actions and observations of an agent i is an *agent history*, $h_{i,t} = \{o_{i,0}, a_{i,0}, o_{i,1}, ..., a_{i,t}, o_{i,t+1}\}$. Similarly, a *joint history* is a sequence of joint actions and joint observations, $\vec{h}_t = \{\vec{o}_0, \vec{a}_0, \vec{o}_1, ..., \vec{a}_t, \vec{o}_{t+1}\}$. $\mathcal{H}_{i,t}$ is the set containing all time t agent i histories, while $\vec{\mathcal{H}}_t$ is the set containing all time t joint histories. The *discounted return* $G_{i,t} = \sum_{k=t}^{\infty} \gamma^{k-t} r_{i,k}$ for an agent i is the total discounted reward accumulated from time t onwards, where γ is a discount factor.

The goal of each agent is to maximize their expected return. Like POMDPs, POSGs are partially observable so the agent must maintain some internal representation of its past observations and actions, $c_{i,t}$. This internal state could be the agent's history itself h_i, or some other representation such as a belief b_i over states possibly including properties of the other agent's internal state (e.g. see Section 2.5). An agent's policy maps the its internal state to a probability distribution over actions $\pi_i(a_{i,t}|c_{i,t})$. The *value function* $V_i^{\pi_i}(c_i)$ for an agent is the expected return starting from internal state c_i when following policy π_i, $V_i^{\pi_i}(c_i) = \mathbb{E}_{\pi_i}[G_{i,t}|c_{i,t} = c_i]$. Like POMDPs the value of a policy depends on the state dynamics of the environment. However, in POSGs a policy's value is also affected by the policies of the other agents present in the environment.

POSGs are very general in that they impose no assumptions about the incentives of each agent in the environment. Instead whether agents are incentivized to compete, cooperate, or do a mix of both, depends entirely on the reward function. This makes POSGs capable for modeling the plethora of multi-agent scenarios that exist. This generality also comes with additional complexity, so researchers have also focused on more restricted versions of POSGs more tailored to the problems they are interested in (Figure 2.10). The most notable of which are Decentralized-POMDPs (Section 2.4) and Interactive-POMDPs (Section 2.5).

2.3.2 *Planning in POSGs*

Finding a policy that maximizes an agent's value function is notoriously difficult in POSGs. Exactly solving finite-horizon fully cooperative POSGs is considered NEXP-complete [Ber+02]. General-sum POSGs pose further challenges due to the possible existence of multiple equilibriums and the difficulty of equilibrium selection [HBZ04]. For this reason there has been limited work on developing algorithms for solving the most general forms of POSGs.

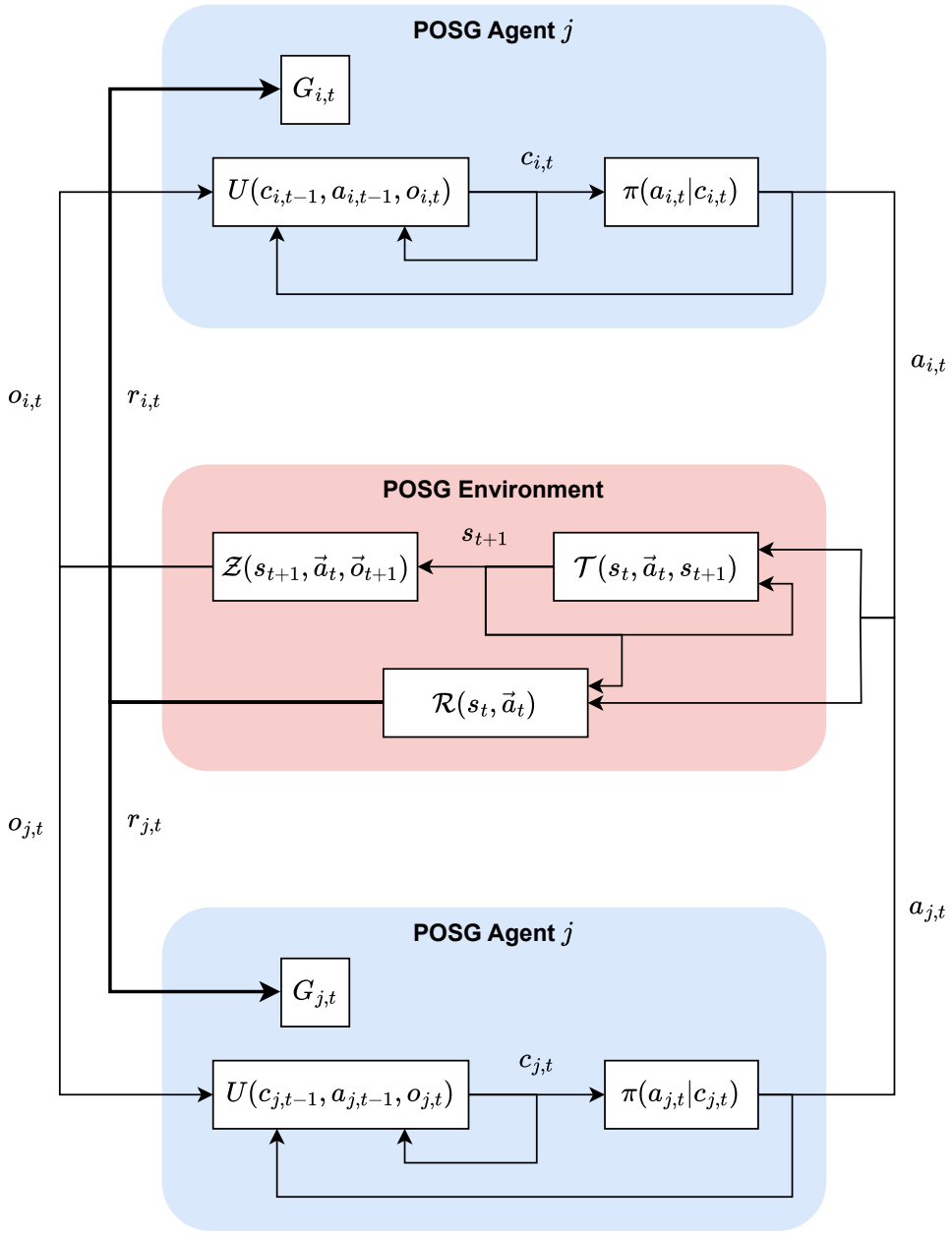


Figure 2.9: The general POSG Agent-Environment model for the two-agent setting. Each agent k maintains an internal state $c_{k,t}$ based on its action-observation history which is updated each step by the update function U. Agents select actions using their policy π_k conditioned on $c_{k,t}$. The environment is updated based on the joint action of both agents.

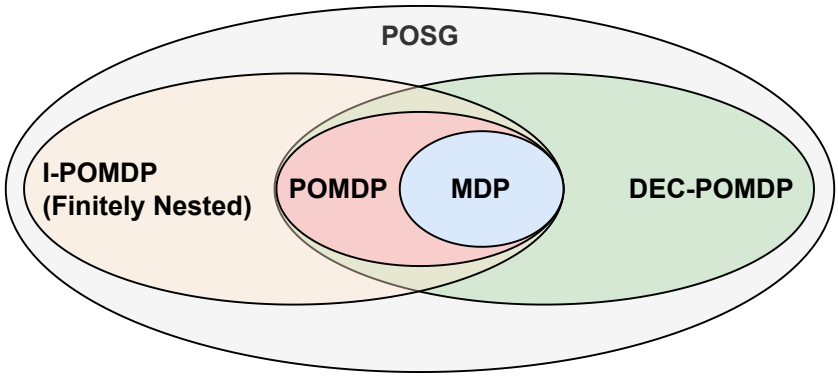


Figure 2.10: Relationship among various decision-theoretic models. *Reproduced from [Wei99; DZ13]*

Early works focused on dynamic programming approaches to solving finite-horizon POSGs [HBZ04; KZ09]. These algorithms generalize dynamic programming for POMDPs and iterated elimination of dominated strategies for normal-form games and can be used to find an optimal set of joint policies (one policy per agent in the environment) in fully-cooperative, finite-horizon POSGs. However, the number of possible policies for each player grows doubly exponentially with the horizon, that is on the order of $|\mathcal{A}_i|^{|\mathcal{O}_i|^t}$. As such, these algorithms are limited to only very small problem sizes. Additionally, they offer no solution to deal with multiple equilibrium and so do not provide a solution for POSGs in general. Selecting the best remaining strategy can be done using existing equilibrium selection methods, however these methods present their own challenges [LS08].

The computational complexity of solving POSGs has lead many researchers to instead focus on specific classes POSGs and related frameworks. A good example of this is the work by [HB19]. In their paper they introduce an approximate algorithm for solving two player, zero-sum POSGs with public observations (PO-POSGs). Their algorithm, which extends Heuristic Value Iteration Search is able to scale to non-trivial infinite horizon games (hundreds of states), however it relies on the specific assumptions of PO-POSGs.

2.4 DECENTRALIZED PARTIALLY OBSERVABLE MARKOV DECISION PROCESS

Environments where all agents share a common goal are a core area of multi-agent research. These types of problems arise in many different domains, examples include coordination between robots [Zil+02; WDM08] and autonomous vehicles [PT02; Tam97], as well as management of decentralized networks and logistics [Cog+06; OW96; Nai+05]. In all these problems, multiple agents work together to achieve a goal, but may be unable to share all their information at all times due to partial observability.

Such problems can be modeled as a *Decentralized Partially Observable Markov Decision Process* (Dec-POMDP) [Ber+02; OA16], which is the de facto formal decision-theoretic model for fully cooperative, partially observable, multi-agent environments.

2.4.1 *Definition*

A Dec-POMDP [Ber+02; OA16] is a POSG where all agents share a common reward function. It is defined as the tuple $\langle \mathcal{I}, \mathcal{S}, S_0, \{\mathcal{A}_i\}, \{\mathcal{O}_i\}, T, Z, R \rangle$ following the same definition of POSGs, with the exception that there is a single reward function R used by all agents. So for a given state and joint action all agents will receive the exact same reward. Thus, Dec-POMDPs define fully cooperative POSGs.

2.4.2 *Centralized Planning for Decentralized Control*

A common setting for Dec-POMDPs is *centralized planning for decentralized control* (also called *centralized learning for decentralized execution*). In this setting, centralized planning is used to find a global joint policy for all agents using shared knowledge and centralized control of each agent. Following centralized planning, localized policies are generated for each agent which are conditioned on the agents local action and observation history. During execution, there is decentralized control with each agent selecting actions using its local policy and its own action-observation history. It is assumed each agent has knowledge of the policy of all other agents, but not the other agents' actions and observations. This setup makes it possible to generate policies for a team of agents that effectively coordinate in a decentralized manner. The challenge in this setting is both to find a good global policy, as well as for each agent to maintain an accurate belief about the states of the other agents during execution.

2.4.3 *Planning in Dec-POMDPs*

Although theoretically simpler than POSGs, finding an exact solution for Dec-POMDPs is still considered intractable with finite-horizon Dec-POMDPs being NEXP-Complete ([Ber+02]). However, the shared reward function makes it easier to reason about the value of the game and finding equilibria, especially when there is centralized planning. For this reason and since Dec-POMDPs formalize many real-world problems of interest, there has been a greater focus on solution methods for Dec-POMDPs compared to POSGs. These have included exact methods based on dynamic programming [HBZ04; BC08; KZ09; ADZ09], optimization [AD10; ABZ10], heuristic search [SCZ05; OSV08; Oli+13], and model transformation [Dib+16]. In this thesis we focus on the more

general setting where agents have separate, and possibly different, reward functions. As such, we will not spend much time discussing methods designed specifically for the cooperative setting. For a more detailed exposition on Dec-POMDPs, we refer the reader to [OA16].

2.5 INTERACTIVE PARTIALLY OBSERVABLE MARKOV DECISION PROCESS

The Interactive Partially Observable Markov Decision Process (I-POMDP) [GD05] is a decision-theoretic framework for controlling a single agent in general partially observable, multi-agent environments. The key idea is to use an explicit model of the other agents and treat this model as part of the environment, transforming the problem into a single-agent decision-making problem, specifically a expanded POMDP. This idea of modeling the other agents as part of the environment is itself not novel [AS18]. Rather, the power of I-POMDPs arises from its use of recursive reasoning which makes it possible to model the other agents internal beliefs, which may themselves include beliefs about other agents, and so on. I-POMDPs incorporate this nested reasoning [Aum99] in a mathematically principled manner, making them valuable for scenarios requiring reasoning about other agents beliefs. I-POMDPs have been applied in a range of areas including security [Mei11; Ng+10], defense simulations [SP09a; SP09b], opponent modeling in games [Wun+12; DGB09], human-robot interaction [WW12], and human behavior modeling [Dos+10].

Another way to view I-POMDPs is as a formal approach to solving general POSGs where we are interested in controlling a single independent agent. I-POMDPs resolve the difficulty of multiple equilibria using an explicit model of the other agents based on nested reasoning. This simplifies the problem conceptually, and practically, while still being quite general. However, it does impose certain assumptions about the behavior of the other agents.

2.5.1 *Definition*

In the following we give a formal definition of an I-POMDP. For simplicity and since it will suffice for the purposes of this thesis, we restrict our definition to I-POMDPs with two agents, i and j. This definition can be extended to account for any finite number of agents [GD05].

Formally, an I-POMDP for agent i is a tuple $\langle \mathcal{IS}_i, b_0, \vec{\mathcal{A}}, \mathcal{O}_i, T_i, Z_i, R_i \rangle$ [GD05] where:

- $\mathcal{IS}_i = \mathcal{S} \times M_j$ is the set of interactive states, where:
  - $\mathcal{S}$ is the finite set of environment states

  - M_j is the set of possible models of agent j (explained further below).

- $b_0 \in \Delta(\mathcal{IS}_i)$ is the initial belief distribution over interactive states
- $\vec{\mathcal{A}} = \mathcal{A}_i \times \mathcal{A}_j$ is the joint action space where $\mathcal{A}_i$ and $\mathcal{A}_j$ are the finite set of actions for each agent, and $\vec{a} = \langle a_1, a_j \rangle$ denotes a joint action.
- $\mathcal{O}_i$ is the finite set of observations for agent i. o_i denotes an observation.
- $T_i : \mathcal{S} \times \vec{\mathcal{A}} \times \mathcal{S} \to [0, 1]$ is agent i's Markovian state transition function, where $T_i(s, \vec{a}, s')$ denotes the probability that taking joint action $\vec{a}$ in environment state s results in a transition to environment state s'.
- $Z_i : \mathcal{S} \times \vec{\mathcal{A}} \times \mathcal{O}_i \to [0, 1]$ is agent i's Markovian observation function, where $Z_i(o_i, s', \vec{a})$ denotes the probability that taking joint action $\vec{a}$ in environment state s' results in agent i receiving observation o_i.
- $R_i : \mathcal{IS}_i \times \vec{\mathcal{A}} \to \mathbb{R}$ is the reward function for agent i. Mapping interactive states and joint actions to a real valued reward for agent i.

The majority of components of the I-POMDP definition follow from the POSG definition. The key difference is the inclusion of the other agent's model m_j as part of the state using *interactive states*. This framing is what allows for nested reasoning.

An other agent model $m_j \in M_j$ is a tuple $\langle h_j, f_j, Z_j \rangle$ where:

- h_j is the history of agent j, where $h_j \in \mathcal{H}_j$ and $\mathcal{H}_j$ is the set of all possible histories of agent j.
- $f_j : \mathcal{H}_j \to \Delta(\mathcal{A}_j)$ is the policy of agent j, mapping history to a distribution over agent actions.
- $Z_j : \mathcal{S} \times \vec{\mathcal{A}} \times \mathcal{O}_j \to [0, 1]$ is the observation function defining how agent j is supplied with their observations.

Models $m_j \in M_j$ can be either:

1. **Sub-intentional** SM_j where models are non-recursive, defined as a mapping from the agent's observation history to action probabilities. This includes models where the other agent takes actions according to some to some fixed distribution, for example uniform random, as well as more sophisticated models such as finite state automata [PG17].
2. **Intentional** IM_j where models are themselves defined as I-POMDPs with beliefs about the environment state and the model of the other agent.

Sub-intentional models are relatively straightforward to incorporate into decision-making since they are fixed. Meanwhile, intentional models assume a Bayesian-rational agent and rely on the idea of agent *type* θ_j. Formally, $\theta_j = \langle b_j, \mathcal{A}_j, \mathcal{O}_j, T_j, Z_j, R_j, OC_j \rangle$, where,

- $b_j \in \Delta(\mathcal{S} \times M_i)$ is the belief of agent j over the environment state and the models of agent i.
- $\mathcal{A}_j, \mathcal{O}_j, T_j, Z_j$, and R_j are as for the I-POMDP definition.
- OC_j is the optimality criterion of agent j, e.g. to maximize total expected discounted reward or total expected average reward. Moving forward we assume this is to maximize total expected discounted reward.

For convenience we define an *agent frame* as $\hat{\theta}_j = \langle \mathcal{A}_j, \mathcal{O}_j, T, Z_j, R_j, OC_j \rangle$, since for many problems these values are constant across an agent's types. With the type differing only in the agent's belief. Using the notation of an *agent frame* an *agent type* can be defined as $\theta_j = \langle b_j, \hat{\theta}_j \rangle$, with the set of types for agent j denoted using Θ_j.

In I-POMDPs an agent's current belief b_i over the set $\mathcal{S} \times M_j$ is a sufficient statistic for its action-observations history [GD05]. This result stems from the fact that the definition of an *agent type* in I-POMDPs permits the modeling of nested beliefs. Since the belief of agent j, b_j, is a belief over states and the model of agent i, which itself can have beliefs about the model of agent j, and so on ad infinitum. Furthermore, it also allows for modeling uncertainty of the model $(\mathcal{O}_j, T_j, Z_j, R_j, OC_j)$ of the other agent. For example, modeling uncertainty over the agent's reward function.

A solution to an I-POMDP is a policy π that maps an agents beliefs b_i over interactive states to a distribution over its actions $\Delta\mathcal{A}_i$. The optimal solution is the policy that maximizes the agent's expected return from every belief. This is equivalent to finding π that maximizes the value function:

$$V(b_i) = \max_{a_i \in \mathcal{A}_i} \left\{ \sum_{is_i \in \mathcal{IS}_i} ER_i(is_i, a_i) b_i(is) + \gamma \sum_{o_i \in \mathcal{O}_i} Pr(o_i | a_i, b_i) V(SE(b_i, a_i, o_i)) \right\}$$

Where $ER_i(is_i, a_i) = \sum_{a_j \in \mathcal{A}_j} R_i(is_i, \langle a_i, a_j \rangle) Pr(a_j | is_i.m_j)$ is the expected reward given an interactive state and agent i's action, and $SE(b_i, a_i, o_i)$ is the belief update function.

This solution concept is the same as that of POMDPs, however in the case of I-POMDPs the agent's belief is over interactive states which include the beliefs of the other agents. This means that for I-POMDPs the solution concept is very clear, however due to the possibly infinitely nested beliefs the optimal value function and policy are only asymptotically computable [GD05].

2.5.2 *Finitely Nested I-POMDPs*

While computing the solution of general I-POMDPs with infinitely nested beliefs is not possible in practice, I-POMDPs also provide a principled model for decision-making using finitely nested beliefs [GD05]. In this setup, the 0^{th} level belief $b_{i,0}$ of agent i is the belief of agent i about the state of the environment. In this way, 0^{th} level beliefs are the beliefs of a sub-intentional model. The 1^{st} level belief $b_{i,1}$ is the belief over the state of the environment and the 0^{th} level models $m_{j,0}$ of the other agent, which includes their level 0 beliefs, $b_{j,9}$. This recursion can be extended to any desired level of nesting, with level l beliefs for $l > 0$ being beliefs for intentional agent models. The key assumptions made by finite-nested reasoning are, firstly, that each agent believes itself to have deeper beliefs than the other agent, and secondly, that each agent in the recursion is rational and will choose the optimal actions with respect to its beliefs.

The practical benefit of finitely nested I-POMDPs is that they can be solved via a finite recursion, where the solution at each nesting level is built from the solution of the level directly below. At level 0, agent models are sub-intentional and are standard POMDPs with the other agent's actions treated as "noise" in the environment. Thus, existing POMDP methods (see Section 2.2.4) can be used for solving level 0 I-POMDPs. The policy found from solving the 0^{th} level I-POMDP is then used to solve the level 1 I-POMDP, which is then used to solve the level 2 I-POMDP, and so on up to the top level of nesting.

Beyond being more tractable, finitely nested I-POMDPs also offer a potentially very useful model for sequential decision-making. In particular, there is growing evidence that for many partially-observable, multi-agent environments effective decision making can be done using only finite depth nested reasoning. Research in the fields of behavioral game theory and experimental psychology has found that humans reason to an average depth of 1.5 down the nested-beliefs hierarchy [CHC04]. Additionally, research pitting artificial recursive reasoning agents against each other in repeated games and sequential negotiation found that reasoning beyond a depth of two did not provide significant benefit [dVV13]. In the area of reinforcement learning (Section 2.6), Lanctot et al. [Lan+17] found that for their approach and test environment reasoning beyond a depth of five offered only slight improvements in final performance. Similarly for planning approaches, performance generally plateaued at a reasoning depth of one to two in the environments tested by Doshi and Perez [DP08] and Doshi and Gmytrasiewicz [DG09]. This evidence suggests in at least in specific settings, a finite depth of nested reasoning is sufficient for good decision-making.

2.5.3 *Planning in I-POMDPs*

While finitely-nested I-POMDPs have a well defined and theoretically computable solution, solving I-POMDPs in practice remains a challenge. In particular, at each level of reasoning an I-POMDP is a POMDP but with a greatly expanded state space. Thus I-POMDPs inherit the curses of dimensionality and history, which make solving POMDPs so difficult. At the same time solving an I-POMDP with nesting level l and at most M models at each level requires solving $O(M^l)$ POMDPs [GD05; SZ08]. Still the computational difficulty has not stymied researchers in coming up with novel approaches for planning in I-POMDPs.

Over the years a number of offline planning algorithms for I-POMDPs have been proposed. Many of these extend techniques first developed for solving POMDPs, including methods based on value iteration [DP08], particle filtering [DG09], and policy iteration [SD12]. Meanwhile, other methods have taken a different approach using model equivalence [RDG06] and structural problem reduction [HL13]. There have also been a works exploring I-POMDPs involving thousands of agents [SCD15] and where there is additional uncertainty about the transition and observation models of the other agents [Ng+12; HG18]. Lastly, I-POMDPs have been combined with iterative reasoning models leading to the creation of *parametrized I-POMDPs* (PI-POMDPs), which include uncertainty about the reasoning level of the other agents [Wun+11; Wun+12].

More recent research has explored online planning and learning for improved scalability in I-POMDPs. Panella and Gmytrasiewicz [PG17] model the other agents as a finite-state automata which is learned online. While Han and Gmytrasiewicz [HG19] introduce a novel neural network architecture based on the I-POMDP model structure. Both these approaches aim to simplify model learning by imposing inductive biases on the types of models that can be learned. Other works have instead focused on extending the POMCP algorithm designed for POMDPs [SV10] to the I-POMDP setting. Eck et al. [Eck+20] introduce the I-POMCP algorithm for online planning in open multi-agent systems, able to scale to a problem with up to 50 agents. Building on this work, Kakarlapudi et al. [Kak+22] propose CI-IPOMCP-PF for planning in I-POMDPs with explicit communication. In Chapter 3 of this thesis, we introduce another method in this vein, extending POMCP to allow planning in I-POMDPs with much deeper reasoning levels.

2.6 REINFORCEMENT LEARNING

To quote Sutton and Barto [SB18]: "Reinforcement Learning (RL) is simultaneously a problem, a class of solution methods that work well on the problem, and the field

that studies this problems and its solution methods". In this way RL and planning are similar, but the similarities don't stop there. RL and planning are both concerned with the problem of sequential decision-making. At their core, they both aim to find a policy π mapping an agent's action-observation history to the optimal action. The key difference between the two, is that whereas planning assumes a model of the environment to generate this policy, RL does not. Instead in RL the agent typically relies on interactions with the world to learn a policy via trial and error.

It is worth highlighting that even though RL does not assume access to a model of the environment, it can still be used for planning. Specifically, a model of the environment can be treated as an environment, and thus RL can be used to generate a policy using this model. If we view planning as any method that uses a model of the environment to generate a policy, then RL in this setting is being used for planning [SB18]. Conversely, some RL approaches, specifically model-based methods, learn a model of the environment as they interact with it, which can subsequently be used for planning to improve decision-making in the real environment (e.g. [Sut90; Sch+20]). So while planning and RL are often studied separately, they have significant overlap. Results from one area can often be applied to improve the other.

2.6.1 *Definition*

More formally, RL is the problem (and the class of methods applied to this problem) of finding a policy π in a sequential decision-making process (Section 2.1) when the model of the environment is unknown [SB18]. Applied to the formal models that have been introduced so far, the goal of RL is to find policy π that maximizes the agents expected return assuming the transition T, observation Z, and reward R functions are initially unknown. So whereas planning can use the model to explore possible future sequences of actions, in RL the agent must learn from its action-observation history – i.e. its *experience* – in order to improve its future rewards. Removing the assumption of knowing the environment model makes RL very general, and especially applicable in settings where an environment model is unavailable, or is difficult to construct (for example when dealing with pixel observations).

2.6.2 *Taxonomy*

RL is a flourishing field of research with a diverse array of approaches. Here we give a very brief overview of the main dimensions of variation of RL algorithms. This serves as a reference for the reader for when we mention these concepts in later chapters of this thesis.

MODEL-BASED VERSUS MODEL-FREE Model-based methods learn a model of the environment as part of learning a policy, while model-free methods learn the policy solely from the agents experience. Typically, model-based methods learn a model of the transition, observation, and reward dynamics of the environment which is then used for planning to improve the policy.

VALUE-BASED VERSUS POLICY GRADIENT Value-based RL learn the value-function for the environment, and then use this to compute the policy [Wat89; WD92]. These value functions are the same as discussed in previous sections, specifically a mapping from the agent's current state/history to its expected return following some policy. Policy-gradient methods on the other hand use the gradient of the agent's returns to improve the policy directly [Wil92]. *Actor-Critic* algorithms are a class of policy-gradient methods that also learn a value function which is used to improve policy updates [PS08; Bha+09].

2.6.3 *Deep Learning for Reinforcement Learning*

In a number of chapters in this thesis we use *deep RL* methods that represent both the value function and policy with a deep neural network (NN) [LBH15]. Integrating NNs with RL has been the underlying technique responsible for RL's many achievements over the last decade [Mni+15; Sil+16; Sil+18]. Representing the policy and value function using NNs allows them to be efficiently learned using backpropagation and stochastic gradient descent, making it possible to effectively leverage a large amount of compute using modern accelerators (e.g. GPUs). For more in-depth coverage of deep learning and RL with function approximators we refer the interested reader to the seminal books by: Goodfellow, Bengio, and Courville [GBC16] and Sutton and Barto [SB18]. Of particular interest to the reader are recurrent NNs which are used when applying RL throughout this thesis. However, a deep understanding of these topics is not crucial for understanding the core contributions of this thesis.

2.6.4 *Multi-Agent Training Schemes*

An important question when applying RL in multi-agent environments is how best to model the other agents during training? This question has given rise to a range of multi-agent training schemes.

One of the most well known is *self-play* [Tes94] where the RL agent plays against itself during training. This has proven very effective, especially for two-player, zero-sum games [Tes94; Sil+16; Sil+18; BS18], or settings where there is centralized learning

for decentralized execution [Ler+20]. While the results using this method have been impressive, there is evidence that purely self-play agents can be prone to over-fitting, resulting in agents that can be exploited [Lan+17; Gle+19] or that use arbitrary behavior that does not generalize well to novel co-players [Hu+20]..

Another approach is *population-based training* where population of policies are trained by matching up partner policies according to some schema [Lan+17; Jad+19; Tea+21]. This type of training has been used to generate superhuman performance in very large partially observable environments like StarCraft 2 [Vin+19] and Dota [Ber+19]. Over the years various schemas have been proposed including those based on cognitive hierarchies and nested reasoning [Lan+17; Cui+21], playing against older versions of a policy [HLS15; Ber+19], and many others.

Finally, it is possible to instead use RL to train a policy against a fixed model of the opponent. In this case the other agent is policy typically either learned from data or handcrafted [He+16], that is kept fixed throughout training. A good example of this is to generate a model of human behavior from data and then use RL to learn a policy that acts well with respect to this model. This technique was crucial for generating the initial policies for superhuman agents in competitive domains Go [Sil+16] and Starcraft 2 [Vin+19], where policies where initialized to replicate human players, before further training using self-play and population-based methods. More, recently it has been used to produce human level agents when playing against humans in the game of Diplomacy [Bak+22], which is a mixed incentive game requiring both cooperation and competition.

2.7 COMBINING PLANNING AND REINFORCEMENT LEARNING

Integrating planning and RL has been an active area of research for the multi-agent community for some years. Indeed, the combination of a policy learned via RL with online search was fundamental for expert performance in board games such as Go [Sil+16], Chess, and Shogi [Sil+18], as well as the card games Poker [BS18; BS19], Hanabi [Ler+20], and Diplomacy [Bak+22]. In these settings, where a model of the environment is available, planning can be used to improve an RL policy online in addition to improving sample efficiency of during training [Sch+20].

The dominant paradigm for combined planning and RL has been to use an RL policy to guide online MCTS. A challenge for planning in complex problems is the exponential growth of the number of possible sequence of actions and observations with the planning horizon. For naive methods this means that in the worst case (i.e. sparse rewards with long horizons) we can expect an exponential increase in the planning time required to find a good policy. Modern deep RL methods are able to

overcome this by effectively leveraging compute and modern hardware accelerators, but can suffer from other issues around over-fitting and robustness. By using an RL policy as a *search policy* within MCTS, it is possible to bias search towards high-value areas of the search space, effectively reducing the number of trajectories needed for exploration and making planning tractable even in complex environments. The benefit for the RL policy is that MCTS acts as a policy improvement operator [TG96], which is especially beneficial when the agent encounters states that are outside of the RL policy's training distribution where it is likely to be less reliable.

2.7.1 *PUCT*

An important method for integrating a policy into MCTS is the *Probabilistic Upper Confidence Tree* (PUCT) algorithm [Sil+16]. In PUCT the usual UCB action selection is replaced with PUCB ("Predictor + UCB") [Ros11]. Specifically, the next action a_t given agent history h_t is selected as

$$a_t = \arg\max_{a \in \mathcal{A}} \left\{ Q(h_t, a) + c\pi(a|h_t) \frac{\sqrt{N(h_t)}}{1 + N(h_t a)} \right\} \tag{2.24}$$

where $Q(h_t, a)$ is the current value estimate of taking action a given history h_t, $\pi(a|h_t)$ is the search policy (e.g. trained using RL), $N(h_t)$ is the number of times a given history has been visited during search, and c is a hyperparameter for controlling the influence of π relative to Q as the number of visits increase. Note, that various versions of PUCT exist with slightly different equations, however all share the same basic form presented in Equation 2.24.

PUCB biases action selection, and thus search, based on the policy π. If π is good, in that it leads to high expected returns, then more planning time will be spent in high-value areas of the search space. This has the effect of finding a better policy using less planning time. However, if π is bad, then more planning time than just naive planning (without a search policy) will be required to overcome the introduced bias. Fortunately, in the limit PUCB has the same guarantees as UCB [Ros11] (under mild assumptions), so given enough planning time even a bad policy can be corrected for.

This concludes the background chapter, which covers the most important common concepts for understanding the remainder of the thesis. Further required background is introduced within each chapter as needed.

3

NESTED ONLINE PLANNING FOR INTERACTIVE-POMDPS

This chapter has been published as:

Jonathon Schwartz, Ruijia Zhou, and Hanna Kurniawati. "Online Planning for Interactive-Pomdps Using Nested Monte Carlo Tree Search". In: *International Conference on Intelligent Robots and Systems*. IEEE, 2022, pp. 8770–8777

This chapter considers the problem of planning in *Interactive-Partially Observable Markov Decision Processes* (I-POMDPs), which is a general framework for modeling partially observable, multi-agent environments using nested-reasoning to predict the actions of other agents. The computational complexity of solving I-POMDPs has so far limited their use in complex environments with a large number of states. To address this, in this chapter we present a new online approximate solver for I-POMDPs, called *Interactive Nested Tree Monte-Carlo Planning* (I-NTMCP), which is able to plan effectively in much larger problems and to deeper reasoning levels than has previously been possible. Through experiments we demonstrate I-NTMCP's ability to generate substantially better policies up to an order of magnitude faster than existing state-of-the-art offline solvers. Furthermore, we demonstrate the proposed methods ability to effectively plan in a problem which, at the time of publication, was an order of magnitude larger than existing I-POMDP planning benchmarks.

3.1 INTRODUCTION

The Interactive Partially Observable Markov Decision Process (I-POMDP) is a framework for decision-making under uncertainty in multi-agent domains [GD05]. It generalizes the single-agent POMDP [SS73; KLC98] by incorporating behavioral models of the other agents present in the environment. Through recursive reasoning an I-POMDP allows an agent to explicitly model the other agents, which in turn model other agents,

and so on down to some finite depth. Similar to POMDPs, an I-POMDP agent never knows its exact state and must infer the best action to perform with respect to beliefs, where a belief is a distribution over the possible physical states. However, in an I-POMDP, the set of possible states is the joint product between the set of physical states and the set of possible models of the other agents and their beliefs. This makes I-POMDPs ideal for decision making problems involving uncertainty about both the environment and the other agents [Ng+10; Wan13; WW12].

While I-POMDPs possess many desirable properties, their application has been restricted to relatively small problems due to their high computational complexity. I-POMDPs inherit the intractability of POMDPs [PT87] while introducing additional complexity stemming from modeling of the other agents. This has motivated research into more tractable methods of solving I-POMDPs [DP08; DG09; HL13; SD15; HG19]. However, there is still a large gap in the size of the problems that can be handled when compared to existing online POMDP solvers [Kur22] which are capable of handling problems with millions of discrete states and even continuous state spaces with many dimensions.

Motivated by recent advances in POMDP planning, this chapter proposes an online I-POMDP planner based on Monte Carlo Tree Search (MCTS) [Cou06], called Interactive Nested Tree Monte-Carlo Planning (I-NTMCP). I-NTMCP is designed for the subclass of I-POMDPs where there is common knowledge of the initial belief and models among agents. This sub-class captures many problems of practical interest and has been a focus of prior I-POMDP solvers [HL13; HG19].

I-NTMCP focuses on computing the best action to perform from the current belief, which allows I-NTMCP to minimize the computation required in estimating the policy of the other agents. To find the best action from a belief, I-NTMCP constructs and maintains a sequence of inter-related belief trees, where each tree encodes an approximately optimal policy for an agent operating at a particular nesting level. In this way I-NTMCP computes both a policy for the planning agent and the other agents' model online. This sequence of trees is built bottom-up, from the lowest to the highest nesting level, using MCTS.

The results are promising. Tests on two competitive two-agent problems indicate that I-NTMCP can plan effectively in a problem that is two orders of magnitude larger than existing offline I-POMDP benchmark problems.

3.2 BACKGROUND

While I-POMDPs can be used to model any finite number of agents, in this chapter we restrict ourselves to environments with two agents. With this is mind, and for readability, we limit the following background to I-POMDPs with only two agents.

Formally, a finitely-nested I-POMDP for an agent i with nesting level l interacting with another agent j is a tuple $\langle \mathcal{IS}_i, b_0, \vec{\mathcal{A}}, \mathcal{O}_i, T_i, Z_i, R_i \rangle$, where:

- $\mathcal{IS}_{i,l}$ denotes the set of *interactive states* defined as, $\mathcal{IS}_{i,l} = \mathcal{S} \times M_{j,l-1}$ for $l > 0$, and $\mathcal{IS}_{i,0} = \mathcal{S}$, where $\mathcal{S}$ is the set of physical states and $M_{j,l-1}$ is the set of possible models for agent j and nesting level $l-1$.
- $\vec{\mathcal{A}} = \mathcal{A}_i \times \mathcal{A}_j$ is the set of joint actions of all agents. Where $\vec{a} = \langle a_i, a_j \rangle$ denotes a joint action.
- $\mathcal{O}_i$ is the finite set of observations for agent i. Where o_i denotes an observation for agent i.
- $T_i : \mathcal{S} \times \vec{\mathcal{A}} \times \mathcal{S} \rightarrow [0,1]$ is the Markovian state transition function, defining $Pr(s'|\vec{a}, s)$.
- $Z_i : \mathcal{S} \times \vec{\mathcal{A}} \times \mathcal{O}_i \rightarrow [0,1]$ is the Markovian observation function, defining $Pr(o_i|\vec{a}, s)$.
- $R_i : \mathcal{S} \times \vec{\mathcal{A}} \rightarrow \mathbb{R}$ is the reward function.

In an I-POMDP, the environment interaction unfolds in discrete time steps. At each step, each agent i simultaneously performs an action a_i and receive an observation o_i and reward r_i. Here, 'simultaneous' means that both agents select their actions independently without observing the other agent's current action choice. The resulting state transition, observations, and rewards depend on the joint action of both agents. The interaction can continue for a finite or infinite number of steps. A sequence of actions and observations of agent i is an *agent history*, $h_i^t = \{a_i^1, o_i^1, ..., a_i^t, o_i^t\}$[1]. Similarly, a *joint history* is a sequence of joint actions and joint observations, $\vec{h}^t = \{\vec{a}^1, \vec{o}^1, ..., \vec{a}^t, \vec{o}^t\}$, where $\vec{o} = \langle o_i, o_j \rangle$ denotes a joint observation. $\mathcal{H}_i^t$ is the set containing all time t agent i histories, while $\vec{\mathcal{H}}^t$ is the set containing all time t joint histories. For this chapter we assume the optimality criteria for each agent is to maximize its expected discounted *return*, which for an agent i is the total discounted reward accumulated from time t onwards, $G_i^t = \sum_{k=t}^{\infty} \gamma^{k-t} r_i^k$, where γ is a discount factor.

Similar to [HL13; SD15; HG19], I-NTMCP focuses on the class of I-POMDPs where the frame, $\hat{\theta}_j = \langle \vec{\mathcal{A}}, \mathcal{O}_j, T_j, Z_j, R_j \rangle$, for each agent is common knowledge and is

1 Here and in the remainder of this chapter the superscript notation is used to specify the time index.

fixed, the transition function is the same for both agents, $T = T_i = T_j$, and at each level the modeling agent only considers the intentional models of the other agent that are one level below. Hence, we simplify the definition of the set of intentional models to be the set of possible beliefs of the other agent over the interactive states, $\Theta_{j,l-1} = \{b_{j,l-1} : b_{j,l-1} \in \Delta(\mathcal{IS}_{j,l-1})\}$. Level 0 beliefs do not explicitly consider other agents, instead modeling them as part of the environment and thus only consider the environment state. At level 1 the belief, $b_{i,1}$, is over the state of the environment and the level 0 beliefs of the other agent $b_{j,0}$. This recursive definition can be extended to any desired level of nesting. Using this well defined hierarchy of nested beliefs, I-POMDPs can be solved via a finite recursion, where the solution at each belief level is built using the solution of the level directly below it [GD05].

Various exact and approximate I-POMDP solvers have been proposed, including methods based on model equivalence [RDG06], particle filtering [DG09], value iteration [DP08], structural problem reduction [HL13], policy iteration [SD15], modeling other agents as finite state-automata [PG17], and function approximation using deep learning [HG19]. Furthermore, I-POMDPs solvers when transition and observation models are initially unknown have also been proposed [Ng+12; HG18].

In this work we extend advances in online planning for POMDPs, specifically methods based on MCTS [Cou06; SV10], to the multi-agent setting using the I-POMDP formalism. There have been a number of works applying online Monte Carlo planning to I-POMDPs. Panella and Gmytrasiewicz [PG17] present a method for online Monte Carlo planning using learned subintentional models of the other agent. Similarly, Eck et al. [Eck+20] develop a method designed specifically for open-world I-POMDPs with many agents. Kakarlapudi et al. [Kak+22], published around the same time as this work, apply MCTS to I-POMDPs with communication. In contrast to these prior works, we propose a method for general two-agent I-POMDPs and go a step further by incorporating intentional models of the other agent to an arbitrary reasoning depth.

3.3 OVERVIEW OF I-NTMCP

3.3.1 *Problem Setting and Assumptions*

I-NTMCP is an online solver for the subclass of I-POMDPs where there is a common initial belief over states, b_0, and where all agents have knowledge of the other agents' frames and share a transition function. While the observation space, observation function, and reward functions of each agent are common knowledge, no assumptions are made about the form of the reward and observation functions. The assumptions we make are used in other I-POMDP solvers, however, unlike many other I-POMDP

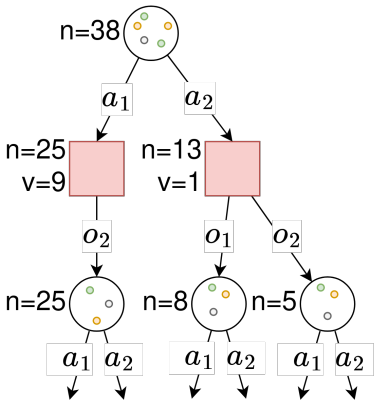


Figure 3.1: An individual I-NTMCP search tree for an agent with 2 actions and 2 observations. Circle nodes show the agent's belief for a given history, represented using a set of particles. While square nodes represent the agent's actions. The visit count n is shown for each node, along with the current value estimate v for each action.

solvers which require an explicit representation of the I-POMDP functions, $\langle T, Z, R \rangle$, I-NTMCP only requires a generative model $\mathcal{G}$ of the joint environment dynamics. This requirement of only a generative model is especially useful for domains that are difficult to model or where a compact representation of the transition or observation probabilities may not be available, as is often the case for large problems.

Similarly to the majority of prior work on I-POMDPs, in this chapter we restrict ourselves to environments with two agents. Conceptually, it is possible to extend I-NTMCP to environments with more than two agents, however for N agents this would require maintaining $(N-1)^l$ belief trees for a given nesting level l. This complexity is not unique to I-NTMCP and problems with more than two-agents are a challenge for I-POMDP solvers in general, especially when dealing with deeper nesting levels. Conversely, the online sample-based nature of I-NTMCP opens up a number of avenues of investigation for efficiently scaling up I-NTMCP to more than two agents - a possibility for future work.

3.3.2 *Algorithm Overview*

At a high level, I-NTMCP constructs and maintains a sequence of inter-related belief trees. Figure 3.1 shows a detailed representation of an individual tree, while Figure 3.2 illustrates the nested tree data structure used in I-NTMCP. Each tree represents the set of beliefs reachable under an optimal policy and encodes an approximately optimal policy for an agent with a particular nesting level. We denote the sequence of inter-related belief trees for agent i as $\mathcal{T}_{i,L}, \mathcal{T}_{j,L-1}, \mathcal{T}_{i,L-2}, \cdots, \mathcal{T}_{k,0}$, where L is the nesting level being considered, and where $k = i$ if L is even and $k = j$ if L is odd. The notation $\mathcal{T}_{k,l}$ refers to the belief tree of agent k, where $k \in \{i, j\}$, operating at nesting level $l \in [0, L]$.

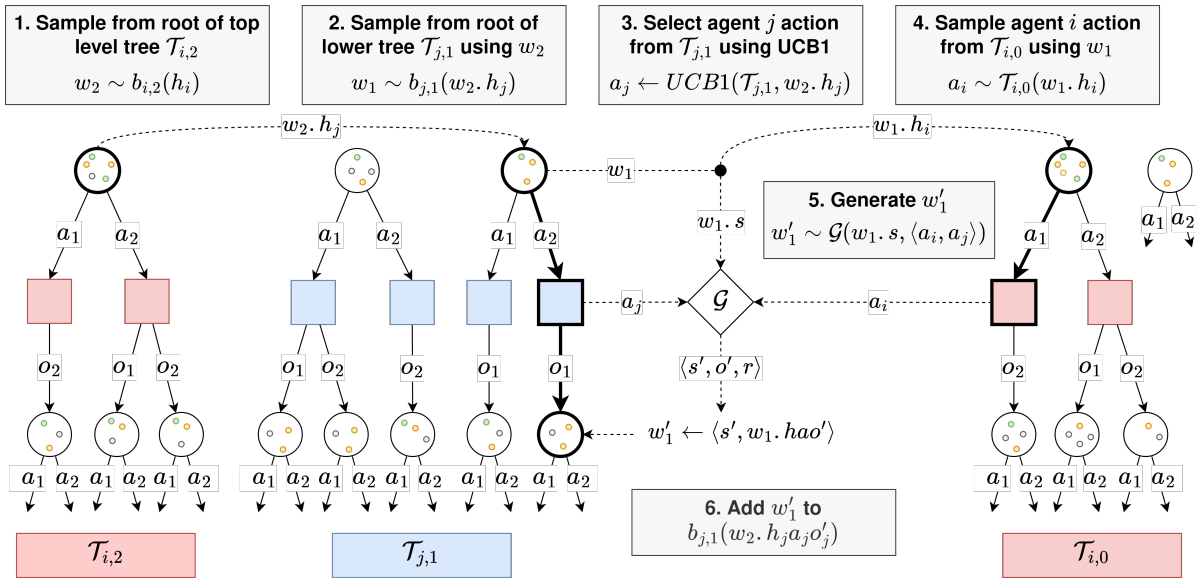


Figure 3.2: Top-down construction of the I-NTMCP Nested Trees Data structure. Shown here is the first step of a single simulation for I-NTMCP using a nesting level of two for an environment involving two agents i (red) and j (blue) who both have two actions and observations. In this case the planning is being done for agent i, and the level 1 tree of agent j is being expanded. Following step 6, the simulation continues until a leaf node is reached, at which point a rollout is used to estimate the value of the leaf node.

Each node of $\mathcal{T}_{k,l}$ is associated with an agent history, h_k, and represents a belief $b_{k,l}$ for agent k when operating with nesting level l. Along with the belief, each node maintains an information tuple $\langle N_{k,l}, V_{k,l} \rangle$, where $N_{k,l}$ is the number of times the node has been visited and $V_{k,l}$ is the value of the belief represented by the node. Each edge of $\mathcal{T}_{k,l}$ corresponds to an action–observation pair $\langle a_k, o_k \rangle$ where $a_k \in \mathcal{A}_k$ and $o_k \in \mathcal{O}_k$. An edge from $b_{k,l}$ to $b'_{k,l}$ implies that there is a pair of action–observation $\langle a_k, o_k \rangle$ such that, if at belief $b_{k,l}$ agent k performs action a_k and perceives observation o_k, agent k's belief will become $b'_{k,l}$.

Now, the question is how do we represent the beliefs of the other agent. I-NTMCP follows the nested-belief idea of I-POMDP, but represents the belief of the other agent using the agent's histories, rather than an explicit distribution over interactive-states. The use of histories to represent beliefs is aligned with the representation used in many state-of-the-art online POMDP solvers. Specifically, we define a history-state w^t as a tuple $w^t = \langle s^t, \vec{h}^t \rangle$, where $s^t \in \mathcal{S}$ is the physical state and $\vec{h}^t = h_i^t \times h_2^t$ is the joint history of agent i and agent j. $\mathcal{W}^t$ denotes the set of all time t history-states. For notational convenience, we denote agent k's history contained in the history-state w^t using $w^t.h_k^t$, and similarly for their action $w^t.a_k^t$ and observation $w^t.o_k^t$. This notation also applies for the state $w^t.s^t$, joint history $w^t.\vec{h}^t$, joint action $w^t.\vec{a}^t$, and joint observation $w^t.\vec{o}^t$. When t is clear, we omit the time superscript, in which case $w = w^t$.

Using history-states, a node in tree $\mathcal{T}_{k,l}$ associated with history h_k^t is a belief $b_{k,l}$ where $b_{k,l}(w^t, h_k^t) = P(w^t \mid h_k^t)$, for $k \in \{i, j\}$ and $w^t \in \mathcal{W}^t$. To represent each belief, I-NTMCP uses a set of unweighted particles of history-states, analogous to using state particles in Monte-Carlo planning for POMDPs [SV10].

I-NTMCP expands the tree, and also updates beliefs, by evolving each particle. To evolve a particle w in belief node $b_{i,l}(\cdot\,, h_i)$, I-NTMCP must estimate the action that will be taken by the other agent. This can be inferred as the best action from the belief node $b_{j,l-1}(\cdot\,, w.h_j)$ in tree $\mathcal{T}_{j,l-1}$ at one nesting level lower. This dependency forms the inter-relation between the belief trees. In the next section we present the details of how each tree is constructed.

The key novelty of I-NTMCP lies in its nested computational structure that explicitly models the other agent's decision-making process. Unlike standard MCTS which searches over primitive actions, I-NTMCP recursively constructs and solves the other agent's planning problem at each node. This nested structure allows the algorithm to reason about how the other agent will respond to different scenarios, going beyond simple action selection to model the full decision-making process. The model affects decision-making by providing the transition and observation functions needed to simulate both agents' future interactions during the nested planning process.

3.4 DETAILS OF I-NTMCP

I-NTMCP takes as its inputs the joint action space $\vec{\mathcal{A}}$, the initial belief over physical states $b_0 \in \Delta(\mathcal{S})$, a fixed policy for the other agent at level 0 $\pi_{-k,0}(a_{-k}|h_{-k})$ corresponding to a subintentional model (e.g. uniform random), and a generative dynamics model $\mathcal{G}$ of the environment, $\langle s^{t+1}, \vec{o}^{t+1}, \vec{r}^{t+1}\rangle \sim \mathcal{G}(s^t, \vec{a}^{t+1})$. Given this information, I-NTMCP constructs the sequence of inter-related belief trees online and bottom-up using MCTS. After each real action and observation, I-NTMCP updates beliefs top-down using particle filtering [RGT05].

3.4.1 *Constructing the Sequence of Inter-Related Trees*

To begin, at time $t = 0$, I-NTMCP initializes the root of each tree $\mathcal{T}_{i,L}, \mathcal{T}_{j,L-1}, \cdots, \mathcal{T}_{k,0}$ to represent the initial belief $b_0 = b_{k,l}(\langle s^0, \emptyset\rangle, \emptyset)$, while the information tuple $\langle N_{k,l}, V_{k,l}\rangle$ in each root node is initialized to 0 for $N_{k,l}(b_0)$ and the highest expected immediate reward over all possible actions for $V_{k,l}(b_0)$. Without loss of generality, we will assume the level $l \in [0, L]$ tree $\mathcal{T}_{k,l}$ corresponds to agent k and use $-k$ to denote the other agent.

The I-NTMCP search algorithm is show in Algorithm 1. At each step t, each tree in the hierarchy is expanded for some fixed number of simulations M, starting with the

lowest level tree $\mathcal{T}_{k,0}$ and working up to the top level tree $\mathcal{T}_{i,L}$. Each simulation begins from a root node corresponding to a time t history, h_k^t, of the agent at the level being expanded. For the top level tree $\mathcal{T}_{i,L}$ this will be the actual history observed from the environment by agent i. Trees at lower levels $l < L$ may have many possible time t histories, since the nodes of lower level trees represent the possible histories of the agent at that level, which are not directly observable by the agent one level above (see Figure 3.2 for an illustration). I-NTMCP uses top-down recursive sampling to ensure that during expansion the number of simulations assigned to a given time t node in a tree is proportional to how likely the history associated with that node is within the time t beliefs of the tree one level above. In this way planning computation is directed to provide better estimates for the lower level beliefs that are more likely within the higher level beliefs.

For tree $\mathcal{T}_{k,l}$ at level l each simulation starts by sampling a history-state from the root node of the level L tree corresponding to the true observed history of the top-level agent i: $w_L^t \sim b_{i,L}(\cdot, h_i^t)$. The history contained in w_L^t is then used to sample a history-state from the level below, $w_{L-1}^t \sim b_{j,L-1}(\cdot, w_L^t.h_j^t)$. This recursion continues until level l, at which point a single simulation is run using MCTS starting from the belief node corresponding to the level l history in the sampled level $l+1$ history-state: $h_k = w_{l+1}^t.h_k^t$. Each nested simulation at level l, starts by sampling a history-state from the root of the tree, $w \sim b_{k,l}(\cdot, h_k)$, and then evolves in two stages.

The first stage involves traversing $\mathcal{T}_{k,l}$ based on the UCT [KS06]algorithm until a leaf node is reached. Suppose the simulation starts from the sampled history-state $w^t \in \mathcal{W}^t$. Similar to UCT, I-NTMCP needs to select an action to progress the Monte Carlo simulation. However, unlike a typical UCT algorithm, I-NTMCP needs to use a joint action $\vec{a} \in \vec{\mathcal{A}}$ to perform this simulation, which requires it to infer the action of the other agent. Therefore, we separate action selection into two types (Algorithm 2). For the agent represented by the tree being expanded, agent k in this case, actions are selected using UCB1 [ACF02] as is typical with UCT, $a_k \leftarrow \arg\max_{a \in \mathcal{A}_k} V_{k,l}(h_k^t a) + c\sqrt{\frac{log N_{k,l}(h_k^t)}{N_{k,l}(h_k^t a)}}$. For the other agent, $-k$, if $l > 0$ actions are sampled using the level $l-1$ belief tree, $\mathcal{T}_{-k,l-1}$, otherwise if $l = 0$ the fixed level 0 policy, $\pi_{-k,0}$ is used. Actions are selected using the history for agent $-k$ contained in the sampled history-state, $w^t.h_{-k}^t$. When selecting actions from $\mathcal{T}_{-k,l-1}$ actions are sampled for the given h_k^t using a softmax function, $Pr(a_{-k}|h_{-k}^t) = \eta \exp \frac{N_{-k,l-1}(h_{-k}^t a_{-k})}{\sqrt{N_{-k,l-1}(h_{-k}^t)}}$, where η is a normalizing constant. The softmax policy accounts for $\mathcal{T}_{-k,l-1}$ being an approximation of the true level $l-1$ policy and its accuracy being dependent on the visit count of the node. Once both actions have been selected, the joint action, $\vec{a}^{t+1} = \langle a_k^{t+1}, a_{-k}^{t+1} \rangle$, is used along with the state in the sampled history-state, $w^t.s^t$, to generate the next state, joint observation, and joint reward, $\langle s^{t+1}, \vec{o}^{t+1}, \vec{r}^{t+1} \rangle \sim \mathcal{G}(w^t.s^t, \vec{a}^{t+1})$. All this information together provides the

Algorithm 1 I-NTMCP Search

Require: Generative model $\mathcal{G}$, Discount γ, Search depth threshold ϵ
Require: Nested Trees $\mathcal{T}_{i,L}, \mathcal{T}_{j,L-1}, ..., \mathcal{T}_{k,0}$
Require: Number of simulations per level M

procedure SEARCH(h_i) ▷ h_1 here is the real history for agent-i at level L
  for $d \leftarrow 0, ..., L$ **do**
    for $1, ..., M$ **do**
      NESTEDSIMULATION(h_i, L, d)
    end for
  end for
  return $\arg\max_{a_i \in \mathcal{A}_i} V_{i,L}(h_i a_i)$
end procedure

procedure NESTEDSIMULATION(h_k, l, $depth$)
  $w \sim b_{k,l}(\cdot, h_k)$ ▷ $w = \langle s, \langle h_k, h_{-k} \rangle \rangle$
  if $l = depth$ **then**
    SIMULATE(w, h_k, l, 0)
  else
    NESTEDSIMULATION($w.h_{-k}$, $l-1$, $depth$)
  end if
end procedure

procedure SIMULATE(w, h_k, l, t)
  if $\gamma^t < \epsilon$ **then**
    return 0
  end if
  if $h_k \notin \mathcal{T}_{k,l}$ **then**
    for all $a_k \in \mathcal{A}_k$ **do**
      $\mathcal{T}_{k,l}(h_k a_k) \leftarrow \langle N_{init}(h_k a_k), V_{init}(h_k a_k), \emptyset \rangle$
    end for
    return ROLLOUT($w.s$, h_k, t)
  end if
  $\vec{a} \leftarrow$ SELECTACTION(w, h_k, l)
  $\langle s', \vec{o}, \vec{r} \rangle \sim \mathcal{G}(w.s, \vec{a})$
  $w' \leftarrow \langle s', w.\vec{h}\vec{a}\vec{o} \rangle$
  $G_k \leftarrow r_k + \gamma$ SIMULATE($w', h_k a_k o_k, t+1$)
  $b_{k,l}(h_k a_k o_k) \leftarrow b_{k,l}(h_k a_k o_k) \cup \{w'\}$
  $N_{k,l}(h_k a_k) \leftarrow N_{k,l}(h_k a_k) + 1$
  $N_{k,l}(h_k a_k o_k) \leftarrow N_{k,l}(h_k a_k o_k) + 1$
  $V_{k,l}(h_k a_k) \leftarrow V_{k,l}(h_k a_k) + \frac{G_k - V_{k,l}(h_k a_k)}{N_{k,l}(h_k a_k)}$
  return G_k
end procedure

Algorithm 2 I-NTMCP Select Action

Require: Nested Trees $\mathcal{T}_L, \mathcal{T}_{L-1}, ..., \mathcal{T}_1, \mathcal{T}_0$

procedure SELECTACTION(w, h_k, l)
  $a_k \leftarrow \arg\max_{a \in \mathcal{A}_k} V_{k,l}(h_k a) + c\sqrt{\frac{log N_{k,l}(h_k)}{N_{k,l}(h_k a)}}$
  if $l = 0$ **then**
    $a_{-k} \sim \bar{\pi}_{-k,0}(\cdot|w.h_{-k})$
  else
    $a_{-k} \sim \mathcal{T}_{-k,l-1}(w.h_{-k})$
  end if
  return $\langle a_k, a_{-k} \rangle$
end procedure

next history-state, $w^{t+1} = \langle s^{t+1}, w^t.\vec{h}^t\vec{a}^{t+1}\vec{o}^{t+1} \rangle$, which is added as a particle to the next belief $b_{k,l}(h_k^t a_k^{t+1} o_k^{t+1})$ as part of the Monte Carlo belief update process. The first stage continues until a leaf node in the search tree $\mathcal{T}_{k,l}$ is reached.

In the second stage, a rollout is used to estimate the value of the leaf node. Rollouts are performed in a similar manner to Monte Carlo planning for POMDPs, with the exception that actions are selected for both agents. Starting from the sampled history-state at the leaf node w, a sequence of history-states is generated using rollout policies for each agent, until a terminal state or discount horizon is reached. The value of the leaf node is estimated by the rollout return G_k. During the rollout, actions for agent i and agent j are selected using history-based rollout policies, $\pi_{i,rollout}(a_i|h_i)$ and $\pi_{j,rollout}(a_j|h_j)$ - e.g. uniform random action selection or using domain knowledge. Following the rollout, the generated return G_k is used to update the value estimates of all the parents of the new leaf node. After each simulation, exactly one new node is added to the tree $\mathcal{T}_{k,l}$ which corresponds to the first new history encountered during that simulation.

3.4.2 *Belief Update*

After agent i at nesting-level L performs action $a_{i,L}^{t-1}$ and receives observation $o_{i,L}^t$, the root beliefs of each search tree are updated. Unlike tree construction, the process of updating beliefs is done top-down starting at $\mathcal{T}_{i,L}$. Specifically, the tree $\mathcal{T}_{k,l}$ at each level is updated using a distribution over possible histories for the agent at that level. Let $X_{k,l}^t$ denote a discrete distribution over the set of possible histories for the level l agent $\mathcal{H}_k^t$. At the top level L, the exact history for the agent is known since it is the actual history for the acting agent. Hence, $X_{i,L}^t$ is the discrete distribution with probability 1 for the true history of agent i and probability 0 for all other time t histories. Using

$X^t_{i,L}$ the distribution over possible histories of the agent at the level below, $X^t_{j,L-1}$, is then calculated. More formally, given the history distribution $X^t_{k,l}$ for agent i at level $l > 0$, the probability of a given history $h^t_j \in \mathcal{H}^t_j$ for agent j at level $l-1$, denoted as $X^t_{j,l-1}(h^t_j)$ is given by:

$$X^t_{j,l-1}(h^t_j) = \sum_{h^t_i \in \mathcal{H}^t_i} \frac{X^t_{i,l}(h^t_i)}{|b^t_{i,l}(h^t_i)|} \sum_{w^t \in b^t_{i,l}(h^t_i)} \delta_K(w^t.h^t_j, h^t_j) \tag{3.1}$$

Where δ_K is the Kronecker delta function which is 1 when $w^t.h^t_j = h^t_j$ and 0 otherwise. $|b^t_{i,l}(h^t_i)|$ denotes the number of particles in the belief node of the level l tree for history h^t_i. Intuitively, $X^t_{j,l-1}$ specifies which time t belief nodes to keep and update in the tree $\mathcal{T}_{j,l-1}$.

Note that in eq. 3.1, $X^t_{j,l-1}(h^t_j)$ is calculated by iterating over the set of all time t histories for both agents. In practice the number of histories used is not directly dependent on the size of the time t history sets, but instead depends on the number of particles stored in the trees which in turn is a function of the number of simulations being used for planning and the observation branching factor of the environment. This means that even as the number of possible histories grows with the planning horizon, the update time remains relatively constant. Of course this speed comes at the cost of accuracy, but one of the big advantages of MCTS based methods is that the compute resources will naturally be used to explore the most likely histories in expectation.

To reduce memory usage and improve belief accuracy, I-NTMCP also employs a pruning and reinvigoration step after each update. Similar to MCTS based POMDP planners, the pruning step removes unreachable branches from each tree. In I-NTMCP this corresponds to removing any time t histories in the tree that have a zero probability in the update history distribution $X^t_{i,l}$. For the top level L tree this means removing all time t branches (and their sub-trees) except the one that corresponds to the true action and observation received by the top level agent at time t. The reinvigoration function adds additional particles to each remaining time t belief node in the tree in order to combat particle depletion. For our experiments we added a fixed number of particles K across the time t belief nodes in each tree. The number of additional particles added to a given belief node is weighted by the probability of that belief node's history. For a given history h^t_i the number of additional particles was equal to $KX^t_{i,l}(h^t_i)$. In this way only a constant number of particles were added each update and the particles were distributed based on belief node probability.

3.5 THEORETICAL ANALYSES

In the following, we show that given the true belief over the space of history-states for an agent, $b_i(w, h_i)$, and the fixed history-based policy for the other agent, π_j, I-NTMCP converges to the optimal value function. This convergence holds for finite horizon I-POMDPs but can be extended to the infinite horizon case [KS06]. The proof evolves in two stages. First, we show how an I-POMDP can be converted into to a POMDP using history-states. Second, we apply existing results for the UCT algorithm in POMDPs to show convergence to the optimal value function in the limit.

Firstly, from the original work on I-POMDPs we known that an I-POMDP can be converted into a POMDP with an expanded state space [GD05, Section 6]. Specifically, given the policy of all other agents at level $l-1$, the I-POMDP for the agent with nesting level l can be converted into a POMDP where each state includes the environment state and the belief of the other agent. Noting, that in the case of I-NTMCP the more general definition of an I-POMDP applies [GD05, Section 5], where the model of the other agent is a triple $m_j = \langle h_j, \pi_j, \mathcal{O}_j \rangle$, containing the history, history-based policy, and observation space of the other agent.

Next we provide the main proof of convergence.

Theorem 3.5.1. *Given the true belief state $b_i(w, h_i)$ and a fixed history based policy for the other agent π_j, for a suitable choice of c, the value function constructed for agent i by I-NTMCP converges to the optimal value function, $V(h_i) \xrightarrow{p} V^*(h_i)$, for all agent histories h_i that are prefixed by h_i^t. As the number of visits $N(h_i)$ approaches infinity, the bias of the value function, $\mathbb{E}\left[V(h_i) - V^*(h_i)\right]$ is $O(\log N(h_i)/N(h_i))$.*

Proof. Using the results from [GD05], given a fixed history based policy for the other agents an I-POMDP can be converted to a POMDP. Thus I-NTMCP, given the true belief state $b_i(w, h_i)$, is equivalent to PO-UCT applied to the converted POMDP and so the proof from [SV10, Theorem 1] applies, showing convergence in probability to the optimal value function of the POMDP for PO-UCT in the limit. □

It is worth pointing out that while this theoretical equivalence exists, it cannot be exploited in practice for solving I-POMDPs with standard POMDP solvers. The transformed POMDP has a state space that grows exponentially with the nesting level, making it computationally intractable for all but the simplest problems. Furthermore, the transformation requires explicitly enumerating all possible models and beliefs of the other agent, which is infeasible for continuous belief spaces. This is why specialized algorithms like I-NTMCP that avoid explicit transformation are necessary.

3.6 EXPERIMENTS

We evaluate I-NTMCP on two competitive environments. The aim of these experiments is two-fold. First, to evaluate the effectiveness of I-NTMCP for planning with nested-reasoning in the multi-agent, partially observable setting. Second, to evaluate the scalability of I-NTMCP in terms of nesting level and problem complexity.

3.6.1 *Experimental Setup*

As an online solver, I-NTMCP interleaves planning and execution. The planning time reported for I-NTMCP was based on the number of simulations used per execution step, which was set to be constant for a given experiment and agent across nesting levels. For example, an experiment using M simulations for I-NTMCP with nesting level L, would use M simulations at all nesting levels $l \in [0, L]$, resulting in a total of $(L+1)M$ simulations for each execution step. $K = M/16$ additional particles were added for each tree during belief reinvigoration after each execution step.

We set the UCT exploration constant c to be $c = G_{hi} - G_{lo}$, where G_{hi} was the highest return achieved during sample runs of I-NTMCP with nesting level 0 and $c = 0$ against a random opponent, and G_{lo} was the lowest return achieved by a random agent against a random opponent [SV10]. We used $\epsilon = 0.1$ with a discount $\gamma = 0.95$ giving a maximum search depth of 45 steps. For all experiments, we used a uniform random policy for the subintentional policy of the other agent at level 0, $\pi_{-k,0}$. It is worth noting that it would be easy to incorporate additional prior knowledge by replacing this with a more intelligent policy.

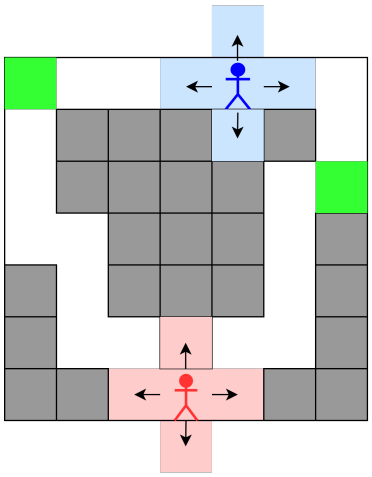
I-NTMCP was implemented using Python [VD95] and all code is available at `https://github.com/RDLLab/i-ntmcp`.

3.6.2 *Scenarios*

For our experiments we used two competitive grid-world domains: *Runner-Chaser* and *Pursuit-Evasion*.

3.6.2.1 *Runner-Chaser*

Figure 3.3 illustrates the 7x7 *Runner-Chaser* scenario. The runner (red) must reach one of the green goal squares while avoiding capture by the chaser (blue). Each agent knows the environment exactly, including the start and goal locations of both agents. Agents have four deterministic actions corresponding to moving one cell in each of the four cardinal directions. They observe perfectly the four adjacent cells,



l	Runner	Chaser
0	right	right
1	left	right
2	left	left
3	right	left
4	right	right

Figure 3.3: The 7x7 *Runner-Chaser* problem (left) and the optimal finite-nested reasoning path choice for each agent based on nesting level l (right).

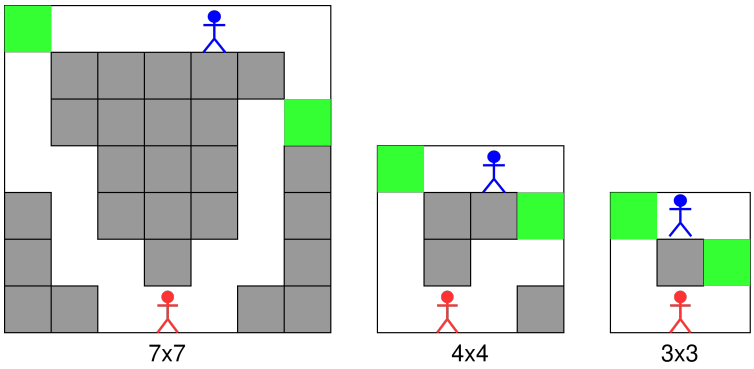


Figure 3.4: All *Runner-Chaser* domain layouts.

specifically whether each cell is empty, has a wall, or contains the opponent. The runner is considered caught if the chaser observes the runner, resulting in a reward of -100 for the runner. Conversely, if the runner reaches a goal it receives a reward of $+100$. Both agents receive a -1 reward per action and the chaser receives the opposite terminal rewards to the runner. An episode ends if the runner reaches a goal, is caught, or has moved over 20 steps (which is considered a draw). Three problem sizes were used: 3x3, 4x4, and 7x7 (shown in Figure 3.4).

This environment requires agents to reason about the sequential strategy of the opponent given no information. The map is designed so that the chaser cannot guard both goals and therefore must reason about which goal the runner will target. Meanwhile, the runner must reason about which goal the chaser will guard. Predicting the strategy of the opponent requires modeling the opponent's sequential decision making problem, making the Runner-Chaser problem ideal for validating the correctness of algorithms for solving I-POMDPs.

The optimal policy under the finite-nested reasoning assumption can be derived from the properties of the Runner-Chaser environment. Specifically, at nesting level

$l = 0$ assuming a uniform random opponent, each agent's optimal policy is to take the shortest path to their respective goal. This is because each agent does not observe their opponent during each episode (except in the terminal state) and against a random opponent the shortest path will maximize expected return by reducing the number of actions and the chance of random interception/goal-reaching by the random opponent. In all maps the right hand path is the shortest path for both agents. The level $l = 1$ policy for each agent is then to select the counter path to a level $l = 0$ opponent, and so on for level $l = 2, ...$ The table in figure 3.3 shows the optimal nested reasoning policies up to $l = 4$. Notice that the nested reasoning policy space is small and can be extrapolated to any reasoning level. This makes it a perfect environment for evaluating that I-NTMCP is in fact able to perform nested reasoning effectively.

3.6.2.2 *Pursuit-Evasion*

Inspired by [SvW18], *Pursuit-Evasion* adds significant additional complexities to the Runner-Chaser problem. It also involves an *evader* (red) trying to reach a goal location before it is caught by a *pursuer* (blue). However, unlike the Runner-Chaser problem the goal location of the evader is not known to the chaser. This is analogous to the chaser having to reason about different I-POMDP models for the runner. Furthermore, the map is larger and more complex with more path options. Each agent is aware of the start location of their opponent, while only the evader knows its goal location. The pursuer only knows the possible goal locations for the evader. Lastly, the observation space of both agents is expanded. Each agent receives six bits per step. Four bits indicate whether there is a wall or not in each of the cardinal directions, one bit indicates whether the opponent can be seen in the agent's field of vision, and the final bit indicates whether the opponent can be heard within Manhattan distance 2 of the agent. Due to the lack of precision of these observations, the pursuer never knows the exact position of the evader and vice versa. The action space, rewards, and terminal conditions are the same as Runner-Chaser except the step limit is increased to 40. The transition and state space are also the same with the addition of the direction each agent is facing and the goal location for the evader. This problem has 88,752 states, 4 actions and 64 observations per agent, more than two orders of magnitude larger than prior general offline I-POMDP benchmarks [HL13; PG17].

Knowledge of the shortest path to the goal was incorporated into the evader agent using preferred actions and the rollout policy. Specifically, at each step the evader preferred to move in the direction of the shortest path to the goal and avoid actions that led them to remaining in the same spot or move in the direction they came from. During tree expansions the tree was initialized to $V_{init}(h_i a_i) = G_{hi}, N_{init}(h_i a_i) = 10$ for actions along the shortest path, and $V_{init}(h_i a_i) = G_{lo}, N_{init}(h_i a_i) = 0$ for actions that

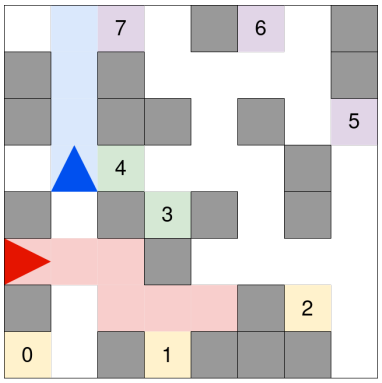


Figure 3.5: The 8x8 *Pursuit-Evasion* problem. Cells 3 and 4 are possible start locations of the *Pursuer* (blue). Cells 0-2 & 5-7 are possible start and goal locations for the *Evader* (red). The evader's start and goal locations are always on opposite sides of the map, shown by the different colors. The field of vision for both agents expands in a cone in the direction the agent is facing as shown by the blue and red colored cells.

lead to the agents previous location, or to the agent remaining stationary. For other actions the initial values were set between these two extremes, weighted by how far away from the shortest path they were. This weighting was done to mitigate myopic shortest path behavior.

3.6.3 *Results*

In this section we present our experiments and results demonstrating the scalability and convergence properties of I-NTMCP.

3.6.3.1 *Scalability*

To evaluate the scaling performance of I-NTMCP, we compare it with I-POMDP Lite [HL13] in the Runner-Chaser domain. I-POMDP Lite is considered one of the fastest offline I-POMDP solvers. For our experiments we used the original program as provided by the authors. We ran both I-POMDP Lite and I-NTMCP using nesting level 1 to compute the policy for the Runner, against random and POMDP Chasers. The POMDP Chaser models the runner's actions as a uniform random policy.

Table 3.1 indicates that as the problem size increases, I-NTMCP scales significantly better than I-POMDP Lite. Even if we aggregate the planning time for I-NTMCP over the entire planning horizon (20 steps), the total planning time of I-NTMCP is over 50× less than I-POMDP Lite for the 7x7 scenario, while generating a substantially better policy.

Moreover, in all but one scenario, I-NTMCP computes an equal or better policy than I-POMDP Lite. The only result where I-NTMCP performs worse than I-POMDP Lite is on the 4x4 problem against a Random Runner. The reason is that unlike I-NTMCP,

Chaser	I-POMDP Lite $l = 1$			I-NTMCP $l = 1$		
	H	Time	Return	Sims.	Time	Return
3x3, $\lvert\mathcal{S}\rvert = 128$						
Random	2	101	94.00 ± 0.00	1024	0.41 ± 0.00	94.00 ± 0.00
POMDP	2	101	94.00 ± 0.00	1024	0.45 ± 0.00	94.00 ± 0.00
4x4, $\lvert\mathcal{S}\rvert = 288$						
Random	4	2524	$\mathbf{82.90 \pm 0.79}$	1024	0.67 ± 0.00	52.56 ± 3.70
POMDP	4	2524	63.80 ± 0.00	1024	0.69 ± 0.00	$\mathbf{77.73 \pm 0.01}$
7x7, $\lvert\mathcal{S}\rvert = 1,152$						
Random	1	5791	-12.91 ± 0.11	4096	5.71 ± 0.17	$\mathbf{54.94 \pm 1.55}$
POMDP	1	5791	NA	4096	5.77 ± 0.32	$\mathbf{56.23 \pm 1.31}$

Table 3.1: Comparison of I-NTMCP and I-POMDP Lite using nesting level 1 in the *Runner-Chaser* domain. For I-POMDP Lite, we use the shortest best performing planning horizon (H) completed in 1 hour for the 3x3 and 4x4 problems and 2 hours for the 7x7 problem. The *Time* column shows (in seconds) total planning time for I-POMDP Lite and mean planning time per step ($\pm$ SD) for I-NTMCP. The *Return* column shows the mean discounted returns ($\pm$ 95% CI) using $\gamma = 0.95$. The POMDP opponent was generated using PBVI [PGT06] for I-POMDP Lite and POMCP [SV10] for I-NTMCP. PBVI could not be run for the 7x7 problem due to problem size. The results above are based on 1000 episodes for each problem size and chaser.

Pursuit-Evasion	Random	Shortest Path	I-NTMCP $l = 0$	I-NTMCP $l = 1$
Random	0.31	0.52	0.06	0.08
Shortest Path	0.94	0.72	0.35	0.4
I-NTMCP $l = 0$	**1.00**	0.59	0.4	0.33
I-NTMCP $l = 1$	0.99	0.79	0.53	0.36
I-NTMCP $l = 2$	0.97	**0.80**	**0.68**	**0.53**
I-NTMCP $l = 3$	0.99	0.75	0.56	**0.53**

Table 3.2: Win rate for Pursuer (row) against Evader (column) agents in the *Pursuit-Evasion* domain. Results are from 100 episodes, with I-NTMCP using 2048 simulations.

during execution, I-POMDP Lite has access to the opponents performed action for its belief update. This information allows I-POMDP Lite to know the exact location of the opponent and overcome errors in its opponent modeling introduced by the random opponent. Despite this additional advantage, the results in Table 3.1 indicate that when the problem size becomes larger, I-POMDP Lite's ability to generate a good policy suffers much more than I-NTMCP.

The performance of I-NTMCP in *Pursuit-Evasion* is shown in Table 3.2. Since the problem is too big for I-POMDP Lite, in *Pursuit-Evasion* we compared I-NTMCP with uniform random and shortest path baselines. The shortest path policy selects actions that follow the shortest path to the goal (evader policy) or the evader's start position (pursuer policy). I-NTMCP outperformed the two baselines against each opponent policy. Furthermore, as the nesting level of the opponent increases, deeper nesting levels are required to perform well, as shown by the drop in performance of I-

NTMCP at levels $l \in [0, 1]$ when faced with I-NTMCP opponents. These result indicate I-NTMCP's ability to plan using finite-nested reasoning on a very large I-POMDP.

3.6.3.2 *Convergence*

We used the *Runner-Chaser* scenario to evaluate the convergence of I-NTMCP. In this scenario, the optimal finite-nested reasoning policy (FNR) for both the runner and chaser can be easily derived and provides a stable baseline policy for comparison (Figure 3.3 (right)). We tested I-NTMCP for the Runner policy using an increasing number of simulations against the optimal chaser policy.

The results shown in Figure 3.6 indicate that as planning budget increases, the performance of the I-NTMCP policy resembles the FNR policy. For instance, when the Chaser uses nesting level 0 (i.e., moving to the right), the optimal Runner policy is the FNR level 1 or 2 policy, where the Runner chooses the opposite path to the Chaser. The opposite is true for the Runner at nesting levels 0 and 3, where the optimal FNR policy chooses the same path as the Chaser. The top-left plot in Figure 3.6 indicates that the policies generated by I-NTMCP indeed converge to the expected behavior of the FNR policies against a level 0 Chaser –that is, for nesting level $l \in \{1, 2\}$, the Runner policies generated by I-NTMCP converge to winning values, while for $l = \{0, 3\}$, the policies converge to loosing values. This trend is present for all nesting levels we tested.

Moreover, Figure 3.7 indicates that the planning time for I-NTMCP is linear with nesting level, allowing it to scale well for problems that require deeper levels of nesting.

3.7 DISCUSSION

We presented the I-NTMCP algorithm and demonstrated its ability to scale to significantly larger problems than existing offline I-POMDP solvers. While we believe that this is a promising result towards the practical application of I-POMDPs, some important questions remain.

Firstly, how should the nesting level be chosen? The choice of nesting level can have a significant impact on performance (Figure 3.6). In this and prior work [HL13; SD15; HG19] the nesting level is treated as a hyper parameter chosen by the user. We hypothesize that for many practical problems of interest performance will be robust to the choice of nesting level given it is sufficiently high. For example, in domains where agents only weakly interact. However, ultimately it would be desirable to have the agent infer the nesting level of the other agent online as they interact.

One possible method for this, which I-NTMCP would be amenable to, is using Bayesian Reinforcement Learning (BRL) [Gha+15] where the other agent's nesting level is included as another hidden variable of the state and whose value is learned online

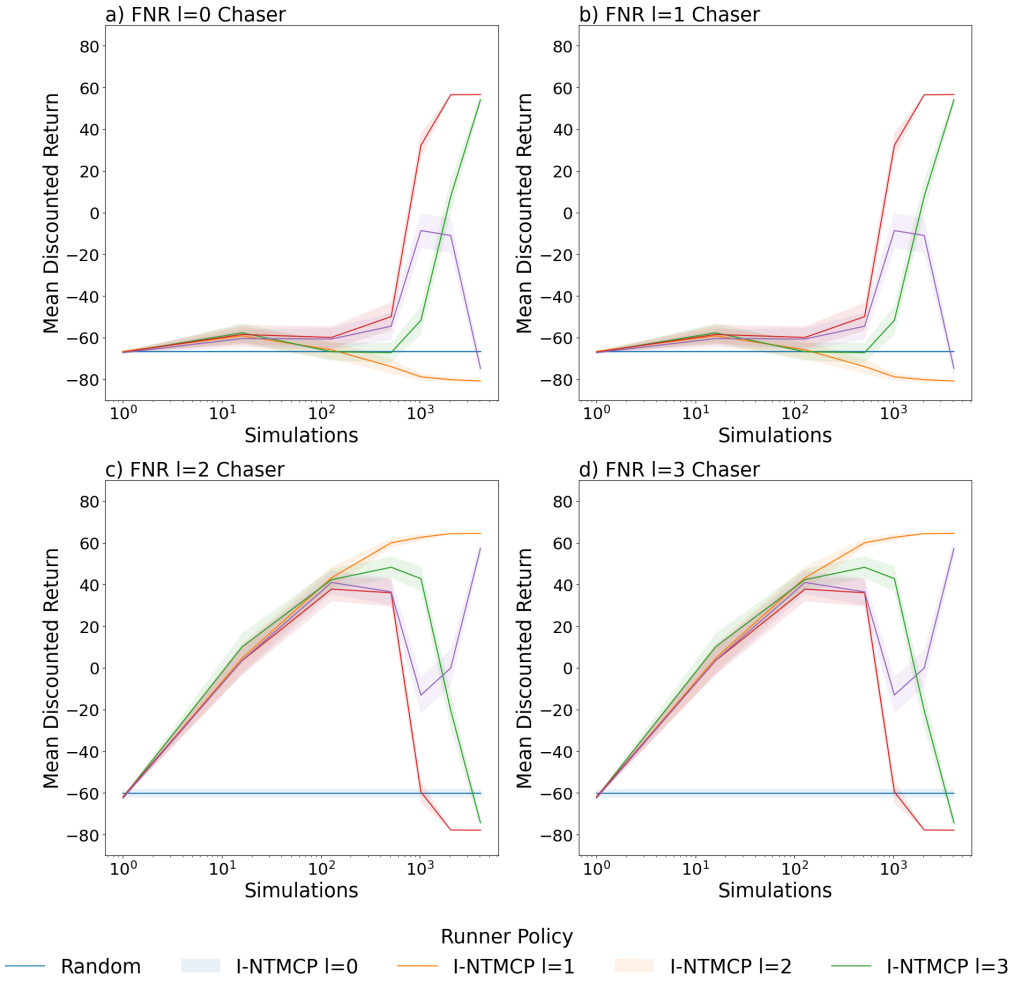


Figure 3.6: Performance of I-NTMCP runner against Finite-Nested Reasoning (FNR) chaser in the 7x7 *Runner-Chaser* problem. Each line shows the mean discounted return (±95% CI) for a random policy and I-NTMCP with different nesting levels l as the number of planning simulations increases. Each figure shows results against a FNR chaser with a different nesting level l.

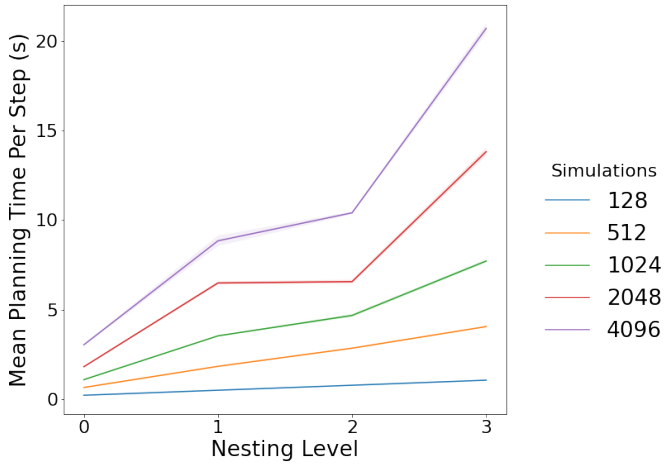


Figure 3.7: I-NTMCP mean planning time per step (±95% CI) for different levels of nesting and simulation number in 7x7 *Runner-Chaser*. Deviations from linear for nesting level two are due to the runner agent's ability to reach terminal states earlier during search, leading to a small difference in mean planning time between runner and chaser agents.

through interaction. Some work has already been done on applying BRL to I-POMDPs [PG16] and similarly for online MCTS based POMDP planning [KOA17].

On potential robotics applications of I-NTMCP. Many applications involve operating in environments containing humans or independent autonomous systems. For example, in autonomous driving, safety and efficiency can be improved by modeling the intentions and internal states of the other drivers [SK22]. I-NTMCP helps alleviate the computational challenges presented by the I-POMDP framework. As such, I-NTMCP has the potential for application in real-world problems with thousands of states and discrete actions and observations. Additionally, it is relatively straight forward to incorporate domain knowledge into I-NTMCP via the rollout policy and belief update function. It should be possible to apply I-NTMCP in suitable real-world applications given a full model of the environment including specifications of the reward functions of the other agents. Some uncertainty over the reward functions of the other agent can be incorporated via the use of additional hidden state variables. We hope to see research in the application of I-NTMCP and I-POMDP solvers, more generally, to real-world scenarios.

3.8 CONCLUSION

In this chapter, we have presented the I-NTMCP algorithm for online planning in multi-agent, partially observable environments. I-NTMCP combines Monte Carlo based planning for POMDPs with the finite-nested reasoning construction of I-POMDPs by constructing and maintaining a sequence of inter-related belief trees. Our experiments indicate that I-NTMCP can plan effectively in significantly larger two-agent I-POMDPs than existing offline solvers. Furthermore, I-NTMCP requires only a generative model of the joint environment dynamics, permitting its application to problems that are difficult to model.

Future work abounds. For instance, although I-NTMCP indicates promising results, the choice of nesting level used is manually selected by the user, similar to [HL13; SD15; HG19]. It would be desirable for the agent to infer the nesting level of the other agent online as they interact. Moreover, efficient methods for further scalability and extension to continuous spaces would also be desirable.

Last but not least, I-POMDP is a general and principled decision-making framework that accounts for various types of uncertainty and other agents' reasoning. We foresee that in scenarios that require close and substantial interactions between autonomous agents and humans, such a principled framework would be required to help ensure safety and natural interactions.

4

SCALABLE TYPE-BASED REASONING USING A META-POLICY

This chapter has been published as:

Jonathon Schwartz, Hanna Kurniawati, and Marcus Hutter. "Combining a Meta-Policy and Monte-Carlo Planning for Scalable Type-Based Reasoning in Partially Observable Environments". In: *arXiv preprint arXiv:2306.06067* (2023). arXiv: 2306.06067

An earlier version was published as:

Jonathon Schwartz and Hanna Kurniawati. "Bayes-Adaptive Monte-Carlo Planning for Type-Based Reasoning in Large Partially Observable, Multi-Agent Environments". In: *Autonomous Agents and Multiagent Systems*. IFAAMAS, 2023, pp. 2355–2357

The design of autonomous agents that can interact effectively with other agents without prior coordination is a core problem in multi-agent systems. Type-based reasoning achieves this by maintaining a belief over a set of potential behaviors for the other agents. However, current methods are limited in that they assume full observability of the state and actions of the other agent or do not scale efficiently to larger problems with longer planning horizons. Addressing these limitations, in this chapter we present *Partially Observable Type-based Meta Monte-Carlo Planning* (POTMMCP) – a online planning method which incorporates a novel meta-policy for guiding search and evaluating beliefs, allowing it to plan effectively to longer horizons. We prove convergence to the optimal solution in the limit and empirically demonstrate that POTMMCP effectively adapts online to diverse sets of other agent behaviors across a range of environments. Comparisons with the state-of-the art method on problems with up to 10^{14} states and 10^{8} observations indicate that POTMMCP is able to compute better solutions significantly faster.

4.1 INTRODUCTION

A core research area in multi-agent systems is the development of autonomous agents that can interact effectively with other agents without prior coordination [BM05; Sto+10; ALS17]. Type-based reasoning methods give agents this ability by maintaining a belief over a set *types* for the other agents [BSK11; AR14; BS15; ACR16]. Each type is a complete specification of the agent's behavior, mapping from the agent's interaction history to a probability distribution over actions. If the set of types is sufficiently representative, type-based reasoning methods can lead to fast adaptation and effective interaction without prior coordination [AR13; BS15].

Unfortunately, type-based reasoning greatly increases the complexity of the planning problem and finding scalable methods remains a key challenge. This is especially true in partially observable settings where the planning agent cannot directly observe the type of the other agent, their interaction history, or the state of the environment. In this setting the agent must maintain a joint belief over these three features leading to a belief space that grows exponentially with the planning horizon and number of agents. Several online planning methods based on Monte Carlo Tree Search (MCTS) have shown promising performance in non-trivial partially observable problems [Eck+20; Kak+22; SZK22]. However, so far these methods have only been demonstrated in settings where the other agent's type is known and scale poorly to domains with longer planning horizons.

Inspired by the success of techniques combining MCTS with a search policy in single-agent [Sch+20] and zero-sum [Sil+18] settings, in this chapter we propose a method for integrating a search policy for type-based reasoning in a partially-observable setting. Using a search policy has a number of advantages. Firstly, it guides exploration, biasing it away from low value actions and allowing effective planning to longer horizons. Secondly, if the search policy has a value function this can be used for evaluating during search. Doing this avoids expensive Monte Carlo (MC) rollouts and can significantly improve search efficiency.

For the search policy we introduce a novel meta-policy using the set of types available to the planning agent. The meta-policy is generated using an empirical game [Wel06] which computes the expected payoffs between each pairing of types using a number of simulated episodes. This makes the meta-policy relatively inexpensive to compute and side-steps the usual method for finding a search policy which is to train one from scratch [Sil+18; Bro+20a; Tim+22; Li+23].

Combining the meta-policy with MCTS, we create a new online planning algorithm, Partially Observable Type-based Meta Monte-Carlo Planning (POTMMCP), designed for type-based reasoning in partially observable environments. Through extensive evaluations and ablations on large competitive, mixed, and cooperative environments -

the largest of which has four agents and on the order of 10^{14} states and 10^{8} observations - we demonstrate empirically that POTMMCP is able to substantially outperform the existing state-of-the-art method [Kak+22] in terms of final performance and planning time. Additionally, we prove the correctness of our approach, showing that POTMMCP converges to the Bayes-optimal policy in the limit.

4.2 RELATED WORK

MONTE CARLO PLANNING We are interested in Monte Carlo planning methods for environments where the agent must adapt to a set of possible types of other agents. When coordination between agents is involved, this is the *ad-hoc teamwork* problem [BM05; Sto+10]. Various approaches to ad-hoc teamwork have been proposed, including those based on stage games [WZC11], Bayesian beliefs [BSK11], the Partially Observable Markov Decision Process (POMDP) [Bar+14], types with parameters [AS17], and for the many agent setting [You+18]. All these methods use MCTS but are limited to environments where the state and actions of the other agents are fully observed. In the *agent modeling* setting [AS18], several MCTS-based methods have been proposed using the Interactive POMDP (I-POMDP) [GD05] framework. Including methods based on finite state-automata [PG17] and nested MCTS [SZK22], as well as methods for open multi-agent systems [Eck+20], and systems with communication [Kak+22]. Other works have focused on planning in strictly cooperative [CO21; Cho+22] or competitive [CPW12] settings. Also related to our work are a number of Bayes-adaptive planning methods using MCTS [GSD13; AO14; KOA17]. However, these methods focus on learning parameters of the environment's transition dynamics, while we focus instead on learning the policy type and history of the other agent.

COMBINING REINFORCEMENT LEARNING AND SEARCH In recent years several works have integrated Reinforcement Learning (RL) with MCTS. Self-play RL and MCTS have been combined in two-player fully observable zero-sum games with a known environment model [Sil+16; Sil+18] and using a learned model [Sch+20]. Similar approaches have been applied to zero-sum imperfect-information games [BS19; Bro+20a], as well as cooperative games where there is prior coordination for decentralized execution [Ler+20]. Our method builds on this line of research, specifically relating to using an existing policy as a prior for search. However, we apply these advances outside of self-play zero-sum games or where there is prior coordination, instead focusing on online adaption to previously unknown other agents. There have also been works looking at combining MCTS with an RL policy for training a best-response to a single known opponent [Tim+22] or a distribution over policies [Li+23].

Compared with these methods, our approach does not rely on any training to generate the search policy from scratch for a given policy set. Instead we propose a way to generate the search policy using the information available to the planning agent in the type-based reasoning setting, namely the types of the other agents. As we show this improves planning performance without requiring additional training, even if the set of types change.

4.3 PROBLEM DESCRIPTION

We consider the problem of *type-based* reasoning in partially observable environments. We model the problem as a Partially Observable Stochastic Game (POSG) [HBZ04] which consists of N agents indexed $\mathcal{I} = \{1, \dots, N\}$, a discrete set of states $\mathcal{S}$, an initial state distribution $b_0 \in \Delta(\mathcal{S})$, the joint action space $\vec{\mathcal{A}} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_N$, the finite set of observations $\mathcal{O}_i$ for each agent $i \in \mathcal{I}$, a state transition function $T : \mathcal{S} \times \vec{\mathcal{A}} \times \mathcal{S} \to [0, 1]$ specifying the probability of transitioning to state s' given joint action $\vec{a}$ was performed in state s, an observation function for each agent $Z_i : \mathcal{S} \times \vec{\mathcal{A}} \times \mathcal{O}_i \to [0, 1]$ specifying the probability that performing joint action $\vec{a}$ in state s results in observation o_i for agent i, and a bounded reward function for each agent $R_i : \mathcal{S} \times \vec{\mathcal{A}} \to \mathbb{R}$. For convenience, we also define the generative model $\mathcal{G}$, which combines T, Z, R, and returns the next state, joint observation, and joint reward, given the current state and joint action $\langle s', \vec{o}, \vec{r} \rangle \sim \mathcal{G}(s, \vec{a})$.

At each step, each agent $i \in \mathcal{I}$ simultaneously performs an action $a_i \in \mathcal{A}_i$ from the current state s and receives an observation $o_i \in \mathcal{O}_i$ and reward $r_i \in \mathbb{R}$. Here, 'simultaneous' means that each agent selects their actions independently without observing the other agents' current action choice. The resulting state transition, observations, and rewards depend on the joint action of all agents. Each agent has no direct access to the environment state or knowledge of the other agent's actions and observations. Instead they must rely only on information in their *interaction-history* up to the current time step t: $h_{i,t} = \langle o_{i,0} a_{i,0} o_{i,1} a_{i,1} \dots a_{i,t-1} o_{i,t} \rangle$[1]. The set of all time t histories for agent i is denoted $\mathcal{H}_{i,t}$. Agents select their next action using their *policy* π_i which is a mapping from their history $h_{i,t}$ to a probability distribution over their actions, where $\pi_i(a_i | h_{i,t})$ denotes the probability of agent i performing action a_i given history $h_{i,t}$.

We assume the other agents are using policies from a known fixed set of policies where each policy corresponds to an agent type. We denote the planning agent by i, and all other agents collectively using $-i$. The set of fixed policies for the other agents is $\Pi_{-i} = \{\pi_{-i,m} | m = 1, \dots, M\}$, where M is the number of policies in the set

1 For clarity the time subscript t is omitted where it is clear from context.

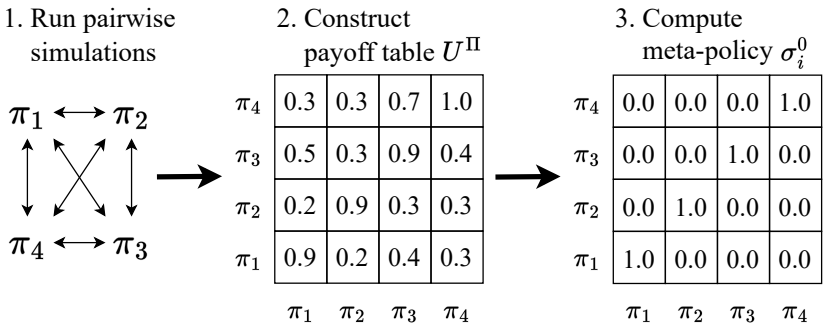


Figure 4.1: Generation process for the greedy meta-policy σ_i^0 generation process for a symmetric environment with two agents.

and $\pi_{-i} = \{\pi_j | j \in \mathcal{I} \setminus \{i\}\}$ is a joint policy that assigns a policy for each non-planning agent. We denote the set of policies available for a specific agent as Π_j for $j \in \mathcal{I}$. Furthermore, we assume the joint-policy used by the other agents is selected based on a known prior distribution ρ, where $\rho(\pi_{-i,m}) = Pr(\pi_{-i,m})$. ρ assigns prior probability to each joint-policy in Π_{-i} which are not permutation-invariant in general. If $N > 2$ and agents are symmetric then there may be multiple equivalent joint-policies, each with a prior probability in ρ.

The goal of the planning agent is to maximize its expected *return* $G_{i,t}$ with respect to ρ within a single episode, $G_{i,t} = \mathbb{E}_{\pi_{-i,m} \sim \rho} \left[\sum_{k=t}^{\infty} \gamma^{k-t} r_{i,k} | \pi_{-i,m} \right]$, where $\gamma \in [0, 1)$ is the discount. The Bayes-optimal policy is the policy that achieves the maximum possible expected return.

4.4 METHOD

Here we present POTMMCP, an online MCTS-based planning algorithm for type-based reasoning in partially observable environments. Like existing planners [Eck+20; Kak+22; SZK22], POTMMCP uses MCTS to calculate the planning agent's best action from its current belief b_i. However, it offers several improvements over existing algorithms. Firstly, it incorporates the PUCT algorithm [Sil+18] for selecting actions during search. PUCT biases search towards the most relevant actions according a *search policy*. This makes it possible to plan for longer horizons, as well as offers improved integration of value functions for leaf node evaluation. To address the limitation of PUCT, namely that it relies on access to a good search-policy, the second improvement offered by POTMMCP is the use of a novel meta-policy as the search-policy. The meta-policy has the advantage that it can be efficiently generated from the policy set Π, and offers a robust prior since it considers performance across the entire set of other agent policies.

4.4.1 *Meta-Policy*

In this work, a meta-policy σ_i is a function mapping a joint policy to a mixture over individual policies. For our purposes it is a mapping from the set of other agent joint policies to a distribution over the set of valid policies for the planning agent $\sigma_i : \Pi_{-i} \to \Delta(\Pi_i)$, so that $\sigma_i(\pi_{i,k}|\pi_{-i,m}) = Pr(\pi_{i,k}|\pi_{-i,m})$ for $\pi_{i,k} \in \Pi_i, \pi_{-i,m} \in \Pi_{-i}$. In symmetric environments, the set of valid policies for the planning agent i is the set of all individual policies for any of the other agents $\Pi_{-i} = \bigcup_{j \neq i} \Pi_j$. In asymmetric environments, it may be necessary to have access to a distinct set of policies for agent i. In practice, these could be any policies used to train the other agent policies in Π_{-i} or could be a separate set of policies generated from data or heuristics.

Ideally, the meta-policy would map from the planning agent's belief to a mixture over policies. However, if we think of each policy as an action we can see that finding a mapping from beliefs to a mixture over policies has the same challenges as finding a mapping from beliefs to the primitive actions of the underlying POSG. Instead we propose a meta-policy σ_i as described that is efficient to compute and which can then be used to improve planning over primitive actions. This meta-policy has the added advantage that it is relatively inexpensive to adapt if Π changes, for example if a new type is added between episodes.

The main idea is to generate the meta-policy using an empirical game constructed from the policies in $\Pi_{-i} \cup \Pi_i$. An **empirical game**, much smaller in size than the full game, is a normal-form game where the actions are policies and the expected returns for each joint policy are estimated from sample games [Wal+02; Wel06; Lan+17]. Formally, an empirical game is a tuple $\langle \Pi, U^{\Pi}, N \rangle$ where N is the number of players, $\Pi = \langle \Pi_1, \ldots, \Pi_N \rangle = \Pi_{-i} \cup \Pi_i$ is the set of policies for all players, and $U^{\Pi} : \Pi \to \mathbb{R}^N$ is a payoff table of expected returns (averaged over multiple games), with $U_i^{\Pi}(\pi_i, \pi_{-i}) \in \mathbb{R}$ denoting the payoff for player i when using policy π_i against the other agents using joint policy π_{-i} (an example is shown in Figure 4.1). If agents are symmetric then we take the average over permutations of the same joint policy (joint policies with same individual policies but assigned to different agents).

Empirical games have the advantage that they can be very efficient to compute, since they require only a finite number of simulations and each simulation can be very fast to run depending on the representation of the policies and the environment. For example, each simulation takes less than a second for our environments with policies represented as neural networks. For slow-to-query policies, such as those that use search, the process will be slower but the process of computing the payoffs is trivially parallelizable and accuracy can be traded-off with time by changing the number of simulations used. It may also be possible to approximate slow-to-query policies with a fast policy by training a function approximator using imitation learning as suggested

by [Tim+22]. Furthermore, when a new policy is added to the set Π, only the payoffs for that policy need to be generated, which requires only $|\Pi|$ comparisons, making it relatively inexpensive to add a new policy.

The next question is, given the empirical-game payoffs U^{Π} how should we define the meta-policy. Since the meta-policy will ultimately be used to guide search for the planning agent, we would like it to select the policy from the set Π_i that maximizes performance of the planning agent. One possible method is to select the policy $\pi_i \in \Pi_i$ that has the highest payoff against a given other agent policy π_{-i} according to U^{Π}. However, it is possible that the best response policy from the set Π_i may change during an episode depending on the planning agent's current belief, while U^{Π} is defined based on the expected performance from the start of an episode. This means that a meta-policy that only selects the maximizing policy from the set Π_i with respect to the payoff table U^{Π} has the potential to select a sub-optimal policy with respect to the planning agent's current belief.

To protect against the potential pitfall's of a myopic meta-policy, we propose a flexible meta-policy based on the softmax function. Specifically, we define the softmax meta-policy σ_i^{τ} where, $\sigma_i^{\tau}(\pi_i|\pi_{-i}) = \frac{1}{\eta}\exp[\frac{1}{\tau}U^{\Pi}(\pi_i, \pi_{-i})]$ with normalizing constant $\eta = \sum_{\pi_i}\exp[\frac{1}{\tau}U^{\Pi}(\pi_i, \pi_{-i})]$ and temperature hyperparameter τ which controls how uniform or greedy the policy is. As $\tau \to 0$ the meta-policy becomes greedier, with σ_i^0 being the greedy policy that selects $\pi_i \in \Pi_i$ that maximizes the empirical payoff U^{Π}. Conversely, as $\tau \to \infty$ becomes more uniform, with $\sigma_i^{\infty}(\pi_i|\pi_{-i}) = 1/|\Pi_i|$ being the uniform meta-policy.

The process of computing a meta-policy is relatively straightforward and needs only be done once for a given set of policies Π. The high-level process for computing a greedy meta-policy σ_i^0 is shown in Figure 4.1. Firstly, each joint policy in the set Π is simulated, for some number of episodes (we use 1000 in our experiments). Next, the average payoff for each pair of policies is used to construct the empirical game payoff table U^{Π}. Finally, the meta-policy is computed according to its equation using the payoff table U^{Π}. If a policy is added or removed from the set Π (between episodes) only the entries for this policy need to be added/removed, before the meta-policy can be computed again. This makes it fairly easy to use with different variations of policy sets, which can be useful for rapid experimentation or for making quick adjustments depending on the settings where it is being used.

4.4.2 *POTMMCP*

We now present POTMMCP, which consists of agent beliefs over history-policy-states, and combining the meta-policy with MCTS for selecting actions from each belief.

Importantly, during planning POTMMCP does not utilize knowledge of the true policy of the other agent in each episode. Rather POTMMCP maintains a joint belief over the environment state, the possible policies of the other agent, as well as the other agent's history. The meta-policy then computes the search-policy from each belief, conditioned on that belief's distribution over the other agent's policy.

4.4.2.1 *Beliefs over history-policy-states*

To correctly model the environment and other agents, POTMMCP maintains a belief over the other agents' histories $h_{-i} \in \mathcal{H}_{-i}$, their policies $\pi_{-i} \in \Pi_{-i}$, and the environment state $s \in \mathcal{S}$. Each belief is thus a distribution over $w = \langle s, \pi_{-i}, h_{-i} \rangle$ tuples, which we refer to as *history-policy-states*. For convenience we denote tuple components using dot notation: $w.s$, $w.\pi_{-i}$, $w.h_{-i}$. We denote the space of history-policy-states with $\mathcal{W}$. Using this notation, the planning agent's belief is a distribution over history-policy-states, where $b_i(w|h_i) = Pr(w_t = w|h_{i,t} = h_i)$.

Defining the belief using history-policy-states transforms the original POSG problem into a POMDP for the planning agent where the other agents' histories and policies, and the environment state are learned online. This conversion is analogous to the one employed by the more general I-POMDP framework [GD05]. However, unlike the I-POMDP, we only consider the possible policies of the other agents, and use their history to represent their internal state, rather than an explicit belief. We show this equivalence in greater detail in Section 4.5.

4.4.2.2 *Meta-policy MCTS with history-policy-states*

POTMMCP extends the POMCP [SV10] algorithm to planning with beliefs over history-policy-states. Being an online planner, each step POTMMCP executes a search to find the next action, followed by particle filtering to update the agent's belief given the most recent observation. To do this POTMMCP builds a search tree $\mathcal{T}$ of agent histories using the PUCT algorithm [Ros11; Sil+18] and a meta-policy as the search policy. Each node of the tree corresponds to a history, where $\mathcal{T}(h_i)$ denotes the node for history h_i, and maintains an approximate belief over history-policy-states $\hat{b}_i(h_i)$ represented using a set of unweighted particles where each particle corresponds to a sample history-policy-state w. For each action $a_i \in \mathcal{A}_i$ from h_i there is an edge $h_i a_i$ that stores a set of statistics $\langle N(h_i a_i), P(a_i|h_i), W(h_i a_i), Q(h_i a_i) \rangle$, where $N(h_i a_i)$ is the visit count, $P(a_i|h_i)$ is the prior probability of selecting a_i given h_i, $W(h_i a_i)$ is the total action-value, and $Q(h_i a_i)$ is the mean action-value.

For each real step at time t in the environment, POTMMCP constructs the tree $\mathcal{T}$ rooted at the agent's current history $h_{i,t}$ via a series of simulated episodes. Each

Algorithm 3 POTMMCP Search

procedure SEARCH(h_i)
  while search time limit not reached **do**
    $w \sim \hat{b}_i(\cdot, h_i)$
    $\pi_i \sim \sigma_i(w.\pi_{-i})$
    SIMULATE(w, h_i, π_i, 0)
  end while
  return $\arg\max_{a_i \in \mathcal{A}_i} N(h_i a_i)$
end procedure
procedure SIMULATE(w, h_i, π_i, $depth$)
  if $\gamma^{depth} < \epsilon$ **then**
    return 0
  end if
  if $h_i \notin T$ **then**
    return EXPAND(h_i, π_i)
  end if
  $a_i \leftarrow \text{PUCT}(h_i)$
  $a_{-i} \sim w.\pi_{-i}(\cdot|w.h_{-i})$
  $\langle s', \vec{o}, \vec{r} \rangle \sim \mathcal{G}(w.s, \langle a_i, a_{-i} \rangle)$
  $w' \leftarrow \langle s', w.h_{-i} a_{-i} o_{-i}, w.\pi_{-i} \rangle$
  $G_i \leftarrow r_i + \gamma$ SIMULATE($w', h_i a_i o_i, \pi_i, depth + 1$)
  $\hat{b}_i(h_i a_i o_i) \leftarrow \hat{b}_i(h_i a_i o_i) \cup \{w'\}$
  $N(h_i a_i) \leftarrow N(h_i a_i) + 1$
  $W(h_i a_i) \leftarrow W(h_i a_i) + G_i$
  $Q(h_i a_i) \leftarrow \frac{W(h_i a_i)}{N(h_i a_i)}$
  for $\hat{a}_i \in \mathcal{A}_i$ **do**
    $P(\hat{a}_i|h_i) \leftarrow P(\hat{a}_i|h_i) + \frac{\pi_i(\hat{a}_i|h_i) - P(\hat{a}_i|h_i)}{N(h_i)}$
  end for
  return G_i
end procedure

simulation starts from a history-policy-state sampled from the root belief $\hat{b}_i(h_{i,t})$ and proceeds in three stages. The search procedure is shown in Algorithm 3.

In the first stage, until a leaf node is reached, actions for the planning agent i are selected using the PUCT algorithm (Algorithm 4), while actions for the other agent $-i$ are sampled using their policy and history contained within the sampled history-policy-state: $a_{-i} \sim w.\pi_{-i}(w.h_{-i})$. Our implementation adds uniform exploration noise $1/|\mathcal{A}_i|$ with a mix-in proportion λ, while the constant c controls the influence of the exploration value $U(h_i a_i)$ relative to the action-value $Q(h_i a_i)$. This differs from [Sil+18] where they sample the noise from a Dirichlet distribution for the current root node only and where the noise is used for exploration during training over multiple episodes. In this work we are instead interested in balancing exploration and exploitation within a single episode and so utilize uniform noise as motivated by Theorem 4.5.5. Specifically,

Algorithm 4 POTMMCP PUCT Action selection

procedure PUCT(h_i)
  for $a_i \in \mathcal{A}_i$ **do**
    $U(h_i a_i) \leftarrow c(P(a_i|h_i)(1-\lambda) + \frac{\lambda}{|\mathcal{A}_i|})\frac{\sqrt{N(h_i)}}{1+N(h_i a_i)}$
  end for
  return $\arg\max_{a_i \in \mathcal{A}_i} \{Q(h_i a_i) + U(h_i a_i)\}$
end procedure

Algorithm 5 POTMMCP Value Function Node Expansion

procedure EXPAND(h_i, π_i)
  for $a_i \in \mathcal{A}_i$ **do**
    $N(h_i a_i) \leftarrow 0$
    $P(a_i|h_i) \leftarrow \pi_i(a_i|h_i)$
    $W(h_i a_i) \leftarrow 0$
    $Q(h_i a_i) \leftarrow 0$
  end for
  return $V^{\pi_i}(h_i)$
end procedure

depending on the values of the constants λ and c, each action will eventually be explored even if the search-policy assigns it zero probability. This ensures the search can always find the optimal action given enough planning time.

In the second stage, upon reaching a leaf node ($h_i \notin \mathcal{T}$), the leaf node is evaluated and expanded by adding it to the tree and adding an edge for each action (Algorithm 5). Evaluation of the leaf node involves estimating two properties of the node; the value v_i and the policy p_i. Both of these are computed using a policy from the set Π_i which is sampled according to the meta-policy and the other agent's policy contained in the history-policy-state particle $\pi_i \sim \sigma_i(\cdot|w.\pi_{-i})$ [2]. Estimating v_i assumes that the policy π_i has a value function, which is the case for policies generated using most learning and planning methods. However, if a value function is not available v_i can be estimated using a MC-rollout instead. Each edge $h_i a_i$ from the leaf node is initialized to $\langle N(h_i a_i) = 0, P(a_i|h_i) = p_i(a_i), W(h_i a_i) = 0, Q(h_i a_i) = 0\rangle$. The value estimate of the leaf node v_i is not used to initialize the edges but instead used in the last stage of the simulation.

In the third and final stage, the statistics for each edge along the simulated trajectory are updated by propagating the value v_i from the leaf node back-up to the root node of the tree. The policy prior for each edge $h_i a_i$ along this path are also updated by averaging over the existing prior and the latest policy $P(a_i|h_i) \leftarrow P(a_i|h_i) + [\pi_i(a_i|h_i) -$

2 Importantly, note that the other agent policy $w.\pi_{-i}$ is sampled according to the planning agents belief $\hat{b}_i$ and so may be different to the true policy of the other agent.

$P(a_i|h_i)]/N(h_i)$. This is a crucial difference in our method. Previous methods apply PUCT to trees where each node is treated as a fully-observed state and so only compute the prior once when the node is first expanded [Sil+18; Sch+20]. In our setting each node in the tree is an estimate of the planning agent's belief. When a node is first expanded it contains only a single particle and so is likely inaccurate, and thus the policy prior will also be inaccurate. As the node is visited during subsequent simulations the belief accuracy improves and so the policy prior is iteratively updated to reflect this. Thus as the number of visits to a belief increases, i.e. $N(h_i) \to \infty$, we have:

$$P(a_i|h_i) \to \sum_{\pi_{-i} \in \Pi_{-i}} b_i(\pi_{-i}|h_i) \sum_{\pi_i \in \Pi_i} \sigma_i(\pi_i|\pi_{-i}) \pi_i(a_i|h_i)$$

Where $b_i(\pi_{-i}|h_i)$ is the true posterior belief over the other agent's policy π_{-i} given the planning agent's history. In this way $P(a_i|h_i)$ is a function of the belief over the other agent's policy.

Once search is complete, the planning agent selects the action at the root node with the greatest visit count $a_{i,t} = \arg\max_{a_i} N(h_{i,t}a_i)$ and receives an observation $o_{i,t+1}$ from the real environment. At this point $\mathcal{T}(h_{i,t}a_{i,t}o_{i,t+1})$ is set as the new root node of the search tree and $\hat{b}_i(h_{i,t}a_{i,t}o_{i,t+1})$ the new root belief.

4.5 THEORETICAL PROPERTIES

In this section we show that POTMMCP converges to the Bayes-optimal policy with respect to the policy set Π_{-i} and prior ρ. The proof is based on converting the problem to a POMDP, we can then apply the analysis in Silver and Veness [SV10]. We point out however, that the original analysis was based on the UCB algorithm [ACF02]. We extend their proof to the PUCB algorithm [Ros11], which requires an additional assumption on the prior probabilities assigned to each action in order to ensure sufficient exploration during search. In practice, we can choose λ so that the assumption is met even if the search-policy prior assigns zero probability to some actions.

4.5.1 *POSG-POMDP Value Equivalence*

Given the set of policies $\Pi_{-i} = \{\pi_{-i,m} | m = 1, \cdots, M\}$ for the other agent $-i$ and a prior distribution over them ρ, where $\rho(\pi_{-i,m}) = Pr(\pi_{-i,m})$, a POSG can be framed as a POMDP for the planning agent i. This frames the original POSG as a Bayesian Reinforcement Learning problem where the joint policy for the other agent π_{-i} and

their history h_{-i} become hidden variables within the state space that must be inferred by the planning agent. Note that this conversion follows a similar construction to I-POMDPs [GD05].

Let $w_t = \langle s, \pi_{-i}, h_{-i,t} \rangle$ denote a history-policy-state at time t. $\mathcal{W}_t$ denotes the space of time t history-policy-states, and $\mathcal{W} = \{\mathcal{W}_t | 0 \leq t \leq T\}$ denotes the space of all possible history-policy-states for time horizon $\mathcal{T}$. $\mathcal{W}$ is finite given finite action and observation spaces of all agents. For problems with an infinite horizon (i.e. no step limit), discounting is required and the horizon can be set such that the value functions are ϵ-optimal, as suggested by Kocsis and Szepesvári [KS06]. We use dot notation to denote the components of a w_t - $w_t.s$, $w_t.\pi_{-i}$, $w_t.h_{-i,t}$. For convenience we use $w_t.a_{-i}$ and $w_t.o_{-i}$ to denote the last action and observation of agent $-i$ contained in the history $w_t.h_{-i,t}$, i.e. $a_{-i,t-1}$ and $o_{-i,t}$. We also substitute in $\vec{a} = \langle a_i, a_{-i} \rangle$ when possible, for brevity.

Lemma 4.5.1. *Given a set of stationary history-based policies Π_{-i} for the other agents $-i$, a prior distribution over this set ρ, and a POSG $\mathcal{M} = \langle \mathcal{I}, \mathcal{S}, S_0, \vec{\mathcal{A}}, \vec{\mathcal{O}}, T, Z, R \rangle$, consider the derived POMDP $\bar{\mathcal{M}} = \langle \mathcal{W}, \bar{b_0}, \mathcal{A}_i, \mathcal{O}_i, \bar{T}, \bar{Z}, \bar{R} \rangle$ for planning agent i with history-policy-states $w_t = \langle s, \pi_{-i}, h_{-i,t} \rangle$ as states, where,*

$$
\begin{aligned}
\mathcal{W} &:= \mathcal{S} \times \Pi_{-i} \times \mathcal{H}_{-i} \\
\bar{b_0}(w_t) &:= b_0(w_t.s)\rho(w_t.\pi_{-i})\delta(w_t.h_{-i}, \emptyset) \\
\bar{T}(w_t, a_i, w_{t'}) &:= w_t.\pi_{-i}(w_{t'}.a_{-i} | w_t.h_{-i}) T(w_t.s, \langle a_i, w_{t'}.a_{-i} \rangle, w_{t'}.s) \\
&\quad \times Z_{-i}(w_{t'}.s, \langle a_i, w_{t'}.a_{-i} \rangle, w_{t'}.o_{-i}) \delta(\langle t', w_{t'}.\pi_{-i} \rangle, \langle t+1, w_t.\pi_{-i} \rangle) \\
\bar{Z}(w_t, a_i, o_i) &:= Z_i(w_t.s, \langle a_i, w_t.a_{-i} \rangle, o_i) \\
\bar{R}(w_t, a_i) &:= \sum_{a_{-i} \in \mathcal{A}_{-i}} w_t.\pi_{-i}(a_{-i} | w_t.h_{-i}) R_i(w_t.s, \langle a_i, a_{-i} \rangle)
\end{aligned}
$$

and δ is the Kronecker Delta function which is 1 if the function arguments are equal, otherwise 0, then the value function $\bar{V}^{\pi_i}(h_{i,t})$ of the derived POMDP is equivalent to the value function $V^{\pi_i}(h_{i,t})$ of the POSG, $\forall \pi_i \bar{V}^{\pi_i}(h_{i,t}) = V^{\pi_i}(h_{i,t})$.

Proof. By backward induction on the Bellman equation, starting from the horizon,

$$
\begin{aligned}
V^{\pi_i}(h_{i,t}) &= \sum_{s\in\mathcal{S}} \sum_{\pi_{-i}\in\Pi_{-i}} \sum_{h_{-i,t}\in\mathcal{H}_{i,t}} b_i(\langle s, \pi_{-i}, h_{-i,t}\rangle | h_{i,t}) \sum_{a_i\in\mathcal{A}_i} \pi_i(a_i|h_{i,t}) \sum_{a_{-i}\in\mathcal{A}_{-i}} \pi_{-i}(a_{-i}|h_{-i,t}) \\
&\quad \times \left[R_i(s,\vec{a}) + \gamma \sum_{s'\in\mathcal{S}} T(s,\vec{a},s') \sum_{o_i\in\mathcal{O}_i} Z_i(s',\vec{a},o_i) V^{\pi_i}(h_{i,t}a_i o_i) \right] \\
&= \sum_{w_t} b_i(w_t|h_{i,t}) \sum_{a_i\in\mathcal{A}_i} \pi_i(a_i|h_{i,t}) \sum_{a_{-i}\in\mathcal{A}_{-i}} w_t.\pi_{-i}(a_{-i}|w_t.h_{-i}) \\
&\quad \times \left[R_i(w_t.s,\vec{a}) + \gamma \sum_{s'\in\mathcal{S}} T(w_t.s,\vec{a},s') \sum_{o_i\in\mathcal{O}_i} Z_i(s',\vec{a},o_i) V^{\pi_i}(h_{i,t}a_i o_i) \right] \\
&= \sum_{w_t} b_i(w_t|h_{i,t}) \sum_{a_i\in\mathcal{A}_i} \pi_i(a_i|h_{i,t}) \left[\sum_{a_{-i}\in\mathcal{A}_{-i}} w_t.\pi_{-i}(a_{-i}|w_t.h_{-i}) R_i(w_t.s,\vec{a}) \right. \\
&\quad \left. + \gamma \sum_{a_{-i}\in\mathcal{A}_{-i}} w_t.\pi_{-i}(a_{-i}|w_t.h_{-i}) \sum_{s'\in\mathcal{S}} T(w_t.s,\vec{a},s') \sum_{o_i\in\mathcal{O}_i} Z_i(s',\vec{a},o_i) V^{\pi_i}(h_{i,t}a_i o_i) \right] \\
&= \sum_{w_t} b_i(w_t|h_{i,t}) \sum_{a_i\in\mathcal{A}_i} \pi_i(a_i|h_{i,t}) \left[\sum_{a_{-i}\in\mathcal{A}_{-i}} w_t.\pi_{-i}(a_{-i}|w_t.h_{-i}) R_i(w_t.s,\vec{a}) \right. \\
&\quad + \gamma \sum_{a_{-i}\in\mathcal{A}_{-i}} w_t.\pi_{-i}(a_{-i}|w_t.h_{-i}) \sum_{s'\in\mathcal{S}} T(w_t.s,\vec{a},s') \\
&\quad \left. \times \sum_{o_{-i}\in\mathcal{O}_{-i}} Z_{-i}(s',\vec{a},o_{-i}) \sum_{o_i\in\mathcal{O}_i} Z_i(s',\vec{a},o_i) V^{\pi_i}(h_{i,t}a_i o_i) \right] \\
&= \sum_{w_t} b_i(w_t|h_{i,t}) \sum_{a_i\in\mathcal{A}_i} \pi_i(a_i|h_{i,t}) \\
&\quad \times \left[\bar{R}(w_t,a_i) + \gamma \sum_{a_{-i}\in\mathcal{A}_{-i}} \sum_{s'\in\mathcal{S}} \sum_{o_{-i}\in\mathcal{O}_{-i}} \bar{T}(w_t,a_i,\langle s', w_t.\pi_{-i}, w_t.h_{-i}a_{-i}o_{-i}\rangle) \right. \\
&\quad \left. \times \sum_{o_i\in\mathcal{O}_i} \bar{Z}(\langle s', w_t.\pi_{-i}, w_t.h_{-i}a_{-i}o_{-i}\rangle, a_i, o_i) \bar{V}^{\pi_i}(h_{i,t}a_i o_i) \right] \\
&= \sum_{w_t} b_i(w_t|h_{i,t}) \sum_{a_i\in\mathcal{A}_i} \pi_i(a_i|h_{i,t}) \\
&\quad \times \left[\bar{R}(w_t,a_i) + \gamma \sum_{w_{t+1}} \bar{T}(w_t,a_i,w_{t+1}) \sum_{o_i\in\mathcal{O}_i} \bar{Z}(w_{t+1},a_i,o_i) \bar{V}^{\pi_i}(h_{i,t}a_i o_i) \right] \\
&= \bar{V}^{\pi_i}(h_{i,t})
\end{aligned}
$$

□

4.5.2 *Rollout Distribution Equivalence*

Here we show show that the proposed algorithm POTMMCP converges to the ϵ-optimal solution of the POSG under the type-based reasoning assumptions. The main steps of our proof are adapted from [SV10], however we extend the original proof to the type-based, multi-agent setting. Note, our proof does not hold for POSGs in general, but only for POSGs under the type-based reasoning assumptions. When mentioning the POSG, we are referring to the POSG problem along with a set of stationary history-based policies Π_{-i} for the other agent $-i$, and a prior distribution over this set ρ.

Our proof is based on showing that, for an arbitrary rollout policy π_i for the planning agent i, the *POSG rollout distribution* (the distribution over full histories of agent i when performing root sampling of the state s, and other agent policy π_{-i} and history h_{-i}) is equal to the *derived POMDP rollout distribution* (the distribution over full histories when sampling in the history-policy-state POMDP). Given that these two distributions are identical, the statistics maintained in the search tree will converge to the same number in expectation. This allows us to apply the analysis for MCTS in POMDPs from Silver and Veness [SV10] to POTMMCP in typed POSGs.

Definition 4.5.2. *The **POSG rollout distribution**, $\mathcal{D}_K^{\pi_i}(h_{i,d})$, is the distribution over histories for planning agent i in a POSG, where the other agent $-i$'s policy is from the set Π_{-i} with a prior distribution over this set ρ, when performing Monte Carlo simulations according to a policy π_i in combination with root sampling an initial state, other agent policy, and other agent history. This distribution, for a particular time d, is given by $\mathcal{D}_K^{\pi_i}(h_{i,d}) := \frac{1}{K_d}\sum_{k=1}^{K} \mathbb{I}_{h_{i,d}}(h_{i,d}^{(k)})$, where K is the number of simulations that comprise the empirical distribution, K_d is the number of simulations that reach depth d (not all simulations might be equally long), and $h_{i,d}^{(k)}$ is the history specified by the k-th particle at time d.*

Definition 4.5.3. *The **derived POMDP rollout distribution**, $\hat{\mathcal{D}}^{\pi_i}(h_{i,d})$, is the distribution over histories for planning agent i in the derived POMDP, when performing Monte Carlo simulations according to policy π_i and sampling state transitions, observations, and rewards from $\hat{\mathcal{M}}$. This distribution, for a particular time $d+1$ history, is given by $\hat{\mathcal{D}}^{\pi_i}(h_{i,d}a_i o_i) = \hat{\mathcal{D}}^{\pi_i}(h_{i,d})\pi_i(a_i|h_{i,d})\sum_{w_d\in\mathcal{W}_d} b_i(w_d|h_{i,d})\sum_{w_{d+1}\in\mathcal{W}_{d+1}} \bar{T}(w_d, a_i, w_{d+1})\bar{Z}(w_{d+1}a_i, o_i)$.*

Now, we show that these distributions are the same in the limit of the number of simulations.

Lemma 4.5.4. *Given a POSG $\mathcal{M}$, a set of stationary history-based policies Π_{-i} for the other agent $-i$, and a prior distribution over this set ρ, for any rollout policy, π_i, for planning agent i the POSG rollout distribution converges in probability to the derived POMDP rollout distribution, $\forall \pi_i, h_{i,d}\ \mathcal{D}_K^{\pi_i}(h_{i,d}) \xrightarrow{p} \hat{\mathcal{D}}^{\pi_i}(h_{i,d})$.*

Proof. By forward induction from $h_{i,t}$, where t is the timestep at the root.

Base case: At the root when ($d = t$, $h_{i,d} = h_{i,t}$), it is clear that $\mathcal{D}_K^{\pi_i}(h_{i,d}) = \hat{\mathcal{D}}^{\pi_i}(h_{i,d}) = 1$ since all simulations go through the root node.

Step case: Assume $\mathcal{D}_k^{\pi_i}(h_{i,d}) = \hat{\mathcal{D}}^{\pi_i}(h_{i,d})$ for all time d histories where $t \leq d < T$. Consider any time $d+1$ history $h_{i,d+1} = h_{i,d}a_i o_i$, the following relation holds:

$$
\begin{aligned}
\mathcal{D}_K^{\pi_i}(h_{i,d}a_i o_i) &= \mathcal{D}_K^{\pi_i}(h_{i,d})\pi_i(a_i|h_{i,d}) \sum_{s\in\mathcal{S}} \sum_{\pi_{-i}\in\Pi_{-i}} \sum_{h_{-i,d}\in\mathcal{H}_{i,d}} b_i(\langle s, \pi_{-i}, h_{-i,d}\rangle|h_{i,d}) \\
&\quad \times \sum_{a_{-i}\in\mathcal{A}_{-i}} \pi_{-i}(a_{-i}|h_{-i,d}) \sum_{s'\in\mathcal{S}} T(s,\vec{a},s')Z_i(s',\vec{a},o_i) \\
&= \mathcal{D}_K^{\pi_i}(h_{i,d})\pi_i(a_i|h_{i,d}) \sum_{w_d\in\mathcal{W}_d} b_i(w_d|h_{i,d}) \sum_{a_{-i}\in\mathcal{A}_{-i}} w_d.\pi_{-i}(a_{-i}|w_d.h_{-i}) \\
&\quad \times \sum_{s'\in\mathcal{S}} T(w_d.s,\vec{a},s')Z_i(s',\vec{a},o_i) \\
&= \mathcal{D}_K^{\pi_i}(h_{i,d})\pi_i(a_i|h_{i,d}) \sum_{w_d\in\mathcal{W}_d} b_i(w_d|h_{i,d}) \sum_{a_{-i}\in\mathcal{A}_{-i}} w_d.\pi_{-i}(a_{-i}|w_d.h_{-i}) \\
&\quad \times \sum_{s'\in\mathcal{S}} T(w_d.s,\vec{a},s')Z_i(s',\vec{a},o_i) \sum_{o_{-i}\in\mathcal{O}_{-i}} Z_{-i}(s',\vec{a},o_{-i}) \qquad (4.1) \\
&= \hat{\mathcal{D}}^{\pi_i}(h_{i,d})\pi_i(a_i|h_{i,d}) \sum_{w_d\in\mathcal{W}_d} b_i(w_d|h_{i,d}) \\
&\quad \times \sum_{s'\in\mathcal{S}} \sum_{a_{-i}\in\mathcal{A}_{-i}} \sum_{o_{-i}\in\mathcal{O}_{-i}} \bar{T}(w_d, a_i, \langle s', w_d.\pi_{-i}, w_d.h_{-i}a_{-i}o_{-i}\rangle) \\
&\quad \times \bar{Z}(\langle s', w_d.\pi_{-i}, w_d.h_{-i}a_{-i}o_{-i}\rangle, a_i, o_i) \\
&= \hat{\mathcal{D}}^{\pi_i}(h_{i,d})\pi_i(a_i|h_{i,d}) \sum_{w_d\in\mathcal{W}_d} b_i(w_d|h_{i,d}) \sum_{w_{d+1}\in\mathcal{W}} \bar{T}(w_d, a_i, w_{d+1}) \\
&\quad \times \bar{Z}(w_{d+1}a_i, o_i) \\
&= \hat{\mathcal{D}}^{\pi_i}(h_{i,d}a_i o_i)
\end{aligned}
$$

Where the fourth line is obtained using the induction hypothesis, and the rest from the definitions.

□

4.5.3 *Convergence*

Now that we have shown that the rollout distributions are equivalent between the POSG and derived POMDP, we can present the main convergence result.

Define $V(h_i) = \max_{a_i\in\mathcal{A}_i} Q(h_i a_i)\ \forall h_i \in \mathcal{H}_i$.

Theorem 4.5.5. *For all $\epsilon > 0$ (the numerical precision, see Algorithm 3), given a suitably chosen c (e.g. $c > \frac{G_{max}}{1-\gamma}$) and prior probabilities $P(a_i|h_i) > 0, \forall h_i \in \mathcal{H}_i, a_i \in \mathcal{A}_i$ (e.g. $\lambda > 0$),*

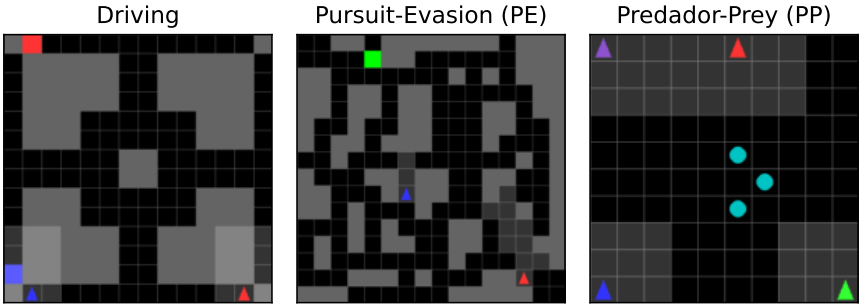


Figure 4.2: The environments use for our experiments.

from history h_i POTMMCP constructs a value function at the root node that converges in probability to an ϵ'-optimal value function, $V(h_i) \xrightarrow{p} V^*_{\epsilon'}(h_i)$, where $\epsilon' = \frac{\epsilon}{1-\gamma}$. As the number of visits $N(h_i)$ approaches infinity, the bias of $V(h_i)$ is $O(\log N(h_i)/N(h_i))$.

Proof. By 4.5.4 the POTMMCP simulations can be mapped to the PUCT simulations in the derived POMDP. By 4.5.1 a POSG with a set of stationary policies for the other agent, and a prior over this set is a POMDP, so the analysis from Silver and Veness [SV10] applies to POTMMCP, noting that for $N(h_i)$ sufficiently large PUCB has the same regret bounds as UCB given each action is given prior probability $P(h_i a_i) > 0, \forall h_i \in \mathcal{H}_i, a_i \in \mathcal{A}_i$ (Rosin [Ros11], Thm. 2 and Cor. 2) and so the same analysis of POMCP using UCT applies to POMCP using PUCT. □

4.6 EXPERIMENTS

We ran experiments on three benchmark environments designed to evaluate POTMMCP against existing state-of-the-art methods across a diverse set of scenarios. In addition, we conducted a number of ablations to investigate the different meta-policies and the learning dynamics of our method.[3]

4.6.1 *Environments*

For our experiments we used one cooperative, one competitive, and one mixed environment (Figure 4.2). The largest of which has four agents and on the order of 10^{14} states and 10^8 observations (Table 4.1). Each environment models a practical real-world scenario; namely, navigation, pursuit-evasion, and ad-hoc teamwork. These environments add additional complexities to existing benchmarks [Eck+20; Kak+22] and were chosen in order to assess POTMMCP's ability across a range of domains that required both planning over many steps and reasoning about the other agent's behavior.

3 All code used for the experiments is available at https://github.com/Jjschwartz/potmmcp

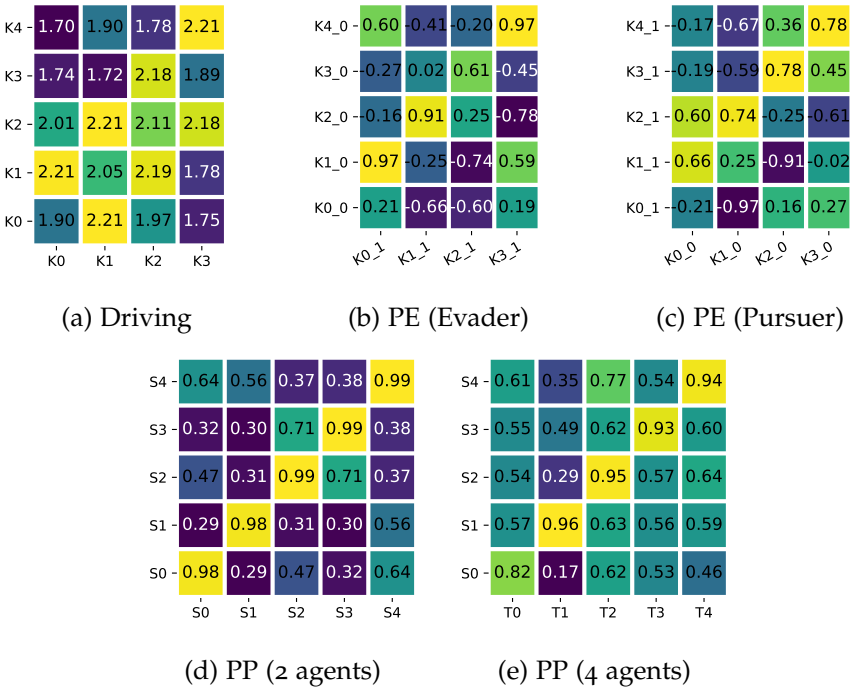
Table 4.1: State, action, and observation space size of each environment.

Environment	$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{A}_i\rvert$	$\lvert\mathcal{O}_i\rvert$
Driving	2.7×10^8	5	3×10^6
PE	2.3×10^7	4	64
PP (two-agents)	7×10^{10}	5	7×10^6
PP (four-agents)	6×10^{14}	5	6×10^{10}

Driving: A general-sum grid world navigation problem requiring coordination [LP19]. Each agent is tasked with driving a vehicle from start to destination while avoiding crashing into other vehicles. Each agent observes its local area, speed, and destination, and whether they've reached their destination or crashed. Agents receive a reward of 1 if they reach their destination, -1 if they crash into another vehicle. To reduce exploration difficulty agents receive a reward of 0.05 each time they make progress towards their destination. The exploration bonus is analogous to GPS in that it provides guidance for navigation but not for how to coordinate with other vehicles. Episodes end when all agents either crash into another vehicle or reach their destination, or 50 steps have passed.

Pursuit-Evasion (PE): A two-agent asymmetric zero-sum grid world problem where the *evader* has to reach a safe location without being observed by the *pursuer* [SvW18; SZK22]. This is a larger version of the Pursuit-Evasion environment from Chapter 3. The evader's goal is to reach a safe location, while the pursuer's aim is to spot the evader before it reaches its goal. The evader is considered caught if it is observed by the pursuer. Both agents have knowledge of each others starting locations, however, only the evader has knowledge of its goal location. The pursuer only knows the set of possible goal locations. Thus, this environment requires each agent to reason about the path the other agent will take through the dense environment. Each agent receives six bits of observation per step. Four bits indicate whether there is a wall or not in each of the cardinal directions, one bit indicates whether the opponent can be seen in the agent's field of vision (which is a cone in front of the agent), and the final bit indicates whether the opponent can be heard within Manhattan distance two of the agent. Due to the lack of precision of these observations, the pursuer never knows the exact position of the evader and vice versa. Similar to the Driving environment the evader agent receives a small bonus whenever it makes progress towards its goal location, while the pursuer receives the opposite reward. Episodes end when the evader is captured or reaches the safe location, or 100 steps have passed.

Predator-Prey (PP): A co-operative grid world problem where multiple predator agents must work together to catch prey [Low+17]. Prey are controlled autonomously and preference movement away from any observable predators or other prey. Predators



(a) Driving (b) PE (Evader) (c) PE (Pursuer)

(d) PP (2 agents) (e) PP (4 agents)

Figure 4.3: Empirical-game payoff matrices for the set of policies Π for each environment. The top row shows the mean payoff after 1000 evaluation episodes, while the bottom row shows the corresponding 95% confidence intervals. Each cell shows the result for the row policy when paired with the column policy or team of policies.

can catch prey by being in an adjacent cell, with the number of predators required to catch a prey based on the prey strength. Both predators and prey can observe a 5-by-5 area around themselves, namely whether each cell contains a wall, predator, prey, or is empty. Each prey capture gives all predators a reward of $1/N_{prey}$. Predators start each episode from random separate locations along the edge of the grid, while prey start together in the center of the grid. We ran experiments on two different versions of the environment, where both versions had three prey. The *two-agent* version has two predators with each prey requiring two predators to capture. The *four-agent* version has four predators with each prey requiring three predators to capture. Episodes end when all prey are captured, or 50 steps have passed.

4.6.2 *Policies*

For each environment, a diverse set of four to five policies was created and used for the set Π. These policies were used for the other agent during evaluations and also for the meta-policy σ_i and policy prior ρ. Figure 4.3 shows the empirical-game payoff matrices used for computing the meta-policy in our experiments. The payoff matrices also demonstrate the diversity in terms of pairwise payoffs of the policy set for each environment.

Each policy was a deep neural network trained using Proximal Policy Optimization (PPO) [Sch+17] and different multi-agent training schemes. We used Rllib [Lia+18] to handle efficient training of policies. The same neural network architecture was used for all policies, namely two fully-connected layers with 64 and 32 units, respectively, followed by a 256 unit LSTM, whose output was fed into two separate fully connected output heads, one for the policy and one for the value function. The neural network architecture and training hyperparameters are shown in Table 4.2. All policies were trained until convergence, as indicated by their learning curves.

Table 4.2: PPO hyperparameters used for training the other agent policies in each environment.

Hyper parameter	Driving	PE	PP
Training steps	10,240,000		
Fully Connected Network Layers	[64, 32]		
LSTM Cell Size	256		
Learning Rate	0.0003		
KL Coefficient	0.2		
KL target	0.01		
Batch size	2048		
LSTM training sequence length	20		
Entropy Bonus Cofficient	0.001		
Clip param	0.3		
γ	0.99		0.999
GAE λ	0.9		0.95
SGD Minibatch size	256		512
Num. SGD Iterations	10		2

For the Driving and PE environments we trained a set of five K-level reasoning (KLR) policies. In the KLR training scheme, policies are trained in a hierarchy, the level $K = 0$ policy is trained against a uniform random policy, level $K = 1$ is trained against the level $K = 0$, and so on with the level K policy trained as a best response to the level $K - 1$ policy for $K > 0$. We trained policies synchronously using the Synchronous KLR training method [Cui+21]. For the policy prior ρ, we used a uniform distribution over the policies with reasoning levels $k = 0, 1, 2, 3$. The meta-policy was defined using all five policies, which included the level $k = 4$ policy. This meant the planning agent had access to a best-response for all the policies in ρ and allowed a fair comparison against the Best-Response baseline.

For the PP environment, since it is fully cooperative we trained five independent teams of agents using self-play [Tes94] where each team consisted of identical copies of the same policy. The policy architecture and hyperparameters were the same across each of the five teams, except for the initial random seed which was different. Using a different seed meant each team converged to a different solution and lead to a diverse

set of policies (as shown by the empirical-game payoffs, Figure 4.3). We used a uniform distribution over the five teams for the prior ρ, with each team made up of copies of the same policy from the set of five trained self-play policies - one copy in the two-agent version, and three copies in the four-agent version. This setup tested the planning agent's *ad-hoc teamwork* ability. The meta-policy was defined using all five policies.

4.6.3 *Baselines*

We compared POTMMCP against a number of baselines in each environment. For the planning baseline, we adapted the current state-of-the-art method for planning in partially observable, typed multi-agent systems designed for I-POMDPs with communication [Kak+22]. The full algorithm incorporates an explicit model of communication and while it can in principle support beliefs over multiple types, it was only tested in the setting where the other agent's type is fixed and known. We adapted their method to our setting by removing the explicit communication model and extending it to handle beliefs over multiple-agent types, we refer to this baseline as *I-POMCP-PF*. We also use the same belief update and reinvigoration strategies for both I-POMCP-PF and POTMMCP to keep the comparison fair and since both would benefit equally from changes to the belief updates.

Meta-policy: Uses a policy from the set Π which is sampled at the start of the episode based on the meta-policy with respect to the distribution ρ. This acts as a lower bound on the performance of POTMMCP, and represents an agent with access to the meta-policy but without using beliefs or search.

Best-Response: This is the best performing policy from the set of fixed policies Π against each policy, assuming full-knowledge of the policy of the other agent. This acts as an approximate upper-bound given there is a policy in the set Π that is a best-response policy to at least one policy in the set, which is the case in our experiments.

I-POMCP-PF + Random: The I-POMCP-PF algorithm using MC-rollouts with a uniform random policy for leaf node evaluations. This is the version used in the original paper, adapted for our problem setting.

I-POMCP-PF + Meta: The I-POMCP-PF algorithm using the value function of the meta-policy for leaf node evaluations in place of MC-rollouts.

4.6.4 *Experimental Setup*

For each experiment we ran a minimum of 400 episodes, or 48 hours of total computation time, whichever came first. We tested each planning method using a search

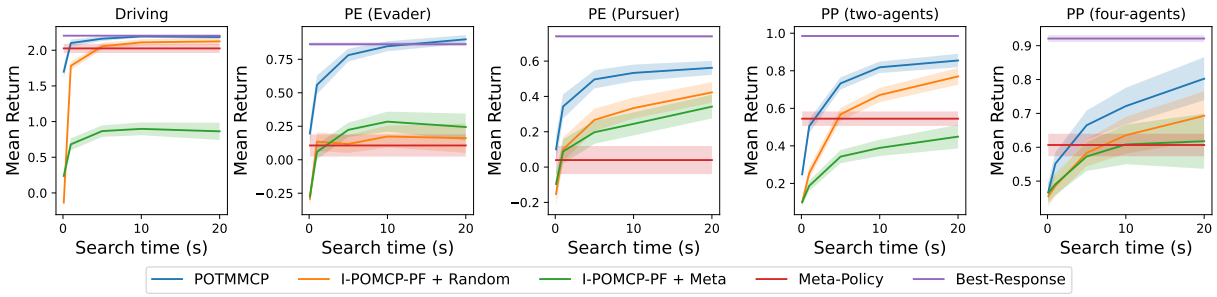


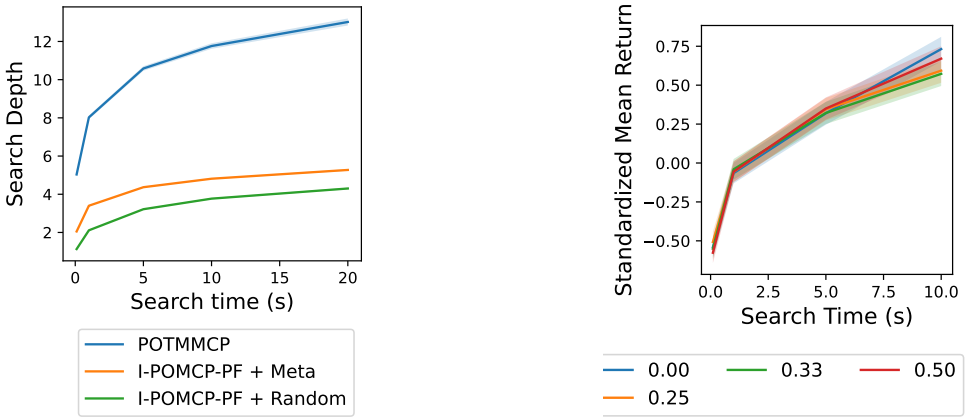
Figure 4.4: Mean episode return of POTMMCP and baselines across each environment. Results are for POTMMCP and baselines using the softmax $\sigma_i^{0.25}$ meta-policy. Shaded areas show the 95% CI.

time per step of $N_{time} \in [0.1, 1, 5, 10, 20]$ seconds and $N_{particles} = 100 N_{time}$ particles. As per prior work [SV10], after each real step in the environment rejection sampling was used to add additional particles to the root belief until it contained at least $(1 + 1/16)N_{particles}$. For all experiments we used $\epsilon = 0.01$ and a discount of $\gamma = 0.99$. For POTMMCP we chose PUCT constants based on prior work [Sch+20]. We used exploration constant $c = 1.25$ along with normalized Q-values to handle the returns being outside of $[0, 1]$ bounds in the tested environments. We used $\lambda = 0.5$ for the mix-in proportion for each experiment, although POTMMCP was very robust to the value of λ in the environments and settings used for our experiments (Figure 4.5b and Appendix 4.A). For the I-POMCP-PF baselines we used a UCB exploration constant of $c = \sqrt{2}$ [ACF02] along with normalized Q-values.

4.6.5 *Evaluation of Returns*

Figure 4.4 shows the mean episode return of POTMMCP and the baseline methods in each environment. Given the same planning time, POTMMCP outperformed both versions of I-POMCP-PF across all environments. Furthermore, given enough planning time POTMMCP matched the performance of the Best-Response baseline in two environments. We attribute the gains in performance of POTMMCP to its ability to useful bias introduced into search by the meta-policy and improved leaf-node evaluation.

Biasing the action-selection meant less planning time was spent exploring lower value actions and lead to significantly deeper searches ($\sim$ 13 for POTMMCP vs $\sim$ 5 for I-POMCP-PF for 20 s planning time) as shown in Figure 4.5a. This translated to a longer effective planning horizon for POTMMCP. The benefits of this are especially evident in the PE (Evader) problem which requires long horizon planning since the evaders choices early in the episode affects its chances of reaching its goal without being spotted by the pursuer.



(a) (b)

Figure 4.5: (a) Maximum search depth vs planning time of POTMMCP and I-POMCP. We found POTMMCP generated much deeper searches due to improved search efficiency and better exploration stemming from the use of the meta-policy. (b) Mean standardized episode returns of POTMMCP using different values for λ. POTMMCP was very robust to the λ value across all environments. Both figures show results averaged across all environments with shaded areas showing the 95% confidence interval.

Using the meta-policy's value function for leaf node evaluation meant that POTMMCP was able to avoid expensive MC-rollouts and ultimately led to faster simulations and more efficient planning. This helps explain the gains in performance over I-POMCP-PF + Random. However, we also observe similar gains over I-POMCP-PF + Meta which, like POTMMCP, uses the meta-policy's value function and avoids MC-rollouts. Indeed I-POMCP-PF's performance actually suffers from using the meta-policy when compared with using the random policy with MC-rollouts. This result highlights a limitation of UCB when combined with value-functions. In our experiments where the rewards are not especially dense, we found the difference in value estimates produced by the search-policy between two actions from the same belief can be very small (i.e. a factor of γ). When using UCB this small difference can be dominated by the variance in returns generated during simulations, causing UCB to be unable to effectively distinguish the best action during search when using the meta-policy's value estimates. PUCT reduces the influence of the simulation variance by biasing the actions according to the search-policy - which by definition selects actions that maximize the value estimates (even if the difference is small between actions). This bias acts to amplify the value of the best actions relative to the other actions and allows POTMMCP use the meta-policy's value estimates more effectively. Of course this has the drawback that if the search-policy is bad, then it takes more planning time to overcome the bias introduced and find the optimal actions. The fact that we see improved performance when using the meta-policy across all environments provides empirical evidence that it makes for a good search-policy.

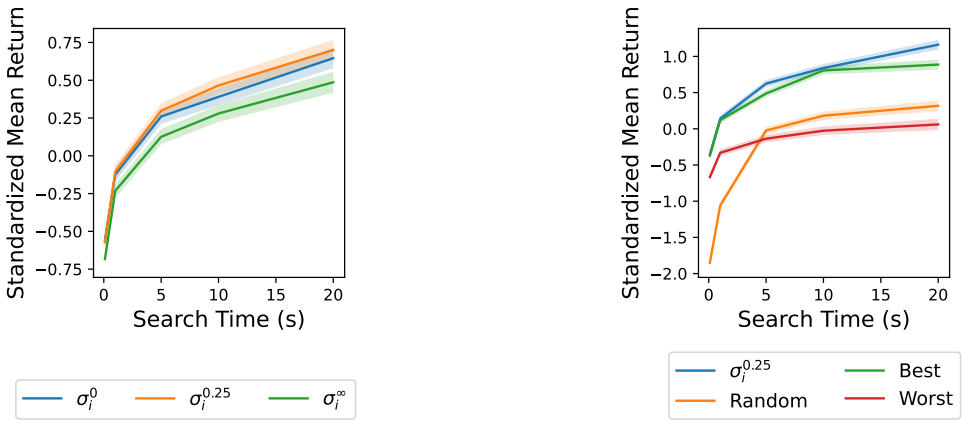


Figure 4.6: Evaluation of POTMMCP using different search policies. Both plots show standardized mean episode returns averaged across all environments. (Left) Shows comparison of greedy σ_i^0, softmax $\sigma_i^{0.25}$, and uniform σ_i^∞ meta-policies. (Right) Shows comparison between $\sigma_i^{0.25}$, the best and worst performing policy from the set of policies Π_i, and the uniform random policy. Shaded areas show 95% CI.

4.6.6 *Evaluation of Meta-Policies*

To assess the effect of meta-policy choice we compared POTMMCP using greedy σ_i^0, softmax $\sigma_i^{0.25}$, and uniform σ_i^∞ meta-policies. Figure 4.6 (left) shows the standardized mean episode returns of POTMMCP using the different meta-policies averaged across all environments, with results for individual environments available in Appendix 4.B. Overall we found $\sigma_i^{0.25}$ was the most robust, performing best or close to best across all environments, and had the highest mean performance when averaged across all environments (although not significantly so). We expect this is due to the reasons covered in Section 4.4.1. It is worth noting that we didn't tune the τ parameter for our experiments and expect marginal performance gains may be seen using tuned values. A question for future work is whether an optimal value for τ can be inferred from the empirical payoff-matrix.

To study the benefit of using a meta-policy, we tested POTMMCP using a range of different search policies. Specifically, we tested replacing the meta-policy with each individual policy in the set Π_i and also the uniform random policy. Figure 4.6 (right) shows the standardized results averaged across all environments, while the results for each individual environment are available in Appendix 4.C. We found that the meta-policy lead to the most consistent results, having similar or better performance than the best single-policy in each environment, and significantly better than the random policy. While the best single-policy was able to perform comparable to the meta-policy, we typically observed a significant gap between the best and worst performing policies and no obvious way to tell beforehand which of the policies out the set would perform

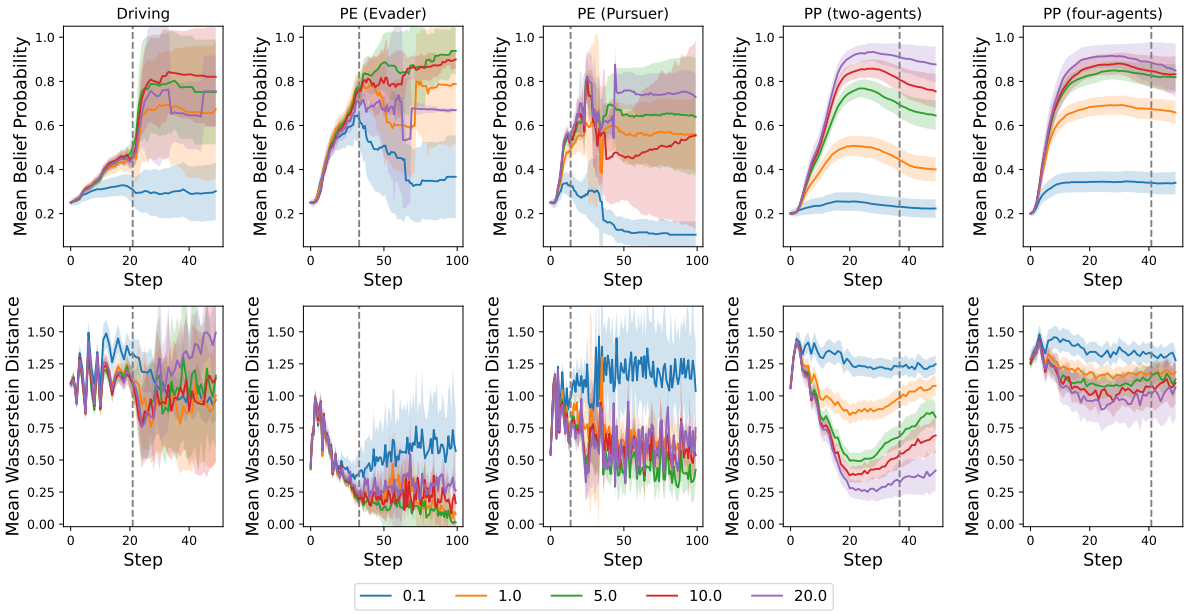


Figure 4.7: POTMMCP's belief accuracy during an episode in each environment. (Top row) Mean probability assigned by the belief to the other agent's true policy. (Bottom row) Wasserstein distance between the belief's estimated action distribution and the other agent's true action distribution. Each line shows a different planning time with shaded areas showing 95% confidence intervals. In all environments episode lengths were often shorter than the max episode lengths, leading to larger confidence intervals for steps later in the episode. The vertical dashed line indicates the mean number of steps taken per episode when using 20 s of planning time. We observe POTMMCP's belief about the other agent improved as each episode evolved, and that increasing planning time lead to improved belief accuracy.

best/worse. In contrast, the meta-policy lead to the most robust performance without the need to test each individual search policy to find the best one.

4.6.7 *Evaluation of Beliefs*

To better understand the adaptive capabilities of POTMMCP we analyzed the evolution of beliefs during an episode. Figure 4.7 shows the accuracy of POTMMCP's belief in each environment. We observed that in all environments POTMMCP's belief in the correct other agent type increased as the episode progressed. A similar trend occurred for the action distribution accuracy, with the distance between the estimated and true distributions decreasing over time. We also found roughly monotonic improvement in belief accuracy with increased planning time. It is worth noting there was a consistent drop in accuracy for steps that occurred beyond the typical episode length (e.g. ~ 37 for the PP (two-agent) problem), this is likely due to increased uncertainty from the environment moving into less common states which can impact the planning agent's beliefs and the behavior of the other agents. However, even taking this into account, these results empirically demonstrate POTMMCP's ability to learn the other agent's

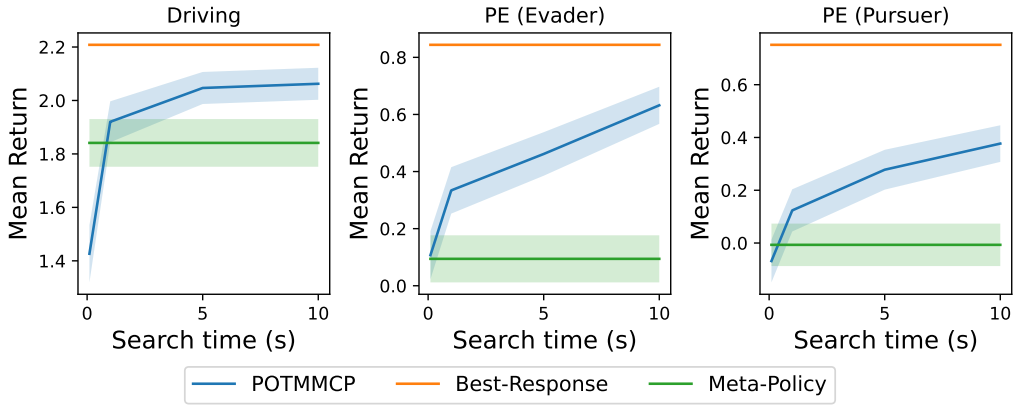


Figure 4.8: Mean episode return of POTMMCP and baselines when using a larger policy set $|\Pi_{-i}| = 15$ in the Driving and Pursuit-Evasion environments. Results are for POTMMCP and baselines using the softmax $\sigma_i^{0.25}$ meta-policy. Shaded areas show the 95% CI.

type from online interactions and helps explain POTMMCP robust performance against a diverse set of policies.

4.6.8 *Scalability with Policy Set Size*

So far our we have shown results for experiments where the number of possible policies for the other agent has been relatively small, $|\Pi_{-i}| \in [4, 5]$. To test how well POTMMCP performs compared to the baselines with larger policy sets, we ran additional experiments for the Driving and Pursuit-Evasion environments with $|\Pi_{-i}| \in 15$, the results are shown in Figure 4.8. Once again POTMMCP's scaled well, performing significantly better than the meta-baseline and monotonically improving with planning time. Furthermore, due to POTMMCP using MC simulations, the computation time per simulation does not increase with policy set size. The key potential limitation in practice is the belief accuracy, since the number of particles needed to represent the space of policies accurately will grow with the policy set size. However, depending on the diversity of the behaviors represented by the different policies, there can be considerable overlap between policy behaviors. In this case the number of particles needed to represent a useful belief may still be manageable.

4.6.9 *Sensitivity to Novel Policies*

As a final experiment we explored how each method fairs when faced with novel policies, outside of the known set Π. In this setting, we violate the assumption that the other agent policies are coming from a known distribution. However, ultimately we would like our planning agents to be robust against model inaccuracies and so we include these results to motivate future research.

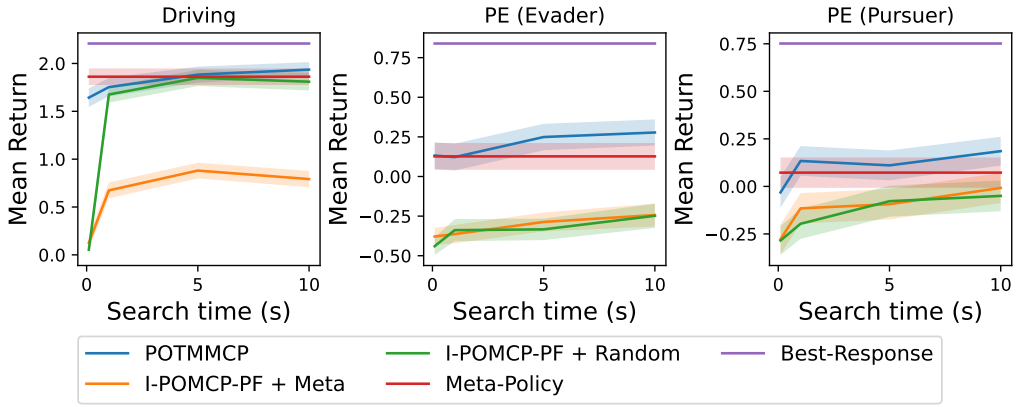


Figure 4.9: Mean episode return of POTMMCP and baseline methods in the Driving and Pursuit-Evasion environments when paired with other agent policies outside of the known set Π_{-i}. Results are for POTMMCP and baselines using the softmax $\sigma_i^{0.25}$ meta-policy. Shaded areas show the 95% CI.

In Figure 4.9 we show the performance of POTMMCP and baselines when paired with other agents using policies from $\hat{\Pi}_{-i}$ that are not included in the set of known policies Π_{-i}. The policies in $\hat{\Pi}_{-i}$ were generated in the same way as those within the set Π_{-i}, but with a different seed leading to differences in behaviors. From the results we can see that POTMMCP is more robust to the out-of-distribution policies than the I-POMCP-PF baselines. However, comparing POTMMCP's performance against the new policies $\hat{\Pi}_{-i}$ with its performance against the known Π_{-i} policies (Figure 4.4), we can see a significant drop. This highlights the sensitivity of POTMMCP to out-of-distribution policies, at least for the policy prior used in our experiments. An interesting follow-up question, and something we leave for future work, is what types of policy priors can lead to more robust performance when faced with novel co-players?

4.7 CONCLUSION

In this chapter we presented a scalable planning method for type-based reasoning in large partially observable environments. Our algorithm, POTMMCP, offers two key contributions over existing decision-theoretic planners. The first is the use of PUCT for action selection during search, which prior to this work had been limited to game-theoretic settings. The second is a new meta-policy which is used to guide the search. Through extensive evaluations we demonstrate that POTMMCP significantly improves on the performance and planning efficiency of existing state-of-the-art methods. Furthermore, we prove that POTMMCP will converge in the limit to the Bayes-optimal solution.

Multiple avenues for future research exist. Extending POTMMCP to handle continuous state, action, and observation spaces would make it more widely applicable. We proposed a general and flexible meta-policy and validated it empirically, however based on prior work [Lan+17], a more principled approach likely exists. Lastly, exploring methods for improving generalization to policies outside of the known set would be a valuable addition.

4.A EVALUATION OF EXPLORATION NOISE LEVELS

Figure 4.10 show the performance of POTMMCP using different values for the λ hyperparameter.

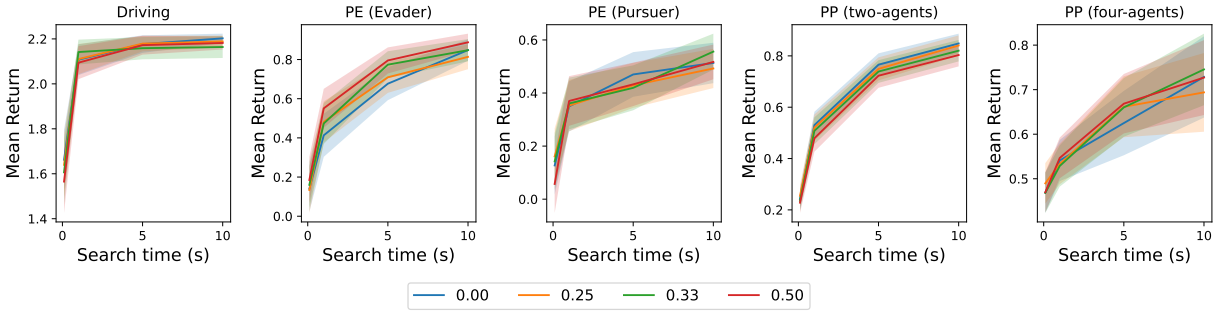


Figure 4.10: Comparison of POTMMCP using different λ values. Each figure shows performance of POTMMCP using the $\sigma^{0.25}$ meta-policy. Shaded areas show the 95% confidence interval.

4.B EVALUATION OF DIFFERENT META-POLICIES

Figure 4.11 shows the performance of POTMMCP using the different meta-policies in each environment.

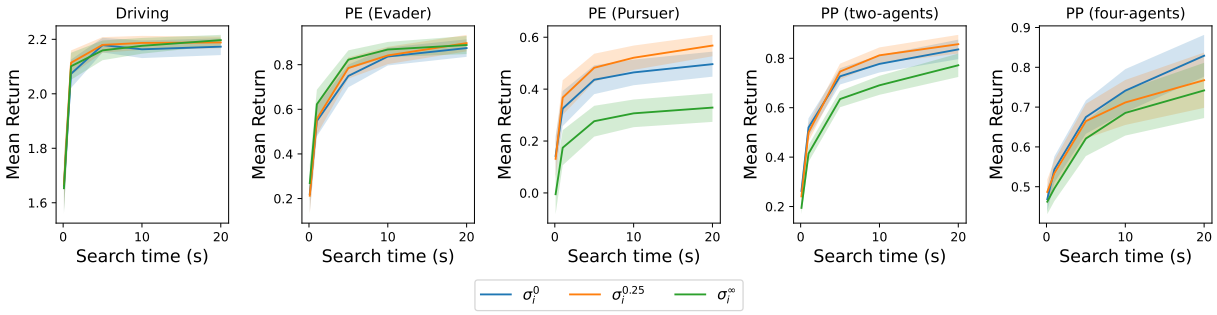


Figure 4.11: Performance of POTMMCP with greedy σ^0, softmax $\sigma^{0.25}$, and uniform σ^∞ meta-policies in each environment. Shaded areas show the 95% CI.

4.C EVALUATION OF DIFFERENT SEARCH POLICIES

Figure 4.12 and Figure 4.13 compares the performance of POTMMCP using different search policies.

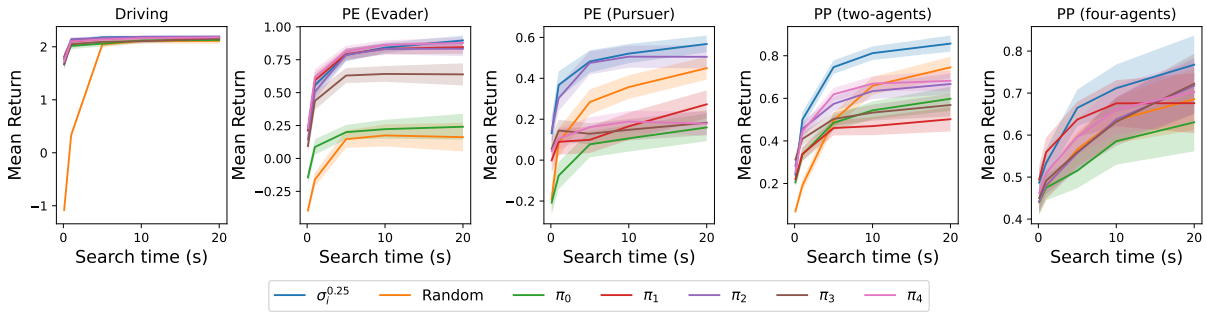


Figure 4.12: Comparison of POTMMCP using different search policies in each environment. Each figure shows performance of POTMMCP using the best performing meta-policy, the uniform random policy, and each of the available fixed policies. Shaded areas show the 95% confidence interval.

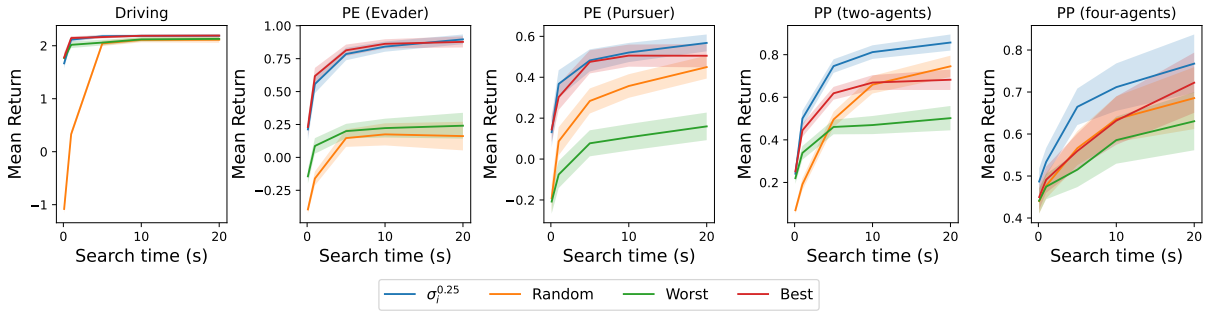


Figure 4.13: Comparison of POTMMCP using different search policies in each environment. Each figure shows performance of POTMMCP using the best performing meta-policy, the uniform random policy, and the best and worst of the available fixed policies (based on their performance given the maximum planning time). Shaded areas show the 95% confidence interval.

5

ON THE BENEFITS OF REINFORCEMENT LEARNING IN PLANNING

This chapter has been published as part of a journal article:

Jonathon Schwartz et al. "POSGGym: A Library for Decision-Theoretic Planning and Learning in Partially Observable, Multi-Agent Environments". In: *Autonomous Agents and Multi-Agent Systems* 39.2 (2025), p. 35. ISSN: 1573-7454. DOI: 10.1007/s10458-025-09716-6

It is also part of a workshop paper:

Jonathon Schwartz et al. "POSGGym: A Library for Decision-Theoretic Planning and Learning in Partially Observable, Multi-Agent Environments". In: *International Conference on Automated Planning and Scheduling PRL Workshop*. 2024

So far in this thesis we have explored algorithms for planning which take a model-driven approach to making optimal decisions in uncertain environments. The theoretical assurances of these types of methods make them appealing for applications that depend on highly reliable autonomous systems. Conversely, learning based methods, such as reinforcement learning (RL), present a data-driven approach for decision-making and have been successfully applied to very large partially observable, multi-agent environments. Seamless integration of planning and RL promises to bring the best of both model-driven and data-driven worlds to multi-agent decision-making, resulting in an approach with assurances on performance that scales well to more complex problems. In this chapter, we empirically investigate existing state-of-the-art planning methods and a method that combines planning and RL in the type-based reasoning setting. Our experiments provide further evidence that combining planning and RL can yield superior performance compared to planning or RL alone, given the model of the environment and other agents is correct. However, our particular setup

also reveals that this integrated approach can result in worse performance than RL alone when the model of other agents is incorrect. Our findings indicate the benefit of integrating planning and RL in partially observable, multi-agent domains, while serving to highlight several important directions for future research.[1]

5.1 INTRODUCTION

Decision-theoretic planning, also known as *planning under uncertainty*, addresses the problem of using a dynamics model to find the optimal way to behave in uncertain environments [Bly99; BDH99]. In scenarios involving a single-agent, this is formalized by the partially observable Markov decision process (POMDP) [SS73; KLC98], which incorporates uncertainty in the state of the environment and the stochastic outcomes of actions. Extending to the multi-agent setting, various approaches exist [SZ08], with the Partially Observable Stochastic Game (POSG) [HBZ04] being one of the most general. The appeal of decision-theoretic planning lies in its mathematical rigor and theoretical guarantees for optimal decision-making under uncertainty, vital for reliable autonomous systems in domains like robotics [Kur22; WW12], autonomous driving [Car+21], and security [SP09a; Ng+10].

Unfortunately, applying planning under uncertainty to large, multi-agent environments has remained a challenge. Existing methods have been consistently limited by the high computational complexity of planning in partially observable multi-agent environments, primarily due to the exponential growth in problem size with planning horizon [Ber+02; SZ08]. Finding novel ways to improve the scaling of planning is essential if it is to be useful for many of the real-world problems we care about solving.

Advances in deep learning [LBH15] offer promising solutions for improving the scaling of planning. Deep learning has been able to effectively leverage growing compute in order to solve increasingly more difficult problems [Kap+20]. Combining deep reinforcement learning (RL) with planning, in the form of search, has already led to remarkable achievements in multi-agent decision making in games such as Go, Chess, Shogi, Poker, and Hanabi [Sil+16; Sil+18; BS18; Bro+20a; Ler+20]. Integrating learning with planning presents an exciting opportunity for the creation of autonomous agents capable of scaling to complex environments while robustly handling uncertainty.

In this chapter we conduct an extensive empirical investigation into current state-of-the-art decision-theoretic planning methods for partially observable, multi-agent environments. We compare these methods with one which combines planning and RL in order to gain insights into the benefits of RL in planning. We evaluate each method across a diverse collection of environments and population of other agents in the

1 Code available at: https://github.com/RDLLab/posggym-baselines

type-based reasoning setting. Importantly, unlike in previous chapters, the evaluation encompasses performance against both known and unknown co-player populations, allowing us to study various properties of each method including generalization to novel partners.

5.2 BACKGROUND

In this chapter we investigate planning and RL agents acting in partially observable, multi-agent environments. In this section we briefly cover the necessary background required for the contributions of this chapter, some of which will be familiar to the reader from previous chapters.

5.2.1 *Partially Observable Stochastic Games*

Partially Observable Stochastic Games (POSG) are a foundational mathematical framework for modeling scenarios involving multiple agents interacting within a stochastic, partially observable environment. They generalize various other formal decision-making models. A Partially Observable Markov Decision Process (POMDP) is a POSG with a single agent [SS73; KLC98], while a decentralized POMDP (Dec-POMDP) [Ber+02] is a fully cooperative POSG where all agents share a reward function. A multi-agent Markov decision processes (MMDP) [Bou96] is a fully cooperative, fully observable POSG. While a Markov game (MG), also known as a stochastic game, is a fully observable POSG [Sha53]. The generality of POSGs has lead to their wide use in decision-theoretic planning and MARL.

Formally, a POSG is a tuple $\mathcal{M} = \langle \mathcal{I}, \mathcal{S}, S_0, \vec{\mathcal{A}}, \vec{\mathcal{O}}, T, Z, R \rangle$ consisting of N agents indexed $\mathcal{I} = \{1, \dots, N\}$, a set of Markov states of the environment $\mathcal{S}$, an initial state distribution $S_0 \in \Delta(\mathcal{S})$[2], the joint action space $\vec{\mathcal{A}} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_N$, the joint observation space $\vec{\mathcal{O}} = \mathcal{O}_1 \times \cdots \times \mathcal{O}_N$, a state transition function $T : \mathcal{S} \times \vec{\mathcal{A}} \times \mathcal{S} \to [0, 1]$ specifying the probability of transitioning to state s' given joint action $\vec{a}$ was performed in state s, the joint observation function $Z : \mathcal{S} \times \vec{\mathcal{A}} \times \vec{\mathcal{O}} \to [0, 1]$ specifying the probability of joint observation $\vec{o}$ after performing joint action $\vec{a}$ and ending up in state s', and a reward function $R : \mathcal{S} \times \vec{\mathcal{A}} \to \mathbb{R}^N$ defining the reward each agent receives when joint action $\vec{a}$ is performed in state s. Combining T, Z, R into a single function produces a generative model $\mathcal{G}$ which returns the next state, joint observation, and joint reward, given the current state and joint action $\langle s', \vec{o}, \vec{r} \rangle \sim \mathcal{G}(s, \vec{a})$.

2 We use S_0 to denote the initial state distribution to distinguish it from the initial belief of an agent b_0 which in the multi-agent setting can be a distribution over more than just environment states.

At each step, each agent $i \in \mathcal{I}$ simultaneously performs an action $a_i \in \mathcal{A}_i$ and receives an observation $o_i \in \mathcal{O}_i$ and reward $r_i \in \mathbb{R}$. Each agent has no direct access to the environment state or knowledge of the other agent's actions and observations. Instead they must rely only on information in their *action-observation history* up to the current time step t: $h_{i,t} = \langle o_{i,0} a_{i,0} o_{i,1} a_{i,1} \dots o_{i,t-1} a_{i,t-1} o_{i,t} \rangle$. The set of all time t histories for agent i is denoted $\mathcal{H}_{i,t}$. Agents select their next action using their *policy* π_i which is a mapping from their history h_i (or belief, see Section 5.2.2) to a probability distribution over their actions, where $\pi_i(a_i|h_i)$ denotes the probability of agent i performing action a_i given history h_i. The goal of each agent i is to maximize its total expected *return* given by $G_i = \mathbb{E}\left[\sum_{t'=t}^{\infty} \gamma^{t'-t} r_{i,t'}\right]$, where $\gamma \in [0, 1)$ is a discount factor. Importantly, each agent's reward at each step depends on the actions taken by the other agents present in the environment.

5.2.2 *Decision-Theoretic Planning*

Decision-theoretic planning is a broad term, in this chapter we consider it as the field of research and methods concerned with finding the optimal way to behave in uncertain environments by explicitly modeling the uncertainty arising from partial observability and stochastic outcomes of actions [KLC98; Bly99; BDH99]. For this reason it is often also referred to as *planning under uncertainty*. In the multi-agent setting, uncertainty about the other agent – their policy, actions, and internal state – must also be considered [SZ08; AS18]. The various methods differ primarily along the axis of whether they aim to find a policy for all agents in the environment [OA16] (e.g. a team of robots), or instead focus on controlling a single agent among independent co-players [GD05] (e.g. a robot that must operate around humans).

In this chapter, we concentrate on the problem of operating a self-interested agent in an environment shared with independent co-players. In cooperative environments this is analogous to the problem of *ad-hoc teamwork* [BM05; Sto+10]. Throughout this paper, we use i to denote the planning agent, $j \neq i$ to denote a single other agent, and $-i = \mathcal{I} \setminus i$ to denote the set of all other agents.

Central to planning is the use of dynamics models of the environment. We assume the agent has access to a generative model $\mathcal{G}$ of the underlying POSG, which it can use to sample next states, observations, and rewards. $\mathcal{G}$ is stochastic in nature, arising from the uncertainty in the environment dynamics, where multiple possible outcomes may be associated with each action, observation, and state transition. This is compared to having access to the full model which provides the full transition and observation probabilities and generally can be impractical to implement for environments with large state and observation spaces, due to the need to compute distributions over very

large - or even infinite in the case of continuous environments - spaces.. Additionally, due to partial observability, the agent does not have direct access to the current state of the environment. Instead, it needs to maintain a belief state, representing a probability distribution over possible states based on past observations and actions.

5.2.2.1 *Beliefs*

At the core of planning under uncertainty lies an agent's *belief*, representing a distribution over possible states the agent could currently be in. Importantly, what constitutes a state varies depending on the context. In single-agent POMDPs, the belief b is a distribution over environment states $b \in \Delta(\mathcal{S})$ and encapsulates uncertainty in the physical state of the environment $s \in \mathcal{S}$. In the multi-agent setting, the policies and internal states of the other agents must also be considered. Multi-agent frameworks differ based on how they represent the other agents' policies and internal states. For instance, the Interactive-POMDP (I-POMDP) [GD05] adopts a recursive reasoning approach, modeling other agents' policies as solutions to lower-level I-POMDPs and their internal state as lower level beliefs over environment states and possibly the beliefs of the other agents. More generally, it suffices to consider the space of possible policies $\pi_{-i} \in \Pi_{-i}$ and action-observation histories $h_{-i,t} \in \mathcal{H}_{-i,t}$ of the other agents [HBZ04; GD05], where the space of policies could be a discrete set of policies [HBZ04; GD05] or a distribution over continuous parameters [AS17]. For the purpose of this chapter, a belief $b_{i,t} \in \Delta(\mathcal{S} \times \Pi_{-i} \times \mathcal{H}_{-i,t})$ encompasses the environment state $s \in S$, other agents' policies $\pi_{-i} \in \Pi_{-i}$, and their histories $h_{-i,t} \in \mathcal{H}_{-i,t}$, where the other agents' policy space is a discrete set.

The planning agent uses its belief $b_{i,t}$ to select its next action according to its policy π_i and then updates its belief given the next observation from the environment. The initial belief $b_{i,0}$ is formed based on the initial environment state distribution and prior ρ over other agents' policies: $b_{i,0}(s, \pi_{-i}, h_{-i,0}) = S_0(s)\rho(\pi_{-i})$. In this chapter ρ is a uniform distribution, unless otherwise stated. Subsequent beliefs are computed based on the agent's most recent action $a_{i,t}$ and observation $o_{i,t}$ using the well-known Bayes filter: $b_{i,t+1}(s_{t+1}, \pi_{-i,t+1}, h_{-i,t+1}) = Pr(s_{t+1}, \pi_{-i,t+1}, h_{-i,t+1} | b_{i,t}, a_{i,t}, o_{i,t})$. Exact representation and computation of beliefs are intractable for all but small problems due to the curse of state dimensionality. Thus, modern planners rely on approximate methods.

5.2.2.2 *Monte Carlo Planning*

Monte Carlo planning using Monte Carlo Tree Search (MCTS) [Cou06; SV10] and particle filters [RGT05] is the dominant paradigm for planning in large partially observable environments. Rather than maintaining exact beliefs, beliefs are approximated using a

particle filter with Monte Carlo updates. To compute the policy for the current belief, MCTS constructs a search tree of nodes online, where each node is a belief, effectively searching over the space of beliefs.

MCTS involves estimating the value of each action from the current belief via a series of simulated episodes. Each simulation starts from a state sampled from the current belief and proceeds in three stages: selection, expansion, and backpropagation.

1. **Selection:** the current search tree is traversed using an *exploration policy* until a leaf node of the tree is reached. The exploration policy balances exploring new regions of the search space with exploiting known high value regions. The two commonly used exploration policies are UCB (Upper Confidence Bound) [ACF02], and PUCB ("Predictor" + UCB) [Ros11]. UCB selects actions as a function of the action's current estimated value and number of times it has been previously selected, while PUCB extends this to include an additional bias based on a *search policy*.

2. **Expansion:** upon reaching a leaf node, the node is evaluated and expanded by adding it to the tree and adding edges for each action. Evaluation involves estimating the node's value. This is typically done through Monte Carlo rollouts or using a pre-computed value function.

3. **Backpropagation**: the nodes and edges visited along the simulated trajectory are updated by propagating the value estimate from the leaf node back-up to the root node of the tree. This includes incrementing the visit count for each node, for use by the exploration policy.

Upon completion of the search, the planning agent selects the action from the root node based on visits and values, typically by choosing the action with the largest value, most visits, or sampling from a distribution computed from these values. Following the action being performed in the environment and the planning agent receiving its next observation, the current belief of the agent is updated and set as the new root node of the tree. The search process is then repeated.

5.2.3 *Reinforcement Learning*

Reinforcement Learning (RL) [SB18] is another approach for solving sequential-decision making problems. Whereas planning uses a dynamics model and explicit beliefs, RL focuses on learning a policy through interactions with the environment or simulations in a model. Deep RL in partially observable environments typically circumvent explicit beliefs by using recurrent networks (RNNs) [BSF94] such as LSTMs [HS97] to

represent the policy [HS15]. These RNNs can learn implicit beliefs, in that they learn representations of the sufficient statistics of the state of the world given the agents action-observation history. However, these learned implicit beliefs do not permit planning since search in planning requires explicitly sampling from beliefs in order to simulate possible future trajectories. Instead, prior work has combined RL with search by using a policy trained with deep RL as the *search policy* within MCTS using PUCB [Sil+16; Sil+18; BS18; Ler+20].

An important question when applying RL in multi-agent environments is how to model the other agents in the environment? This question has lead to the development of a range of multi-agent training schemes. One of the most well known is *self-play* [HLS15] where the RL agent plays against itself, using the same policy for all agents in the environment. Self-play has proven very effective, especially for two-player, zero-sum games [Tes94; Sil+16; Sil+18; BS19], or settings where there is centralized learning for decentralized execution [Ler+20]. While the results using self-play have been impressive, there is evidence that purely self-play agents can be prone to over-fitting, resulting in exploitable agents [Lan+17; Gle+19].

Another approach is *population-based training* where multiple independent policies are trained simultaneously and paired up according to some schema [Lan+17; Jad+19; Tea+21]. This type of training has been used to generate superhuman performance in large, complex, partially observable environments such as StarCraft 2 [Vin+19] and Dota [Ber+19]. Over the years various schemas have been proposed including those based on cognitive hierarchies and nested reasoning [Lan+17; Cui+21], playing against older versions of a policy [HLS15; Ber+19], and using meta-game solvers [Lan+17].

Finally, other methods instead use a fixed model of the opponent, typically either learned from data or handcrafted [He+16]. A good example of this is to generate a model of human behavior from data and then use RL to learn a policy that acts well with respect to this model. This technique was crucial for generating the initial policies for superhuman agents in competitive domains like Go [Sil+16] and Starcraft 2 [Vin+19]. More, recently it has been used to produce human level play against humans in the game Diplomacy [Bak+22], which requires both cooperation and competition. This is the approach we use in this chapter, where the RL agent has access to a set of possible policies for the other agent during training, and tasked with learning a best-response (BR) to this set.

5.3 RELATED WORK

5.3.1 *Multi-Agent Planning*

In this chapter we empirically evaluate existing state-of-the-art methods for planning under uncertainty in large partially observable environments where the task is to control a single agent among other independent agents. A closely related problem is that of *ad-hoc teamwork* [Sto+10] in cooperative environments where the goal is to design an agent that can adapt to novel teammates. Proposed methods in this area include those based on stage games [WZC11], Bayesian beliefs [BSK11; Bar+14], types with parameters [AS17], and for the many agent setting [You+18]. All these methods use MCTS but are limited to environments where the state and actions of the other agents are fully observed, so are not applicable in our settings. For the more general setting of type-based reasoning, earlier in this thesis (Chapter 4) we introduced POTMMCP that incorporates a meta-policy for guiding search. POTMMCP was shown to outperform related methods across a range of cooperative, competitive, and mixed environments. A number of other works have focused on planning in more restricted settings. This includes strictly cooperative [CO21; Cho+22] and competitive [CPW12] environments, and settings where there is centralized control [AO14].

Several Monte Carlo planning methods have been proposed for solving large I-POMDPs. This includes methods based on finite-state automata [PG17] and for systems with communication [Kak+22]. However, most relevant for our setting are IPOMCP [Eck+20], which extends the single-agent POMCP [SV10] to solving I-POMDPS, and INTMCP (Chapter 3) which uses nested-MCTS. These two methods represent the current state-of-the-art planning methods when it comes to solving large, general I-POMDPs.

5.3.2 *Combined Planning and Learning*

Combining search with RL has been an important component for superhuman performance in games. Self-play RL and MCTS have been combined to achieve beyond expert performance in two-player fully-observable zero-sum games with both a known [Sil+16; Sil+18] and learned [Sch+20] environment model. Similar methods have been applied to zero-sum imperfect-information games [BS18; BS19; Bro+20a; Tim+22], as well as cooperative games where there is prior coordination for decentralized execution [Ler+20; Hu+21]. Methods combining MCTS and RL in games for training a best-response policy against a distribution over policies have also been proposed [Li+23]. A key focus of existing works combining RL and search has been for games which

are fully-observable, where the agent has access to exact *information-states*, or using beliefs representations trained using specific domain knowledge. To the best of the authors knowledge ours is the first work to do a comprehensive empirical investigation combining RL with existing methods for planning under uncertainty based on Monte Carlo planning with particle filters.

5.4 EXPERIMENTS

We empirically evaluated planning, RL, and combined planning plus RL methods across diverse environments. The goal of our experiments was to investigate current state-of-the-art planning under uncertainty methods and compare these with combining RL with planning. Our experiments serve an additional purpose of providing baseline results and algorithm implementations for to facilitate future research[3].

Some questions we hoped to answer in our experiments:

- How does the performance of planning and RL methods compare across various settings?
- How well does each method generalize when faced with novel partners?
- How effective is combining existing planning methods with RL?

5.4.1 *Experiment Setup*

In our experiments we control a single planning agent in an environment with one other agent with unknown behavior. We focus on the *type-based reasoning* [ACR16] setting where the planning agent maintains a belief over a set of possible policies P, or "types", for the other agent. The goal of the planning agent is to choose actions that maximize its own total expected reward given its beliefs about the environment and the policy and internal state of the other agent. We make no assumptions about the reward structure of the environment and test across competitive, cooperative, and mixed incentive domains.

The high-level experimental design is shown in Figure 5.1. For each environment there is a set of agent populations $\mathcal{P}$, where $P \in \mathcal{P}$ is a population containing a set of possible policies for the other agent. The planning agent is given access to the model of the environment as well as a known population of policies $P_{\text{plan}} \in \mathcal{P}$ which can be used for planning and/or training. We assess each method against a separate population of policies $P_{\text{test}} \in \mathcal{P}$, which may be the same or different to the planning population.

3 Algorithm implementations and experiment code is available at: https://github.com/RDLLab/posggym-baselines

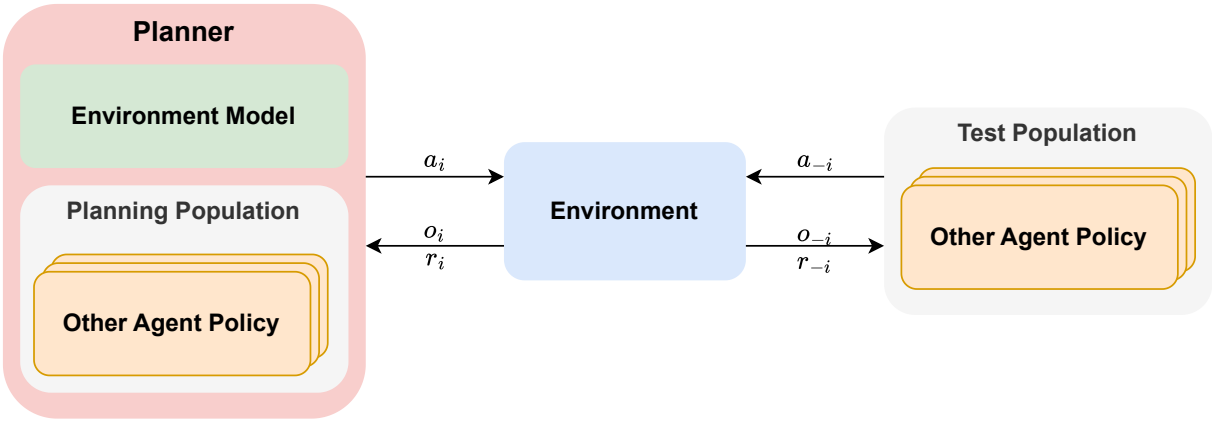


Figure 5.1: High level experimental setup. During planning/training, each planning and RL method has access to the environment model and a planning population of policies for the other agent P_{plan}. Each method is then evaluated against a separate, possibly different population of policies P_{test}. In our experiments there are two populations P_0 and P_1 for each environment, each containing between five and six policies, which are used for P_{plan} and P_{test}.

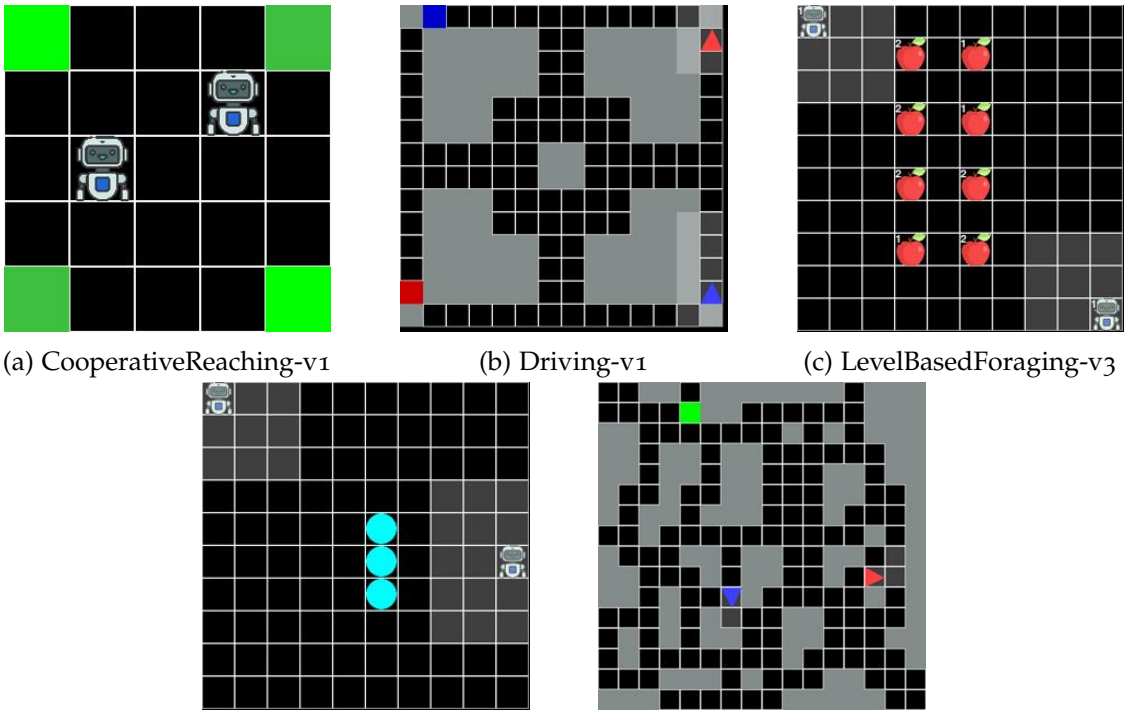
Experiments where the planning and test populations are the same, $P_{\text{plan}} = P_{\text{test}}$, are referred to as *in-distribution*. Conversely, experiments where $P_{\text{plan}} \neq P_{\text{test}}$ are referred to as *out-of-distribution*. By comparing in- and out-of-distribution performance we are able to investigate each method's ability to generalize to novel co-players.

For each environment we employ two distinct populations, denoted as P_0 and P_1 with $\mathcal{P} = \{P_0, P_1\}$. These are used for both the planning P_{plan} and test P_{test} populations. We conduct our experiments for each method by iterating over combinations of planning and test populations, $\langle P_{\text{plan}}, P_{\text{test}} \rangle \in \mathcal{P} \times \mathcal{P}$, resulting in four repetitions of each experiment per algorithm (each with a different $\langle P_{\text{plan}}, P_{\text{test}} \rangle$ combination). For every population $P \in \mathcal{P}$ we used a uniform distribution over policies as the prior, meaning for each episode every policy in the set P had equal probability of being used by the other agent.

A total of 10 to 12 policies were generated for the populations $\mathcal{P}$ in each environment. Depending on the environment the set was made up of heuristic, learned, or a mix of heuristic and learned policies. The 10 to 12 policies were divided roughly equally into the two populations P_0 and P_1 with each containing five to six policies. More details about the specific policies used in each population are provided in Appendix 5.A.

5.4.2 *Experiment Environments*

We tested each method on a range of environments. This included: Cooperative-Reaching (cooperative) [Rah+23], Driving (mixed) [McK+22; LP19], Level-Based Foraging (mixed) [CSA20; Pap+21], Pursuit-Evasion (competitive) [SvW18; SZK22], Predator-



(a) CooperativeReaching-v1 (b) Driving-v1 (c) LevelBasedForaging-v3

(d) PredatorPrey-v0 (e) PursuitEvasion-v1

Figure 5.2: The environments used in our experiments. We used a diverse set of environments, including cooperative, mixed, and competitive scenarios. Since it is asymmetric we use two versions of PursuitEvasion-v1: *i0* and *i1* where planner controls the pursuer (blue) and evader (red), respectively.

Prey (cooperative) [Tan93; Lei+17]. These environments have all been previously studied in the RL and planning literature and cover cooperative, mixed, and competitive settings, as well as a range of multi-agent concepts (see Chapter 6, Table 6.2). We limited the selection to environment with discrete actions and observations, as this was what the planning methods in our experiments were designed for. The size of the various components of each environment are shown in Table 5.1.

Table 5.1: Environment properties. $|S_0|$ denotes the number of environment states in the planning agent's initial belief which have non-zero support given the agent's initial observation. $|P_0|$ and $|P_1|$ are the number of policies within agent population.

Environment	Reward Type	$\lvert\mathcal{S}\rvert$	$\lvert S_0\rvert$	$\lvert\mathcal{A}_i\rvert$	$\lvert\mathcal{O}_i\rvert$	$\lvert P_0\rvert$	$\lvert P_1\rvert$
CooperativeReaching-v1	Cooperative	625	1	5	625	5	6
Driving-v1	Mixed	8.9×10^{12}	9	5	6.6×10^{6}	5	5
LevelBasedForaging-v3	Mixed	1.5×10^{11}	5×10^{6}	6	30	5	5
PredatorPrey-v0	Cooperative	7.2×10^{10}	64	5	100	5	6
PursuitEvasion-v1_i0	Competitive	2.8×10^{8}	3	4	768	6	6
PursuitEvasion-v1_i1	Competitive	2.8×10^{8}	1	4	4608	6	6

5.4.3 *Planning Methods*

We focus on planning methods designed for controlling a single agent within a general, multi-agent environment. We do not include methods designed for environments with specific structures, e.g. with communication [Kak+22], centralized control [AO14; Cho+22; CO21] or full observability [You+18]. Furthermore, since we are interested in studying planning in large, complex environments, restricting us to methods that are capable of handling environments with millions of states. In practice, this limits us to online planning approaches based on MCTS [Cou06] as currently these have proven to be the most scalable, general methods.

For our experiments we compare the following planning methods:

- **Interactive Nested Tree Monte Carlo Planning** (INTMCP) [SZK22]: Models the environment as an I-POMDP and uses nested-MCTS to solve the I-POMDP at each reasoning level online. We use a reasoning level of $l = 2$ for our experiments. Uses UCB for the exploration policy and Monte Carlo rollouts for node evaluation. Notably, it does not incorporate P_{plan} into its decision-making, instead relying on a recursive I-POMDP model for the other agent.

- **Interactive Partially Observable Monte Carlo Planning** (IPOMCP) [Eck+20; Kak+22]: Models the environment as an I-POMDP with uncertainty over the other agents internal state (history and policy). Uses P_{plan} for modeling the other agent[4]. Uses UCB for the exploration policy and Monte Carlo rollouts for node evaluation

- **Partially Observable Monte Carlo Planning** (POMCP) [SV10]: Models the environment as a POMDP with the other agent's actions selected uniformly at random. This approach acts as a baseline since it uses naive modeling for the other agent. Uses UCB for the exploration policy and Monte Carlo rollouts for node evaluation

- **Partially Observable Type-based Meta Monte Carlo Planning** (POTMMCP) [SKH23]: Similar to IPOMCP except the set of other agent policies are additionally utilized to inform planning via a meta-policy. Uses PUCB for the exploration policy with the meta-policy as the search policy. Uses Monte Carlo rollouts and pre-computed value function for node evaluations, depending on the representation used for the other agent policies in P_{plan}.

A high-level comparison of the key differences between each planning algorithm is shown Table 5.2. Noting that both INTMCP and POMCP utilize their own model

4 In the original IPOMCP[Eck+20] and CIPOMCP[Kak+22] papers a single policy for the other agent policy was generated using an I-POMDP solver, here we use the policies from the planning population.

for the other agent and thus do not incorporate the planning population into their decision-making. For these two methods we look only at their out-of-distribution performance, since their internal models are always different from the test population. Our experiments help evaluate how beneficial the inductive biases of these two methods are.

Table 5.2: Comparison of the components of the planning and combined algorithms used in our experiments. $\mathcal{U}(P_{\text{plan}})$ in the Other Agent Model column indicates the method models the other agent using a uniform distribution over the planning population of policies P_{plan}.

Algorithm	Search Policy	Exploration Policy	Other Agent Model
INTMCP [SZK22]	Random	UCB	Nested MCTS
IPOMCP [Eck+20; Kak+22]	Random	UCB	$\mathcal{U}(P_{\text{plan}})$
POMCP [SV10]	Random	UCB	Random
POTMMCP [SKH23]	Meta-Policy over P_{plan}	PUCB	$\mathcal{U}(P_{\text{plan}})$
COMBINED	π_{BR}	PUCB	$\mathcal{U}(P_{\text{plan}})$

We tested each method across a range of planning budgets $S \in [0.1, 1, 5, 10, 20]$, where S represents seconds of search time per step. The hyperparameters used by each method are shown in Table 5.3. To reduce the amount of hyperparameter tuning required, for all methods we we used normalized Q-values during planning as per Schrittwieser et al. [Sch+20]. For methods that used UCB (INTMCP, IPOMCP, POMCP) we used rollouts using a random policy for leaf node evaluations. For INTMCP and POMCP actions for the other agent during rollouts were chosen using a random policy, while for IPOMCP they were chosen using the other agent policy sampled from the root belief. POTMMCP used the value function from its meta-search policy for leaf node evaluation where available, otherwise used rollouts using the meta-policy for action selection. For all methods we used rejection sampling for belief reinvigoration after each update. This was used over other methods such as weighted particle filtering [SK18] as it did not require access to an observation function which can be hard to generate or expensive to compute in large environments.

5.4.4 *Learning Method*

For the learning method we train a single deep RL policy as a best-response against each other agent population $P_k \in [P_0, P_1]$ in each environment. We refer to the learning method as *Reinforcement Learning-Best Response* (RL-BR). Proximal Policy Optimization (PPO) [Sch+17] was used as the specific RL algorithm since it is used extensively for

Table 5.3: Hyperparameters for the planning methods used in our experiments.

Hyperparameter	Value
Per step search time (S)	$[0.1, 1, 5, 10, 20]$
Discount (γ)	0.99
Discount horizon (ϵ)	0.01
Belief particles	$\lceil 100 \times S \times 1.16 \rceil$
C_{PUCB}	1.25
PUCB exploration (λ)	0.25
C_{UCB}	$\sqrt{2}$

MARL research [Ber+19; Bak+20; Yu+22] and worked well across all environments we tested.

In each environment a separate RL-BR policy $\pi_{BR,k}$ was trained for each population $P_k \in \mathcal{P}$. During training, at the start of each episode a policy for the other agent π_{-i} was sampled from a uniform distribution over the planning population $\pi_{-i} \sim \mathcal{U}(P_k)$ and this policy was used to select actions for the other agent $-i$ while actions for the ego agent i were sampled from the BR policy $\pi_{BR,k}$. By training to maximize the planning agent's expected return against the uniform mixture over the planning population $\mathcal{U}(P_k)$, each policy policy $\pi_{BR,k}$ was trained to be a best-response to $\mathcal{U}(P_k)$.

Each RL-BR policy was represented using a neural network. The specific neural network architecture consisted of a fully-connected network (FCN) trunk, followed by a single layer LSTM [HS97], and then separate FCN actor and critic heads. The hyperparameters used for training each RL-BR policy using PPO are shown in Table 5.4.

To ensure reproducibility in our results, we trained five versions of the same policy using different random seeds for each planning population in each environment. Each individual RL-BR policy was trained until convergence as indicated by their learning curve as shown in Figure 5.3.

5.4.5 *Combined Learning and Planning Method*

The combined planning and learning method incorporates the RL-BR policies from the previous section as a search policy within an MCTS based planner. POTMMCP [SKH23] was used as the base planner with the search policy (normally the meta-policy) replaced with the RL-BR policy. We refer to this method simply as *Combined*. The value function from the trained RL-BR policy was used for leaf node evaluation in all environments, instead of Monte Carlo rollouts. A side-by-side comparison of the Combined method's properties with those of the other planning methods is shown in Table 5.2.

Combining a policy trained via RL with MCTS in this way has been applied in a number previous works in varying ways [Sil+16; Sil+18; BS18; BS19; Fic+21]. Prior work

Table 5.4: Training hyperparameters for RL-BR policies used in our experiments.

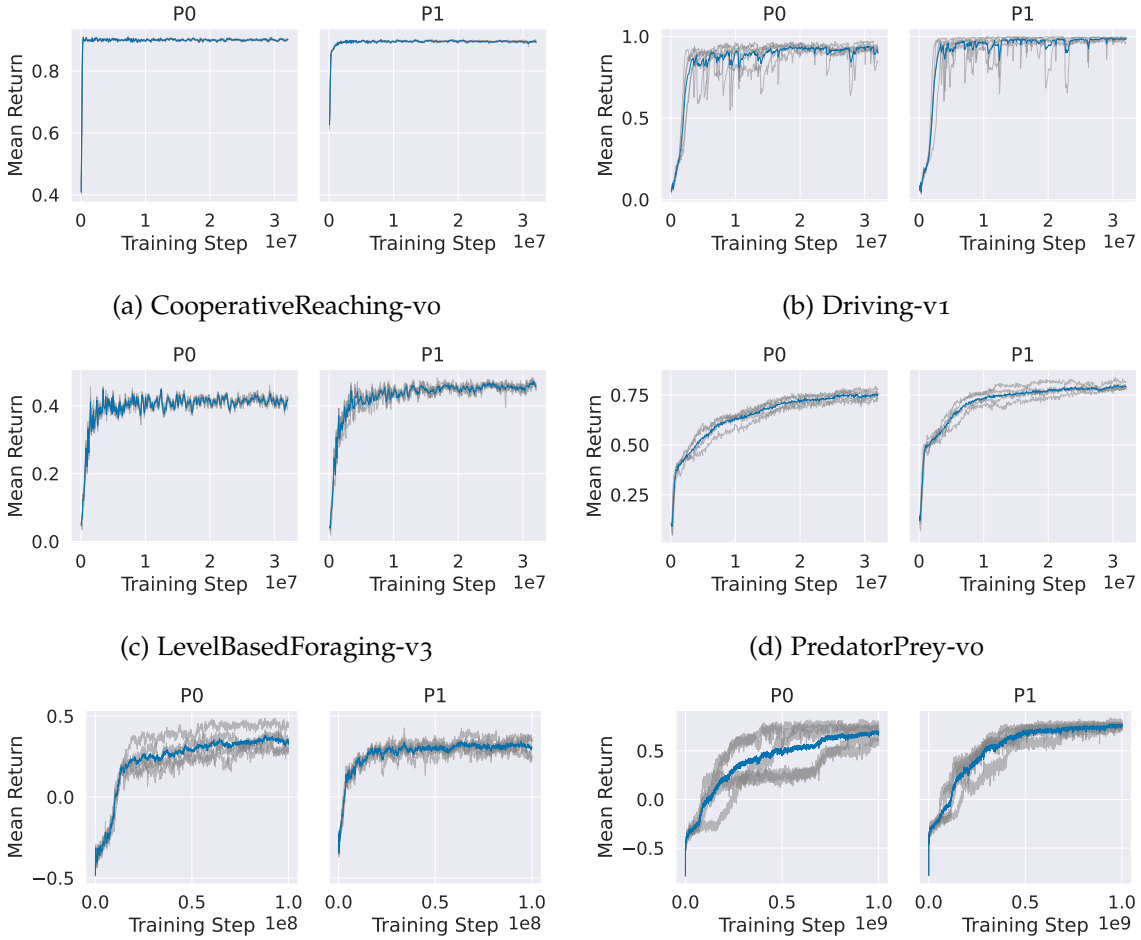
Hyperparameter	Value
Training steps	100M (PursuitEvasion-v1 i0) 1B (PursuitEvasion-v1 i1) 32M (all other environments)
Parallel workers	32
Trunk layer sizes	[64, 64]
LSTM size	64
Head layer sizes	[64]
Discount (γ)	0.99
Learning rate	$3 \times 10^{-}4$
GAE λ	0.95
Batch size	65536
Mini-batch size	2048
Rollout horizon	64
Update epochs	2
BPTT sequence length	10
Entropy bonus	0.01
Value function coeff.	0.5
Clip parameter	0.2
Global gradient clipping	10

has focused on settings where the environment is fully-observable or there is access to information-states. Here we investigate combining RL plus planning using particle based beliefs. Using particle based beliefs is the dominant paradigm for planning under uncertainty in large environments. The research presented in this chapter is the first, to the best of the authors knowledge, to provide a comprehensive evaluation of combining planning with RL using particle based beliefs in large multi-agent environments.

For all combined method experiments we follow the same experiment protocol used for the planning methods, testing across a range of planning budgets $S \in [0.1, 1, 5, 10, 20]$. We also repeat each experiment using each of the five RL-BR policies trained for each environment and planning population.

5.5 RESULTS

For the results, we start with the key high-level findings looking at performance when averaged across the entire set of environments. This gives us a view of how each method compares over a broad distribution. Following this, we take a deeper dive into the results of each environment separately. Importantly, this allows us to gain some insights into where the methods do well, where they do not, and why.



(a) CooperativeReaching-v0

(b) Driving-v1

(c) LevelBasedForaging-v3

(d) PredatorPrey-v0

(e) PursuitEvasion-v1 i0 (Pursuer)

(f) PursuitEvasion-v1 i1 (Evader)

Figure 5.3: Learning curves for the RL-BR policy in each environment against each policy population P_0, P_1. Grey lines show the mean episode return throughout training for each of the five different seeds. The blue line shows the average across seeds.

5.5.1 *Overall Results*

Figure 5.4 shows in- and out-of-distribution performance of each method averaged across environments and planning populations. To ensure all environments are weighted equally, we normalize the returns to be within $[0, 1]$, so that 0 and 1 correspond to the minimum and maximum possible return within each environment, respectively. Below we discuss some of the key takeaways from these results.

FOR THE IN-DISTRIBUTION SETTING, COMBINING PLANNING AND LEARNING IMPROVES ON EITHER APPROACH ALONE, GIVEN ENOUGH PLANNING TIME. We observe the benefits of incorporating planning alongside a trained RL policy when provided with an accurate model of the world and other agents. This finding aligns with previous research on combining search and RL in two-player, zero-sum games

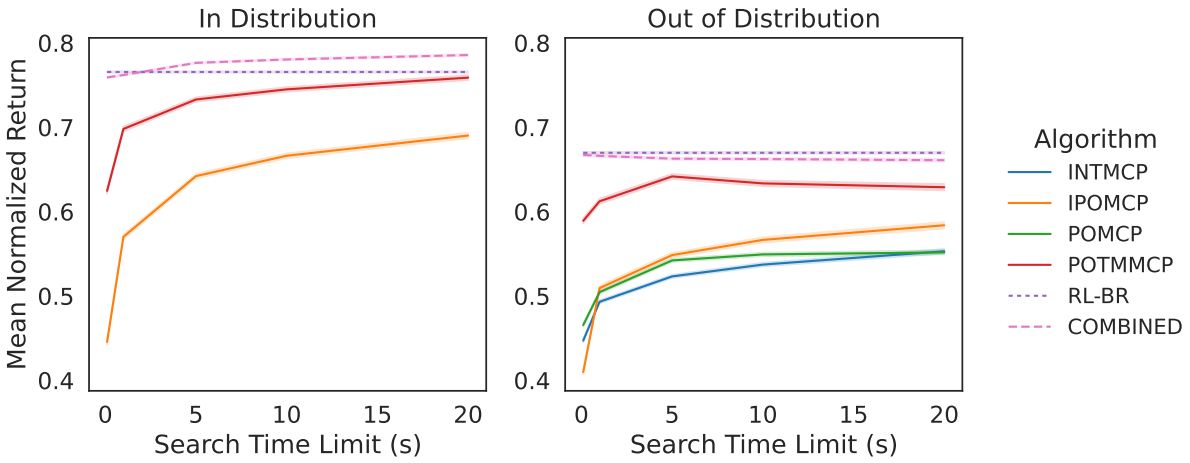


Figure 5.4: In- (left) and out-of-distribution (right) performance of each method averaged across all environments for different planning budgets (x-axis). Each plot shows the mean return normalized to the interval $[0, 1]$ from the $[\min, \max]$ possible returns for each environment. In-distribution shows results for when the planning and test populations are the same, out-of-distribution shows results when the populations are different. We find that combined planning and learning leads to improvements over each approach alone, but only in the in-distribution setting where the agent has an accurate model of the environment and the other agent. Shaded areas show 95% confidence intervals.

[Sil+16; Sil+18; Bro+20a] and cooperative games [Ler+20; Hu+21]. Unlike previous studies that utilize exact or learned belief models with factored public and private observations, here we show that combining a RL policy with a planner is also effective when using particle based beliefs. We believe this is a promising result for efforts to scale up planning to more complex domains, showing that existing particle based planning methods can benefit from an RL trained policy, and vice versa.

Conversely, the relative gain in performance of the combined method does not occur in the out-of-distribution setting where the model of the other agent is inaccurate. In fact, performance appears to degrade slightly with increased search time. We discuss some of the underlying factors for this in subsequent sections.

THE LEARNING METHOD (RL-BR) OUTPERFORMS ALL PURE PLANNING METHODS IN BOTH IN- AND OUT-OF-DISTRIBUTION SETTINGS. This demonstrates a general benefit of RL over pure planning within our experimental setup. However, the RL performance does not come without some cost, requiring a larger amount of offline compute for training –up to 48 hours on 32 CPUs and 1 GPU per policy– compared to the online planners, which utilize no offline compute in our experiments. Nonetheless, RL policies offer much faster per-step time during execution ($\sim$ 0.1s compared to up to 20s for the planning methods at maximum search budget). Our observations highlight a key advantage of modern deep RL based methods, namely their ability to effectively leverage large amounts of compute for training policies offline. In comparison, existing

offline planning methods quickly run into memory limits for the size of problem we run in our experiments.

Looking a bit closer at the results, for the in-distribution setting we see planning methods exhibit improved performance with longer search time, with POTMMCP matching RL-BR's performance with 20 s of search time. We observe that planning methods tends to converge towards optimality in environments where they posses a perfect model of the environment and the other agent. In the next section (Section 5.5.2) we will see that in some specific environments, planning methods actually outperform RL-BR.

The most significant performance gap between learning and planning occurs in the out-of-distribution setting. Here, planning methods are prone to over-fitting to an inaccurate model of the other agent, leading to erroneous beliefs and subsequently policies that do not necessarily improve with more planning. We discuss belief error during planning in greater detail in Section 5.5.3. Conversely, the RL-BR method likely benefits from its neural network policy representation, offering some degree of generalization to out-of-distribution settings. RL-BR also does not suffer due to belief approximation error stemming from Monte Carlo belief updates.

MONOTONIC IMPROVEMENT IN PERFORMANCE WITH SEARCH TIME IS EVIDENT FOR ALL PLANNING AND COMBINED METHODS IN THE IN-DISTRIBUTION SETTING, CONTRASTING WITH THE OUT-OF-DISTRIBUTION SETTING. While accurate models of the environment and the other agent lead to performance gain with increased search time, this trend does not hold uniformly in multi-agent settings. Notably, when the test population differs from the planning population, we see performance plateau or even decline with planning budget. Further exploration into the underlying cause of this discrepancy is discussed in Section 5.5.3.

A LARGE GAP EXISTS BETWEEN IN- AND OUT-OF-DISTRIBUTION PERFORMANCE ACROSS ALL METHODS. Figure 5.5 clearly illustrates this contrast in performance, highlighting the impact of inaccurate other agent models, regardless of whether methods are learning or planning based, or a combination of both. This observation indicates the general brittleness of the tested methods when encountering novel partners.

It is worth noting that our results are influenced by the specific test populations used. Adjusting the planning population, such as a larger or more diverse population based on some diversity metric, could potentially narrow this performance gap. The design of populations to enhance the robustness of autonomous agents in multi-agent settings is an active area of research [Lan+17; Lup+21; Xin+21; Rah+23], and our finding emphasize its significance for both planning and RL.

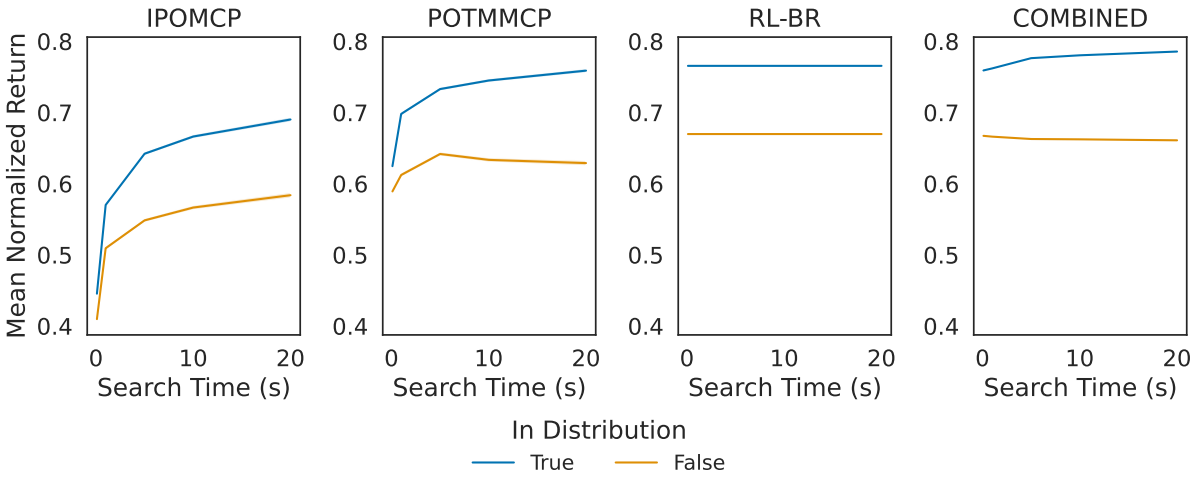


Figure 5.5: In- versus out-of-distribution mean normalized returns for each method averaged across environments. The x-axis shows planning budget. Note, RL-BR does not use search and thus no planning budget, so for consistency and ease of comparison we display mean performance achieved as horizontal line.

5.5.2 *In-Distribution Performance*

Here we analyze the in-distribution performance in each environment individually. The primary findings are presented in Figure 5.6. Contrary to the overall results, we observe more nuanced trends at the individual environment level.

IN SOME CASES, COMBINING PLANNING AND LEARNING CAN LEAD TO DETERIORATING PERFORMANCE AS THE PLANNING BUDGET INCREASES. Specifically, in the Level-Based Foraging environment, the performance of the combined method decreases with longer search times. This unexpected outcome is notable because the combined method has access to an accurate model of the environment and the other agent, so we expect increasing performance with search time, as observed in all other environments. The result highlights a limitation of current particle-based planners, namely the introduction of error due to poor belief approximation.

From Table 5.1, we observe that for the Level-Based Foraging environment the size of the initial state distribution is significantly larger than for any other environment we tested with $|S_0| = 5.0 \times 10^6$. In contrast, in our experiments the number of particles each planning method uses to represent its initial belief is at most 2320 (at the maximum planning budget). This is more than double what has been used in prior works on multi-agent decision theoretic planning, [Eck+20; Kak+22], yet is still only a fraction of the initial belief size for Level-Based Foraging. To make matters worse, in Level-Based Foraging many state features remain constant throughout an episode following initialization. Since all our planning methods rely on Bayesian updates for the beliefs, if the true value of a state features is not within the initial belief, the

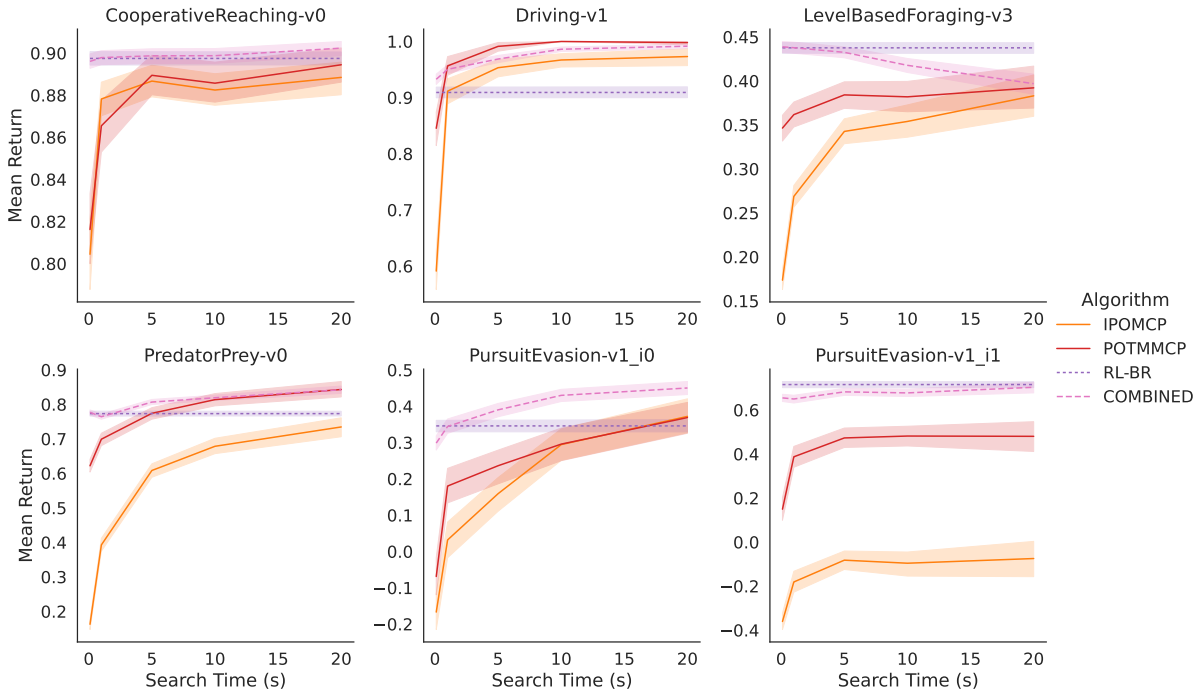


Figure 5.6: In-distribution performance of each method in each environment. The plots show the mean return when the planning and test populations are the same across planning budgets (x-axis), with results averaged over the two experiment populations. Shaded areas show 95% confidence intervals.

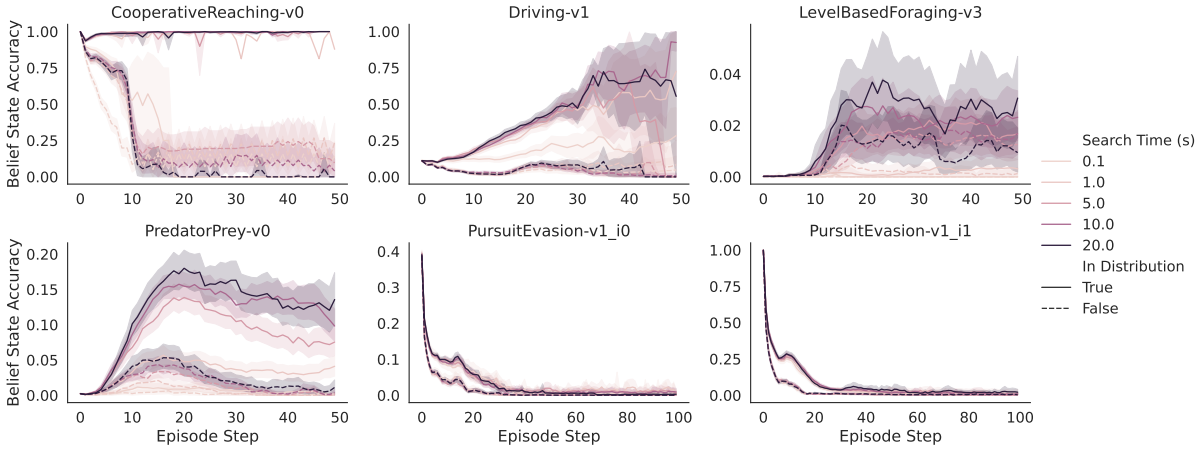


Figure 5.7: Probability assigned by the combined method's belief to the true environment state during an episode. In general belief accuracy is significantly worse in the out-of-distribution setting, and also in the LevelBasedForaging-v3 environment where initial belief space density is largest (y-axis scales differ between plots).

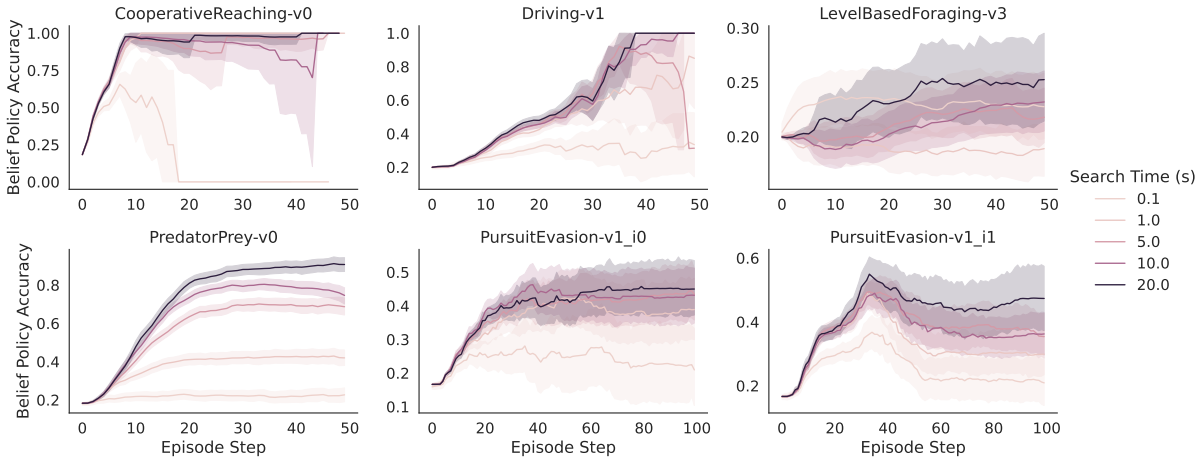


Figure 5.8: Probability assigned by the combined method's belief to the true policy of the other agent throughout an episode in the in-distribution setting. In general belief accuracy increases with search budget, and as the episode progresses.

planner will never be aware of it, even with unlimited planning budget post-initial belief. This explanation is supported by examining the probability assigned to the true environment state by the Combined agent's belief (Figure 5.7), revealing a near-zero probability initially and reaching at most < 0.04. This is significantly lower than the belief accuracy during an episode than in any of the other environments.

Poor belief accuracy leads to worsening performance as the planning budget increases due to how PUCB, which is used in the Combined method, operates. Initially, action selection at the root of the tree is heavily biased by the search policy, in this case the RL-BR policy. However, as more time is spent searching, the visit counts of each action come to dominate, leading to less bias towards the search policy. Consequently, as planning time increases, the policy at the root of the tree diverges away from the search policy and towards the optimal action based on the environment model and agent's belief. In environments like Level-Based Foraging, where belief accuracy is problematic, this means the agent biases more towards a policy based on an incorrect belief.

Continuing to increase planning time might eventually lead to performance improvement as belief approximation improves. However, this would require using significantly more particles for the belief representation, quickly becoming impractical for even larger environments. Our results in the Level-Based Foraging environment underscores a key limitation of current methods for belief-based planning. They either work only for relatively sparse belief spaces or require domain-specific knowledge to compute the true belief state [Ler+20] or learn a good approximation [Hu+21; Li+23]. Our result suggests that novel, general methods for producing robust approximate beliefs is an important direction for future planning under uncertainty research looking to scale to larger environments.

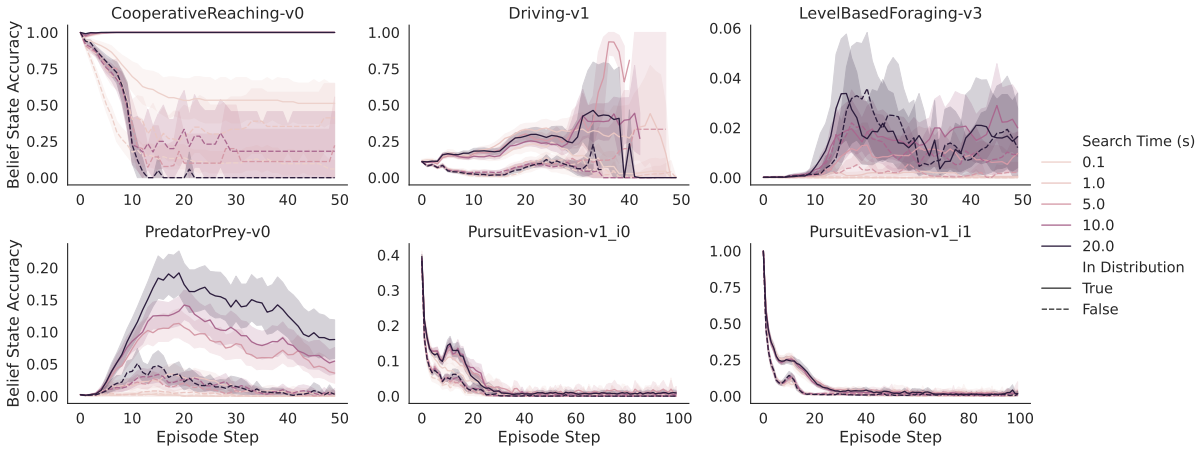


Figure 5.9: Probability assigned by POTMMCP's belief to the true environment state throughout an episode. In general belief accuracy is significantly worse in the out-of-distribution setting, and also in the LevelBasedForaging-v3 environment where initial belief space density is largest (y-axis scales differ between plots).

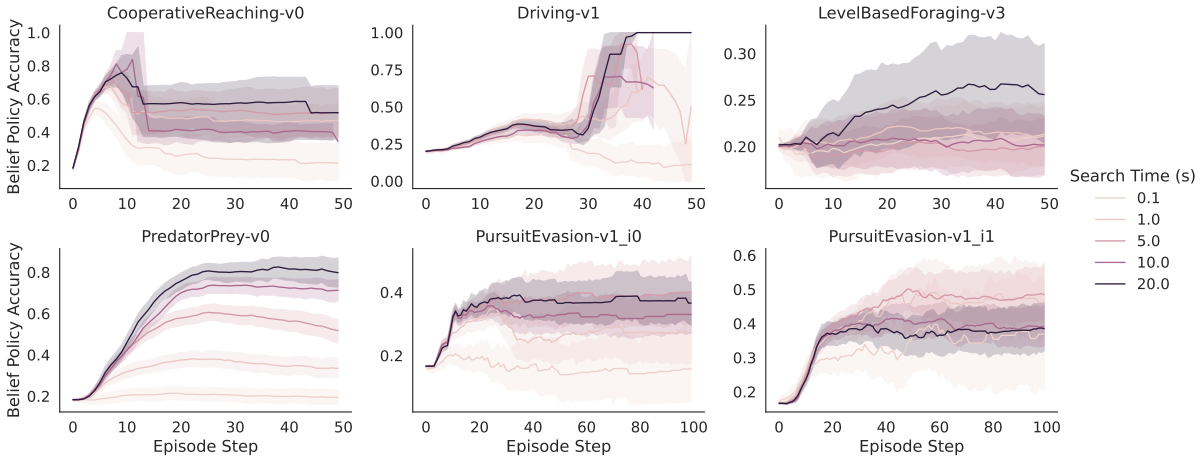


Figure 5.10: Probability assigned by POTMMCP's belief to the true policy of the other agent throughout an episode in the in-distribution setting. We observe that belief accuracy generally increases with search budget, and as the episode progresses.

AT LEAST ONE PLANNING METHOD EQUALS OR EXCEEDS THE LEARNING (RL-BR) METHOD GIVEN SUFFICIENT PLANNING TIME IN FOUR OF SIX ENVIRONMENTS. Specifically, POTMMCP outperforms RL-BR in Driving-v1 and PredatorPrey-v0, and matches the performance in CooperativeReaching-v0 and PursuitEvasion-v1_i0.

One explanation for this is that in these environments, each planning method is able to improve its prediction about the other agent's policy throughout an episode, leading to more accurate beliefs about the environment state and the other agent's future actions, as evidenced in Figures 5.8, 5.9, and 5.10, which show the belief accuracy for the Combined and POTMMCP methods. Improved belief accuracy directly translates to more accurate value estimates for actions and thus a better policy for the planning agent.

Another explanation is the ability of planning to generalize to novel situations. With an accurate belief and model of the environment, planning allows the agent to improve its action value estimates online for any encountered state, regardless of how often the state is encountered. In contrast, RL-BR performs all policy improvement offline, making it brittle when faced with situations rarely encountered during training. Evidence of this can be seen in the success rate of each method in the Driving-v1 environment shown in Table 5.5. Specifically, all methods succeed the vast majority of the time ($> 93\%$), however in the minority of failures RL-BR crashes significantly more often than the planning methods (4.14% vs 0.98%), suggesting that planning methods are able to improve on the robustness of RL-BR in a minority of situations, likely those rare in RL-BR's training distribution. Given the large negative reward for crashing, this small difference in crash probability has a big impact of final expected return. We believe this observation most likely holds beyond the environments used in our experiments and propose further investigation of this as an interesting direction for future research.

Table 5.5: Percentage of in-distribution Driving environment episodes that ended with a crash (*Crashed*), reaching the goal (*Success*), or the step limit being reached (*Timedout*). Results using the maximum search time (20 s) are shown.

Algorithm	Crashed	Success	Timedout
IPOMCP	0.98%	97.46%	1.56%
POTMMCP	0.12%	99.88%	0.00%
RL-BR	4.14%	93.12%	2.74%
COMBINED	0.46%	99.51%	0.03%

In two environments, LevelBasedForaging-v3 and PursuitEvasion-v1_i1, no planning method reached the performance of RL-BR. In LevelBasedForaging-v3, the failure mode of particle-based planning is attributed to poor belief accuracy due to the large initial belief size, as discussed above.

In PursuitEvasion-v1_i1, where the planning agent starts with a perfect belief, the performance gap between planning and learning methods is likely due to the challenge of planning over long horizons. Unlike LevelBasedForaging-v3, poor belief accuracy is not the main cause, as indicated by the increasing performance of the Combined method with search time[5]. Rather, the environment's structure presents a fundamental long horizon planning challenge. Specifically, reaching the goal requires the agent to plan over many time steps while avoiding large negative rewards (in this case, being spotted by the other agent). If the agent is unable to plan deep enough to find

5 In Figure 5.7 we see belief state accuracy decreases over time in the PursuitEvasion-v1_i1 environment, this is expected due to the naturally growing uncertainty over time, this is in contrast the other environments where we expect more certainty over time.

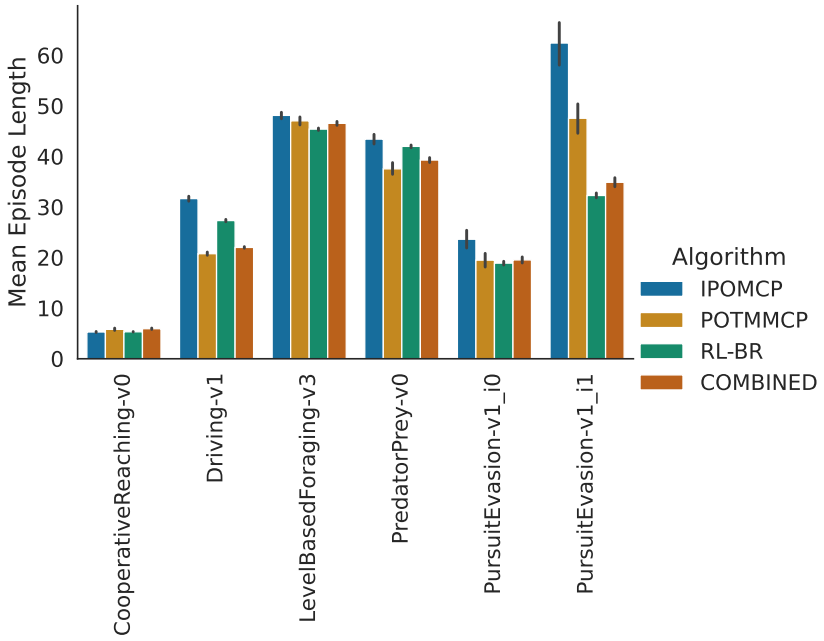


Figure 5.11: Mean in-distribution episode length for each method in each environment. Results for planning (POTMMCP, IPOMCP) and combined methods are using the maximum search time (20 s). Maximum episode length is 50 for all environments except PursuitEvasion-v1, where it is 100. PursuitEvasion-v1_i1 had the longest effective planning horizon. All other environments had shorter effective planning horizons due to the availability of more frequent positive rewards for the agent.

trajectories where it reaches the goal, it instead gets stuck in the sub-optimal strategy of ignoring the goal and instead focusing on just avoiding the other agent. RL-BR benefits from significantly more compute to move beyond the sub-optimal strategy. The pure planning methods on the other hand are less capable of doing this, given their limited search budget. We see evidence of this in the longer average episode times for the planning methods in this environment, with the gap in episode length between RL-BR and the planning methods much greater in PursuitEvasion-v1_i1 than for any other environment (Figure 5.11). Additionally, we found that 10× more training steps were required to train RL-BR till convergence in PursuitEvasion-v1_i1 compared to PursuitEvasion-v1_io (Figure 5.3). The gap in performance between RL-BR and the planning methods highlights the challenge of long-horizons for planning methods. Fortunately, improved search policies, such as using RL-BR in the Combined method, offer a way to overcome the challenge of long-horizons.

5.5.3 *Out-of-distribution performance*

In this section, we explore how well each method generalizes when the planning and test populations differ ($P_{\text{plan}} \neq P_{\text{test}}$) within each environment. Here we include results for INTMCP and POMCP, which do not integrate a planning population into

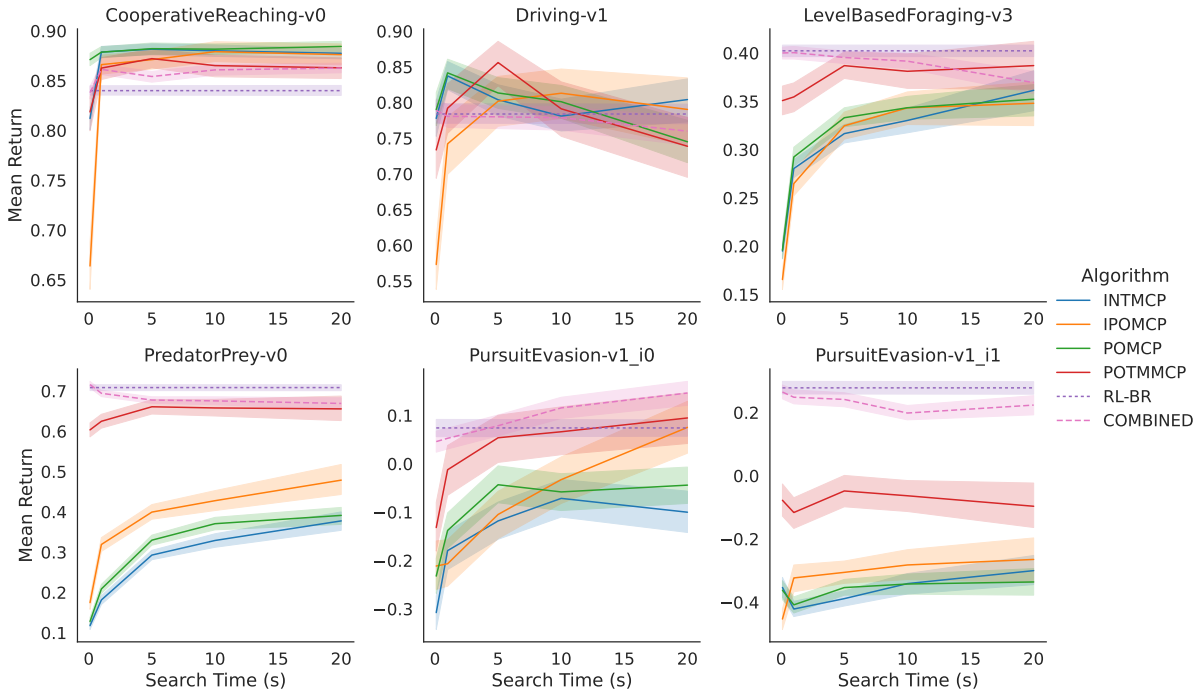


Figure 5.12: Out-of-distribution performance of each method in each environment. The plots show the mean return of each method across planning budgets (x-axis) when the planning and test populations are different, with results averaged over the two experiment populations. Shaded areas show 95% confidence intervals.

the planning process. Instead, INTMCP models other agents using a level two I-POMDP, while POMCP models other agents as uniform random, serving as baselines for common agent modeling frameworks. The main results are shown in Figure 5.12.

COMBINING LEARNING AND PLANNING LEADS TO WORSE PERFORMANCE THAN LEARNING ALONE IN FOUR OF SIX ENVIRONMENTS. This outcome is largely attributed to poor belief accuracy similar to the LevelBasedForaging-v3 environment in the in-distribution setting. As shown in Figure 5.7, incorrect models of the other agent in the out-of-distribution setting results in worse beliefs about the environment state compared to the in-distribution setting, despite having a perfect environment model in both cases. This highlights a critical challenge in integrating planning methodologies in the multi-agent setting: the quality of the model predicting other agents' behaviors profoundly impacts belief accuracy and performance. Finding more robust methods for modeling the other agents, or developing planning techniques better able to handle model inaccuracy in multi-agent settings are both important directions for future work.

PLANNING METHODS THAT MODELED THE OTHER AGENT NAIVELY (POMCP) OR RECURSIVELY (INTMCP) GENERALLY PERFORMED WORSE THAN THE OTHER APPROACHES. While frameworks based on recursive reasoning hold promise for

domains like human-robot interaction (HRI) [WW12], they showed limited utility in our experiments, where the other agent policies did not behave randomly or employ explicit recursive reasoning. Rather our experiments show the benefits that can be gained by explicitly considering the possible types of other agent behavior. Furthermore, a drawback of INTMCP is its reliance on nested MCTS which disperses simulations across multiple decision trees, thus requiring a higher number of simulations for effective decision-making. Improving the accuracy of recursive models remains a practical challenge, suggesting that adopting diverse policies might offer a more universally resilient solution, especially considering the burgeoning field of population-based training [Lan+17; Lup+21; Xin+21; Rah+23].

PLANNING METHODS PERFORMED NO BETTER OR EVEN WORSE THAN PURE LEARNING IN FIVE OUT OF SIX ENVIRONMENTS. Again, this trend is likely due to the formation of inaccurate beliefs about the environment state (as shown in Figure 5.7). This result again highlights the reliance on accurate other agent models for planning in multi-agent settings in current state-of-the-art methods. It also points to a clear need for planning methods better equipped to handle model inaccuracy.

INCREASING SEARCH BUDGET DID NOT CONSISTENTLY LEAD TO MONOTONICALLY IMPROVING PERFORMANCE FOR PLANNING METHODS. Unlike the in-distribution setting, higher planning time did not always result in improved performance in the out-of-distribution setting. Performance varied significantly across environments, attributed to the formation of inaccurate beliefs and challenges in predicting other agent's actions (Figure 5.13). These findings underscore the complexities and limitations of planning methods in environments where accurate modeling of agents and their interactions is challenging.

5.6 CONCLUSION AND FUTURE DIRECTIONS

In this chapter we conducted an extensive empirical evaluation of existing state-of-the-art planning under uncertainty methods and compared them with a method combining planning and RL. Our investigation was the first comprehensive analysis comparing state-of-the-art decision-theoretic online planners for large, partially observable multi-agent environments in the type-based reasoning setting. Furthermore, it is the first exploring combining learning with planners based on particle beliefs. Our results highlight that combining RL with particle based planning can be an effective method for improving the robustness of agents in the multi-agent setting. Importantly, we observed that given an accurate model of the environment and other agents, planning

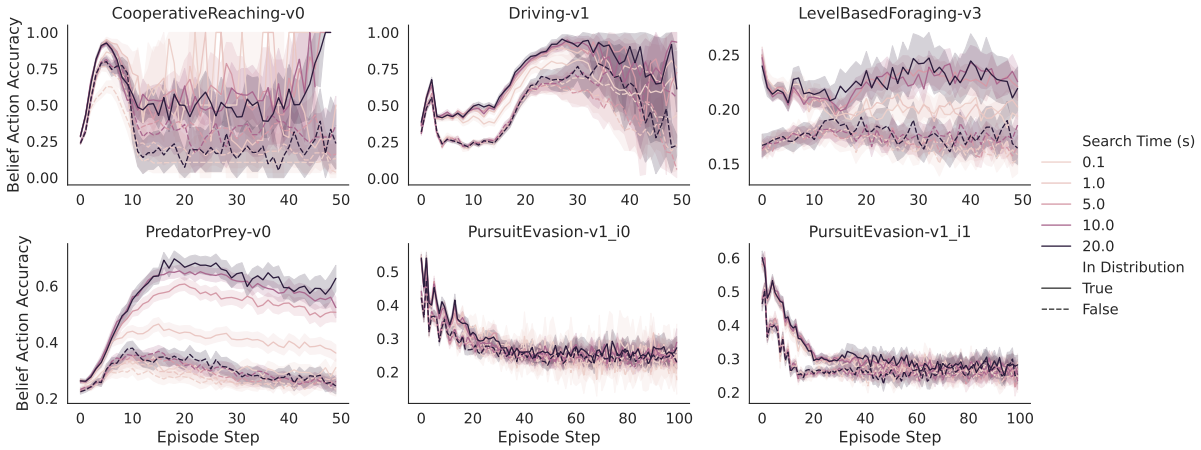


Figure 5.13: Probability assigned by the Combined method's belief to the true action of the other agent during an episode. In general we find beliefs about the other agent's next action are significantly worse in the out-of-distribution setting.

was able to improve on planning or RL alone, and indicated much more reliable performance in some settings. We also discover some key limitations of combining existing methods of particle based planning with learning, namely performance can suffer if there is inaccuracy in the other agent model, or due to poor belief approximation in environments with dense belief spaces.

We hope that these empirical results will spur further research looking at the integration of model-driven and data-driven approaches for robust decision-making in partially observable, multi-agent domains. We suggest that future research should focus on improving robustness to novel partners, and exploring scalable representations of complex beliefs.

5.A EXPERIMENT POPULATIONS

For our experiments we used a set P of 10 to 12 policies for each environment.

COOPERATIVEREACHING-V1 P consisted of 11 heuristic policies `H[1-11]`. With $P_0 = \{$`H1, H2, H3, H4, H5`$\}$ and $P_1 = \{$`H6, H7, H8, H9, H10, H11`$\}$. These policies were based on prior work [Rah+23] with some adjustments made to ensure the population was diverse according to their pairwise returns (Figure 5.14a).

DRIVING-V1 P consisted of 10 policies: five heuristic and five RL trained policies. $P_0 = \{$`A0, A40, A60, A80, A100`$\}$ was made up of the heuristic policies, while $P_1 = \{$`RL1, RL2, RL3, RL4, RL5`$\}$ contained all the RL policies. Each heuristic policy followed the shortest path from the agent's start position to the goal but differed on how aggressive they were, from least aggressive `A0` to most aggressive `A100`. The aggressiveness of a policy controlled how far away another agent had to be within the agent's field of vision before the policy would stop the agent's vehicle from moving. `A0` would stop if another agent was observed anywhere and would only continue once that agent was out of view. Conversely, `A100` would continue along the shortest path irrespective of how close another observed agent was. `A[40-80]` followed policies between the two extremes. The RL policies `RL[1-5]` were produced by first training six policies using Self-Play (SP) then pruning away any similar policies based on pairwise returns to give the final set of five policies. The pairwise returns for each policy in the population P for this environment are shown in Figure 5.14b.

LEVELBASEDFORAGING-V3 P consisted of 10 policies: five heuristic and five RL trained policies. $P_0 = \{$`H1, H2, H3, H4, H5`$\}$ contained the heuristic policies, while $P_1 = \{$`RL1, RL2, RL3, RL4, RL5`$\}$ contained all the RL policies. The heuristic policies were based on prior work [Rah+23], adapted to deal with partial observability. We pruned many of the heuristic policies used in the prior work as we found that they resulted in similar behaviors based on their returns. The five heuristic policies used were:

- `H1` always goes to the closest observed food, irrespective of the foods level.
- `H2` goes towards the visible food closest to the center of visible players, irrespective of food level.
- `H3` goes towards the closest visible food with a compatible level.
- `H4` selects and goes towards the visible food that is furthest from the center of visible players and that is compatible with the agents level.

- H5 targets a random visible food whose level is compatible with all visible agents.

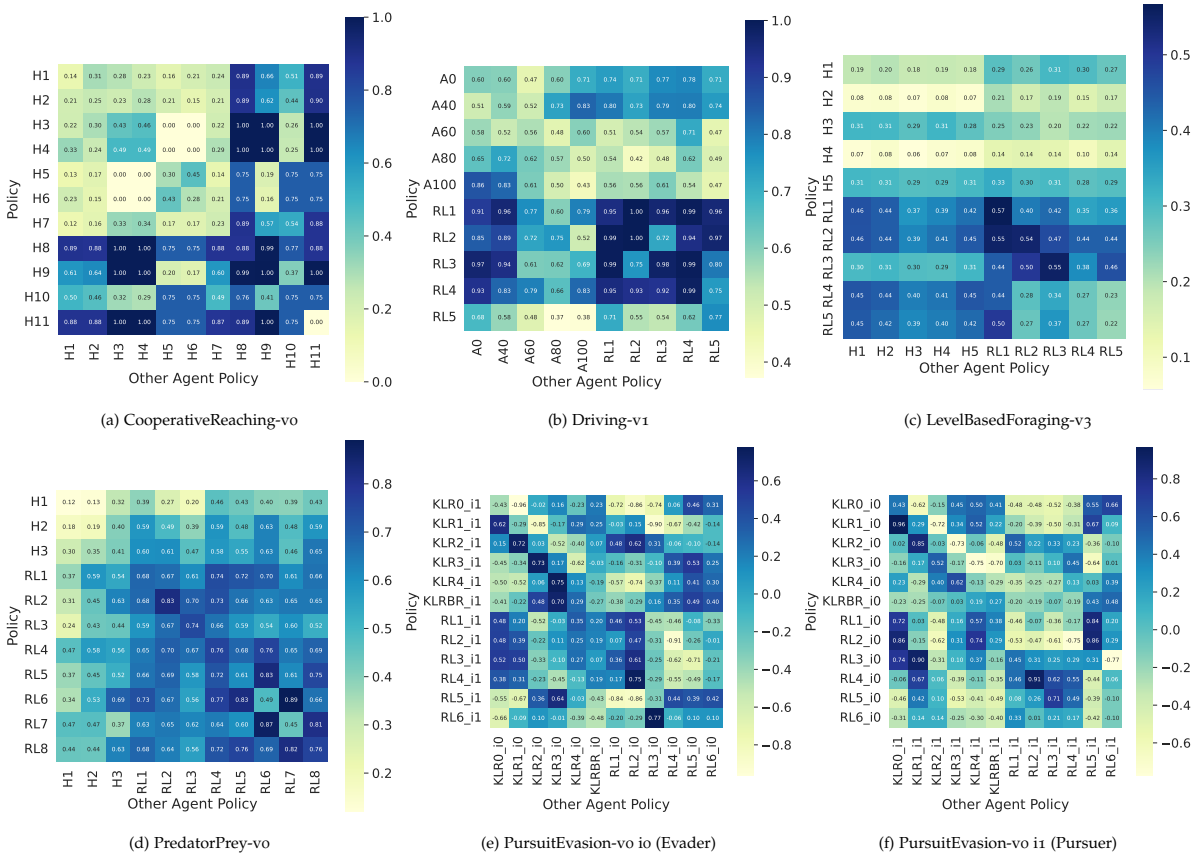
For the RL policies we trained a population of 13 RL policies including six using SP and seven using the Synchronous K-Level Reasoning with Best-Response (SyKLRBR) algorithm [Cui+21] (up to $K = 5$). The resulting five RL policies RL[1-5] were found by pruning away similar policies from the full set of 13 policies. To do this the policies were clustered based on their pairwise returns then a single policy from each cluster was chosen. The pairwise returns for each policy in the population P for this environment are shown in Figure 5.14c.

PREDATORPREY-V0 P consisted of 11 policies: three heuristic and eight RL trained policies. $P_0 = \{$H1, H2, H3, RL1, RL2$\}$ was made up of a mix of heuristic and RL policies, while $P_1 = \{$RL3, RL4, RL5, RL6, RL7, RL8$\}$ contained the remaining RL policies. The heuristic policies were chosen based on trying various heuristics and selecting those that had diverse pairwise returns. The three heuristic policies used were:

- H1 moves towards closest observed prey, closest observed predator, or explores randomly, in that order.
- H2 moves towards closest observed prey, closest observed predator, or explores in a clockwise spiral around arena, in that order.
- H3 moves towards closest observed prey to the closest observed predator or explores in a clockwise spiral around arena, in that order.

For the RL policies we followed an identical protocol to the Level-Based Foraging environment; first training 13 policies using SP and SyKLRBR and then pruning similar policies to produce the final population of RL policies. The pairwise returns for each policy in the population P for this environment are shown in Figure 5.14d.

PURSUITEVASION-V0 (EVADER "0" AND PURSUER "1") For both agents $X \in \{0, 1\}$, P consisted of 12 RL policies. $P_0 = \{$KLR0_iX, KLR1_iX, KLR2_iX, KLR3_iX, KLR4_iX, KLRBR_i$X$$\}$ was a population of KLR policies trained using SyKLRBR, while $P_1 = \{$RL1_iX, RL2_iX, RL3_iX, RL4_iX, RL5_iX, RL6_i$X$$\}$ contained RL policies trained using a mix of SP and SyKLRBR. For the RL policies a population of 30 SyKLRBR policies (five separate populations of six policies with up to $K = 4$) and five self-play (no best-response) policies were trained. The most diverse (based on pairwise returns) SyKLRBR population of six policies was then selected for P_0. P_1 was then chosen by pruning away similar policies from the remaining 29 SyKLRBR and self-play policies, based on pairwise returns. The pairwise returns for each policy in the population P for this environment are shown in Figure 5.14e and Figure 5.14f.



(a) CooperativeReaching-v0 (b) Driving-v1 (c) LevelBasedForaging-v3

(d) PredatorPrey-v0 (e) PursuitEvasion-v0 i0 (Evader) (f) PursuitEvasion-v0 i1 (Pursuer)

Figure 5.14: Payoff tables for policies used in the populations P_0 and P_1 used in the experiments. Each table shows the mean returns for the row policy when paired with the column policy after 1000 episodes.

6

POSGGYM

This chapter has been published as part of a journal article:

Jonathon Schwartz et al. "POSGGym: A Library for Decision-Theoretic Planning and Learning in Partially Observable, Multi-Agent Environments". In: *Autonomous Agents and Multi-Agent Systems* 39.2 (2025), p. 35. ISSN: 1573-7454. DOI: 10.1007/s10458-025-09716-6

It is also part of a workshop paper:

Jonathon Schwartz et al. "POSGGym: A Library for Decision-Theoretic Planning and Learning in Partially Observable, Multi-Agent Environments". In: *International Conference on Automated Planning and Scheduling PRL Workshop*. 2024

A key driver of progress in artificial intelligence research is the availability of high quality, open-source benchmark domains. Benchmarks allow researchers to make meaningful comparisons, as well as help guide research direction. In recent years, an ecosystem of open-source libraries has emerged supporting multi-agent research in reinforcement learning (RL) and game-theory, greatly aiding advancements in these areas. Comparatively, progress in decision-theoretic planning has been hindered by the lack of adequate evaluation and simulation platforms. Instead, researchers have had to rely on creating custom environments, which reduces efficiency, and makes comparison of new methods difficult. In this chapter, we introduce POSGGym: a library for facilitating planning and RL research in partially observable, multi-agent domains. It provides a diverse collection of discrete and continuous environments, complete with their dynamics models and a reference set of policies that can be used for evaluation, for example to test generalization to novel other agent behaviors. POSGGym assimilates the various environments used throughout this thesis, and more, into a single convenient package in order to aid future research in planning and RL. Library code is available at: https://github.com/RDLLab/posggym.

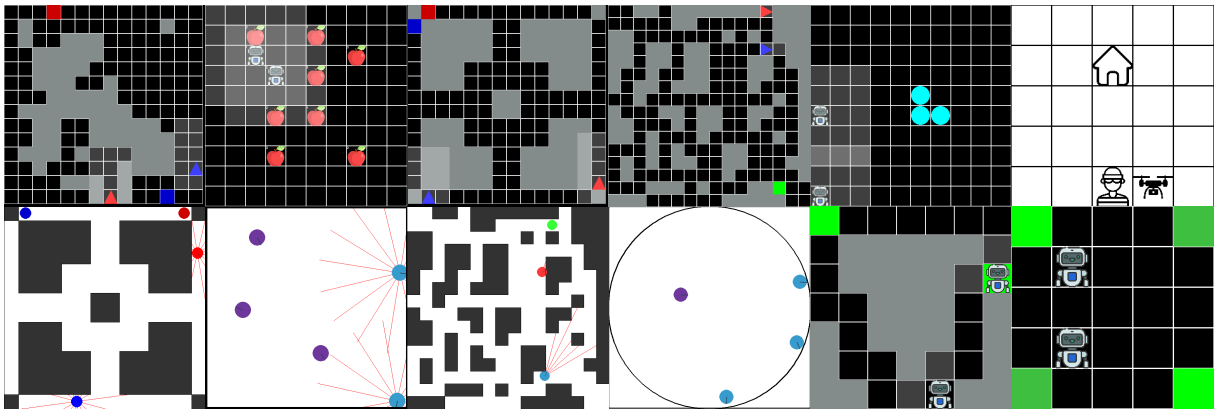


Figure 6.1: POSGGym is a library for planning and RL research in partially observable, multi-agent domains. It includes a diverse set of discrete and continuous environments along with a collection of reference policies for reproducible evaluation.

6.1 INTRODUCTION

Recent years have seen substantial progress in multi-agent decision making, leading to significant achievements in complex domains such as Go [Sil+16], Dota [Ber+19], Starcraft [Vin+19], Poker [BS19], Stratego [Per+22], and Diplomacy [Bak+22]. These accomplishments were driven both by progress in deep learning [LBH15], and the availability of high-quality benchmark domains. Access to such benchmarks is crucial for facilitating research, as it simplifies comparison, reduces experiment times, and helps guide research direction.

In response to the growing demand for high-quality environments, several multi-agent research libraries have emerged. This includes large environment collections with standardized APIs such as PettingZoo [Ter+21], MeltingPot [Lei+21], and OpenSpiel [Lan+19]. Additionally, there are libraries focusing on specific problem domains such as the Starcraft multi-agent challenge (SMAC) [Sam+19; Ell+24], and Google football [Kur+20]. Apart from providing researches with a diverse set of problems, these libraries have also driven the adoption of standardized environment interfaces. This, in turn, has allowed for greater reuse of existing algorithm code [Lia+18; Hua+22; Raf+21; Pet+20], leading to further improvements in research efficiency.

Unfortunately, no existing library offers a diverse set of benchmarks for decision-theoretic planning. Most popular libraries are primarily designed for research in multi-agent reinforcement learning (MARL) [BBD08; SPG07; TW12] or game-theory [OR94] research. Libraries designed for MARL [Ter+21; Lei+21; Sam+19; Ell+24; Jul+18; Tun+20; Bam21] assume that agents do not have access to the dynamics model of the environment. While this assumption aligns with the goals of these libraries, it renders them unsuitable or requiring of significant extra effort to be used for planning. Conversely, the most established library that does support planning, namely OpenSpiel

[Lan+19], is built around facilitating game-theoretic research under the extensive form games (EFG) [OR94] paradigm and focuses on discrete domains, primarily classic turn-based board and card games.

As a result of the current library ecosystem, researchers interested in decision-theoretic planning have had to rely on implementing their own environments [PG17; Eck+20; Kak+22; SZK22; SKH23]. This lack of standardized environments makes it difficult to perform comparisons and reduces research efficiency.

In this chapter, we introduce *POSGGym*, a research library for planning and RL in partially observable, multi-agent environments. POSGGym models environments using the Partially Observable Stochastic Games (POSG) framework [HBZ04] and supports both discrete and continuous domains. It includes implementations of established and newer planning benchmarks, all under a unified API that is compatible with the existing ecosystem of MARL libraries [Ter+21; Lia+18; Hua+22; Raf+21; Pet+20]. Additionally, POSGGym provides reference policies for many of the environments, a crucial ingredient for multi-agent research. These help save researcher time and enable reproducible evaluations.

6.2 BACKGROUND

POSGGym models environments as Partially Observable Stochastic Games (POSGs) [HBZ04], which generalize many other formal-decision-making models[1]. The generality of POSGs has lead to their wide adoption in decision-theoretic planning and MARL research, making it an ideal framework to design POSGGym around.

Formally, a POSG is a tuple $\mathcal{M} = \langle \mathcal{I}, \mathcal{S}, S_0, \vec{\mathcal{A}}, \vec{\mathcal{O}}, T, Z, R \rangle$ consisting of N agents indexed $\mathcal{I} = \{1, \dots, N\}$, a set of Markov states of the environment $\mathcal{S}$, an initial state distribution $S_0 \in \Delta(\mathcal{S})$[2], the joint action space $\vec{\mathcal{A}} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_N$, the joint observation space $\vec{\mathcal{O}} = \mathcal{O}_1 \times \cdots \times \mathcal{O}_N$, a state transition function $T : \mathcal{S} \times \vec{\mathcal{A}} \times \mathcal{S} \to [0, 1]$ specifying the probability of transitioning to state s' given joint action $\vec{a}$ was performed in state s, the joint observation function $Z : \mathcal{S} \times \vec{\mathcal{A}} \times \vec{\mathcal{O}} \to [0, 1]$ specifying the probability of joint observation $\vec{o}$ after performing joint action $\vec{a}$ and ending up in state s', and a reward function $R : \mathcal{S} \times \vec{\mathcal{A}} \to \mathbb{R}^N$ defining the reward each agent receives when joint action $\vec{a}$ is performed in state s. Combining T, Z, R into a single function produces a generative model $\mathcal{G}$ which returns the next state, joint observation, and joint reward, given the current state and joint action $\langle s', \vec{o}, \vec{r} \rangle \sim \mathcal{G}(s, \vec{a})$.

At each step, each agent $i \in \mathcal{I}$ simultaneously performs an action $a_i \in \mathcal{A}_i$ and receives an observation $o_i \in \mathcal{O}_i$ and reward $r_i \in \mathbb{R}$. Each agent has no direct access

1 See Chapter 2 for more in-depth coverage of POSGs and other formal models.

2 We use S_0 to denote the initial state distribution to distinguish it from the initial belief of an agent b_0 which in the multi-agent setting can be a distribution over more than just environment states.

Table 6.1: Existing multi-agent environment libraries and their properties. $\mathcal{O}$: supported observation space types, $\mathcal{A}$: supported action space types, D: Discrete, C: Continuous, P: RGB Pixels. Noting that not all observation and action types may be supported in all environments.

	Name	Formalism	Model Support	$\mathcal{O}$	$\mathcal{A}$	Reference Policies
Suites	**POSGGym**	**POSG**	**Yes**	**D/C/P**	**D/C**	**Yes**
	OpenSpiel [Lan+19]	EFG	Yes	D	D	Yes[1]
	PettingZoo [Ter+21]	POSG	No	D/C/P	D/C	No
	Melting Pot [Lei+21]	POSG	No	P	D	Yes
	MADPToolbox [SO08]	POSG	Yes	D	D	No
	JaxMARL [do +22]	POSG	Yes	D/C/P	D/C	No
Creation platforms	UnityML-Agents [Jul+18]	POSG	No	D/C/P	D/C	No
	Griddly [Bam21]	POSG	Yes	D/P	D	No
	DMLab2d [Bea+20]	POSG	No	D/P	D/C	No
	Gigastep [Lec+23]	POSG	Yes	C/P	D/C	No
Specific domains	SMAC [Sam+19; Ell+24]	Dec-POMDP	No	C	D	Yes
	Neural-MMO [Sua+19]	POSG	No	D/C	D	No
	MAgent [Zhe+18]	POSG	No	D	D	No
	Pommerman [Res+18]	POSG	Yes	D	D	No
	DMControl [Tun+20]	POSG	No	C	C	No
	MaMuJoCo [Pen+21]	Dec-POMDP	No	C	C	No
	Google-Football [Kur+20]	MG	Yes	D/C/P	D	Yes
	CityFlow [Zha+19b]	MMDP	No	C	D	No
	SUMO [Lop+18]	MMDP	No	C	D	No
	POGEMA [Skr+22]	POSG	Yes	D	D	No
	rlcard [Zha+19a]	EFG	Yes	D	D	Yes
	Pgx [Koy+24]	EFG	Yes	D	D	Yes

to the environment state or knowledge of the other agent's actions and observations. Instead they must rely only on information in their *action-observation history* up to the current time step t: $h_{i,t} = \langle o_{i,0} a_{i,0} o_{i,1} a_{i,1} \dots o_{i,t-1} a_{i,t-1} o_{i,t} \rangle$. The set of all time t histories for agent i is denoted $\mathcal{H}_{i,t}$. Agents select their next action using their *policy* π_i which is a mapping from their history h_i (or belief) to a probability distribution over their actions, where $\pi_i(a_i|h_i)$ denotes the probability of agent i performing action a_i given history h_i. The goal of each agent i is to maximize its total expected *return* given by $G_i = \mathbb{E}\left[\sum_{t'=t}^{\infty} \gamma^{t'-t} r_{i,t'}\right]$, where $\gamma \in [0, 1)$ is a discount factor. Importantly, each agent's reward at each step depends on the actions taken by the other agents present in the environment.

6.3 SURVEY OF MULTI-AGENT LIBRARIES

The recent success of multi-agent methods has motivated the creation of numerous research libraries. These can roughly be categorized into *environment suites*, *creation*

platforms, and *specific domain* libraries. The full list of related libraries and their properties is shown in Table 6.1.

6.3.1 *Environment Suites*

Environment suites offer a diverse collection of environments, often accompanied by a general API that can be readily adopted by third-party libraries.

PettingZoo [Ter+21] is one of the most popular libraries for MARL research, providing a standard API and an extensive range of environments. It includes continuous and discrete domains, catering to fully and partially observable problems alike. Its API has been adopted by a number of other popular libraries, including MAgent [Zhe+18] and Neural-MMO [Sua+19]. Additionally, PettingZoo is supported by most popular RL algorithm libraries [Lia+18; Hua+22; Raf+21; Pet+20]. However, PettingZoo is designed specifically for MARL research and does not provide environment models for planning purposes. Furthermore, while it is compatible with many existing algorithm libraries, it does not supply pre-trained policies for any of its environments.

OpenSpiel [Lan+19] is an environment suite primarily focused on research into games. It models environments as EFGs and provides a large collection of classical board and card games, alongside some grid-world and social dilemma problems. Notably, it includes popular benchmarks like Go and Hanabi [Bar+20]. A key feature of OpenSpiel is its support for planning by exposing the environment model in its API. While OpenSpiel's API can be transformed into a POSG, it has a number of limitations for decision-theoretic planning research. Firstly, it lacks support for continuous actions, a common and important case for planning in areas such as robotics. Secondly, while OpenSpiel provides a few grid-world environments its primary focus remains on classic card and board games, which does not cover all the domains of interest for decision-theoretic planning. Along with environments OpenSpiel incorporates implementations of many key multi-agent algorithms but provides reference policies for only a few classic games.

Melting Pot [Lei+21] is a suite offering a diverse collection of grid-world environments. Its significant contribution lies in providing pre-trained agents for all the environments, thereby, offering a wide range of *scenarios* (environment + agents) for evaluation purposes. This feature makes MeltingPot particularly suitable for investigating the generalization of agents when interacting with novel partners. However, MeltingPot is designed for MARL research; providing environments limited to RGB pixel observations and discrete actions, and no access to the environment model.

The Multi-Agent Decision Process Toolbox (MADPToolbox) [SO08] is a collection of algorithms and problems created to facilitate research in decision-theoretic planning

and learning within multi-agent systems. Prior to POSGGym, MADPToolbox was arguably the only library for decision-theoretic planning containing a diverse collection of problems and algorithms. While designed to be rather general, the primary focus of MADPToolbox has been on planning algorithms for fully cooperative settings modeled as discrete Decentralized-POMDPs (Dec-POMDPs) [Ber+02; OA16]. It contains many classic planning benchmarks and algorithms. However, unfortunately is no longer actively maintained, which has resulted in some of its problems becoming less relevant given recent advances in planning. Additionally, the design of the library makes it difficult to integrate with the modern ecosystem of deep learning and RL libraries.

JaxMARL [Rut+23] is another suite which focuses on speeding up deep MARL research by integrating hardware accelerators into its design. It provides implementations of a number of common MARL benchmarks such as SMAC, and Hanabi, with support for vectorization. As it stands it does not provide reference policies for evaluation. Additionally, the large speed-ups from the hardware accelerators only come when you are able to heavily parallelize your policy and environment. This is the case for deep learning based MARL policies which can easily take advantage of batched computing, however this does not transfer easily to most existing planning methods based on Monte-Carlo Tree Search (MCTS).

6.3.2 *Creation Platforms*

Environment creation platforms provide a framework for creating new environments. While they typically come with a collection of reference environments, their main design goal is to enable researchers to produce their own problem scenarios. The most popular platforms include UnityML-Agents [Jul+18], Griddly [Bam21], and DMLab2D [Bea+20]. These platforms are highly optimized and robust. However, most are designed specifically for single-agent RL research. As a result, they either do not provide access to environment models, offer limited multi-agent environments, lack reference policies, or have restrictions to purely discrete settings. Gigastep [Lec+23] is a more recent MARL focused library that provides a platform with a collection of 2D and 3D cooperative and adversarial environments. The main advantage of Gigastep is its support for hardware accelerators for faster simulations when performing batched training common with modern MARL methods.

6.3.3 *Specific Domains*

Specific domain libraries concentrate on particular problem domains rather than of providing a diverse collection of environments. These libraries typically revolve around

specific benchmarks proposed by the community, and have played a crucial role in focusing research efforts on specific challenges.

Several specific domain libraries have emerged to facilitate research in MARL. One of the most popular is the StarCraft Multi-Agent Challenge (SMAC) [Sam+19; Ell+24], which is based on the StarCraft 2 game (specifically the StarCraft 2 Learning Environment (SC2LE) [Vin+17]). It concentrates on complex cooperative, decentralized control scenarios. Other MARL libraries include Neural MMO [Sua+19] for open-ended massively multi-agent game environment, MAgent [Zhe+18] where the number of agents ranges from the hundreds to the millions, libraries for autonomous driving [Lop+18; Zha+19b], and robotics [Tun+20; Pen+21]. While these libraries offer unique challenges, none of them currently support planning models or provide reference policies for evaluation.

There are a few specific domain libraries that incorporate dynamics models. Examples includes Pommerman [Res+18], Google Football [Kur+20], POGEMA for multi-agent path-finding [Skr+22], rlcard for card-games [Zha+19a], and Pgx for accelerator supported board games [Koy+24]. These libraries offer valuable resources for planning and RL research in their respective domains. However, all are restricted to discrete domains, and lack a unified model API, which can make it challenging to reuse algorithm code across the different libraries.

6.4 POSGGYM

The aim of POSGGym is to streamline planning and learning research in partially observable, multi-agent environments, with a particular emphasis on planning since this is currently lacking in existing research libraries. To accomplish this, POSGGym uses a general yet user-friendly API, and includes a diverse collection of environments and reference agents. POSGGym's API is based on the Gymnasium [Bro+16; Fou22] and PettingZoo [Ter+21] libraries, as these are widely used by the RL community. This design choice has the additional benefit of making it simple to integrate POSGGym with the many MARL algorithm libraries that are currently compatible with PettingZoo, e.g. [Lia+18; Hua+22; Raf+21].

To summarize, the key design goals for POSGGym are:

- Provide a general API for environments, models, and reference agents that supports both planning and RL
- Provide implementations of a range of established and newer planning benchmark environments

Figure 6.2: POSGGym's high-level architecture. Agents interact with the environment by selecting actions according to their policy which has access to a model of the environment for planning.

- Provide a diverse set of co-player policies (reference agents) for implemented environments
- Be similar to the Gymnasium and PettingZoo APIs and compatible with the main RL algorithm libraries

6.4.1 *API Design*

The POSGGym API has three main components: *environment*, *model*, and *agents* (Figure 6.2).

6.4.1.1 *Environment API*

The environment API, depicted in Figure 6.3a, closely follows the structure of PettingZoo's parallel environment API (Figure 6.3b) with some aspects aligning more closely with the Gymnasium API (Figure 6.3c).

At each timestep, each agent provides an `action`, which collectively form a joint action that is passed to the `step` function. This function updates the state of the environment and returns `observations`, `rewards`, `terminations`, `truncations`, `all_done`, `infos`. Each of these return values (except `all_done`) is a mapping from the ID of the agent to their respective return value. The `step` function mirrors PettingZoo's step function, with the addition of `all_done`, which indicates when all agents in the environment have reached a terminal state. Similarly, the `reset` method mirrors PettingZoo's and resets the environment to a starting state and returns an `observation` and `info` for each active agent, where `info` may contain additional environment specific information. The state, observation, and action spaces utilize the same underlying classes as Gymnasium and PettingZoo, greatly simplifying compatibility across libraries. The `render` and `close` functions operate identically to Gymnasium: `render` provide visual representation of the current state of the environment, while `close` shuts-down the environment and performs any necessary clean-up.

POSGGym's environment API differs from the PettingZoo parallel environment API in two key aspects: the `make` function for environment initialization, and the inclusion of `all_done` in the `step` method's return values. The use of the `make` function aligns

```python
import posggym
env = posggym.make("PursuitEvasion-v1", render_mode="human")
observations, infos = env.reset(seed=42)

for _ in range(1000):
    actions = {i: policies[i](observations[i]) for i in env.agents}
    observations, rewards, terminations, truncations, all_done, infos = env.step(actions)

    if all_done:
        observations, infos = env.reset()

env.close()
```

(a) POSGGym

```python
from pettingzoo.butterfly import pistonball_v6
env = pistonball_v6.parallel_env(render_mode="human")
observations = env.reset(seed=42)

for _ in range(1000):
    actions = {i: policies[i](observations[i]) for i in env.agents}
    observations, rewards, terminations, truncations, infos = env.step(actions)

    if not env.agents:
        observations = env.reset()

env.close()
```

(b) PettingZoo

```python
import gymnasium as gym
env = gym.make("LunarLander-v2", render_mode="human")
observation, info = env.reset(seed=42)

for _ in range(1000):
   action = policy(observation)
   observation, reward, terminated, truncated, info = env.step(action)

   if terminated or truncated:
      observation, info = env.reset()

env.close()
```

(c) Gymnasium

Figure 6.3: Environment APIs.

more closely with the design of Gymnasium, offering users more convenience and control. While the inclusion of `all_done` differs from both PettingZoo and Gymnasium. This addition aims to simplify the tracking of agent termination during an episode, which can be non-trivial in open environments where the agents that are active may change over time. It also accounts for scenarios where agents can leave or join the environment within a single episode.

To make integration with existing MARL libraries easier, POSGGym provides a `PettingZoo` wrapper class that enables the conversion of any POSGGym environment into an equivalent PettingZoo parallel API environment. By using this wrapper, any POSGGym environment can be used seamlessly with any library that supports the PettingZoo API.

6.4.1.2 *Model API*

POSGGym's model API serves as the distinguishing feature between POSGGym and existing environment libraries. The model API, shown in Figure 6.4, provides access to a generative model of the environment, which can be utilized planning. The design of the model API closely resembles the environment API, with the exception that the majority of methods take an environment `state` as an additional input. The main methods are as follows:

- `sample_initial_state` – samples an initial environment state.
- `sample_initial_obs` – samples initial observations for each agent given a state.
- `get_agents` – returns the IDs of agents that are active in a given state.
- `step` – similar to the environment `step` function, but also returns the next state. It takes both a state and joint actions as inputs.
- `seed` – sets the random seed for the model.

Models in POSGGym are stateless, except for their random seed. This is the key difference when comparing them to the environment API, which maintains an internal state. In fact, the default implementation of the Environment API in POSGGym is merely a wrapper that manages the state of an underlying model class.

By separating out the underlying model from the environment, agents have access to the model without direct access to the environment and its internal state (except through their actions of course). This differs from integrating the model functionality into the environment API, within the `POSGGym.Env` class, which would require users to be careful when selecting which methods they use in their planners so as to avoid accessing information that the agent shouldn't have. Having separate classes for

```python
import posggym
env = posggym.make("PredatorPrey-v0")
model = env.model
model.seed(seed=42)

state = model.sample_initial_state()
observations = model.sample_initial_obs(state)

for t in range(50):
    actions = {i: policies[i](observations[i]) for i in model.get_agents(state)}
    timestep = model.step(state, actions)

    # timestep attribute can be accessed individually:
    state = timestep.state
    observations = timestep.observations

    # Or unpacked fully
    # state, observations, rewards, terminations, truncations, all_done, infos = timestep

    if timestep.all_done:
        state = model.sample_initial_state()
        observations = model.sample_initial_obs(state)
```

Figure 6.4: POSGGym Model API.

the model and environment makes it explicit which model functions can be used for planning (the model API), and what functions control the true environment (the environment API). It also allows for greater flexibility. For example, to study planning with inaccurate models one can easily provide agents with models that are different from the environment, say with varying initial parameters or simplified dynamics.

In addition to the generative model functionality described above, POSGGym defines the `POSGGym.POSGFullModel` API, which extends the `POSGGym.POSGModel` class to include all components of the formal POSG definition. This extension allows POSGGym to be used for defining models for algorithms that require the full model, rather than just a generative model. Specifically, the `POSGGym.POSGFullModel` includes the following additional methods:

- `get_initial_belief` – returns the initial state distribution S_0
- `transition_fn` – defines the state transition function $T : \mathcal{S} \times \vec{\mathcal{A}} \times \mathcal{S} \to [0, 1]$
- `observation_fn` – defines the joint observation function $Z : \mathcal{S} \times \vec{\mathcal{A}} \times \vec{\mathcal{O}} \to [0, 1]$
- `reward_fn` – defines the joint reward function $R : \mathcal{S} \times \vec{\mathcal{A}} \to \mathbb{R}^N$

The full model definition is not included in the main `POSGGym.POSGModel` model class due to the difficultly of implementing full models in environments with very large state, action, or observation spaces and complex dynamics. In such cases, it is typically more practical and common to define a generative model that can be used for sample-based planning approaches like MCTS [Ros+08; SV10].

6.4.1.3 *Agent API*

POSGGym includes a collection of *reference policies* for many of its environments. Access to these policies is done through the Agent API, shown in Figure 6.5.

In this context it is important to distinguish between an agent and a policy. At a high-level, a reference policy π_i represents a fixed policy that maps from an agent's action-observation history to its actions. Conversely, within each environment, there exists a predefined number of agents N who interact within the environment. For each episode, each agent $i \in 1, \ldots, N$ has an associated policy π_i. In certain environments, such as symmetric environments where all agents are equivalent, it is possible for multiple agents to use the same policy. For example, in the game *Rock, Scissors, Paper* both agents could use the same policy of always playing "rock". In this way, POSGGym provides reference policies that can used for one or more agents within a given environment.

The goal of the Agent API is to offer a diverse collection of high-quality reference policies that can be leveraged for evaluation. As we demonstrate in Chapter 5, the policies can be used to measure generalization by holding-out the policies for evaluation only, with no use of the policies during RL training or within the planning algorithm (for other examples see [Lei+21; Jad+19; Ber+19; Vin+19]). Alternatively, the policies can be used during planning, for example by using the policies to guide search [SKH23]. Providing a set of high-quality reference policies enables many possibilities for research, helps boost research efficiency, and makes reproducible comparisons easier. This is particularly valuable in complex, partially-observable environments where generating policies can be especially challenging, requiring time, knowledge, and often access to substantial computing resources.

The Agent API, depicted inFigure 6.5, follows a similar design to the Environment API. Its key methods include `make`, `step`, and `reset`. The `make` function initializes a new instance of a policy, from the policy's unique ID, the environment model, and the ID of the agent the policy will be used for. It returns an `POSGGym.agents.Policy` class instance, the main Agent API class, which has two main methods: `reset` and `step`. `reset` sets the policy to its initial state, while `step` updates the policy with the latest observations for the policy's agent and returns the agent's next action.

When tackling POSGs, a critical consideration is the requirement for policies to maintain an internal state in order to handle partial observability. As discussed in Chapter 2 of this thesis, approaches vary: some utilize explicit beliefs where agents maintain and update a probabilistic representation of the unobservable features of the environment, while others rely on implicit beliefs that leverage learned representations or neural network architectures to capture relevant information from observations. Incorporating these crucial elements into decision-making allows policies to exhibit

```
import posggym
import posggym.agents as pga
env = posggym.make("PursuitEvasion-v1", grid="16x16")

policies = {
    "0": pga.make(
        "PursuitEvasion-v1/grid=16x16/RL1_i0-v0", env.model, "0"
    ),
    "1": pga.make(
        "PursuitEvasion-v1/ShortestPath-v0", env.model, "1"
    ),
}

seed = 42
observations, infos = env.reset(seed=seed)
for policy in policies.values():
    seed += 1
    policy.reset(seed=seed)

for _ in range(1000):
    actions = {
        i: policies[i].step(observations[i]) for i in env.agents
    }
    observations, rewards, terminations, truncations, all_done, infos = env.step(actions)

    if all_done:
        observations, infos = env.reset()
        for policy in policies.values():
            policy.reset()

env.close()
for policy in policies.values():
    policy.close()
```

Figure 6.5: POSGGym Agent API.

more sophisticated and adaptive behaviors within complex environments. To provide support for flexible internal states, the `POSGGym.agents.Policy` class API includes a number of additional methods that provide information and finer-grained control over the policy. These methods include:

- `get_initial_state` - returns the initial state of the policy. For example, the initial belief or initial hidden neural network state.
- `get_next_state` - returns the next policy state, given the current policy state and the next observation.
- `sample_action` - sample an action given a policy state
- `get_pi` - get the distribution over actions given a policy state
- `set_state` - set the internal state of the policy
- `get_state` - get the internal state of the policy
- `get_state_from_history` - unrolls the policy to get its state given an action-observation history.

Table 6.2: POSGGym environments, their properties, and the multi-agent concepts they involve. Related properties are grouped by row color

		Classic			Grid-World							Continuous			
		MABC	MA Tiger	RPS	CR	LBF	Two Paths	UAV	Driving	Predator Prey	Pursuit Evasion	Driving	Predator Prey	Pursuit Evasion	DTC
Properties	Cooperative	x			x	x				x			x		x
	Mixed		x			x			x	x		x	x		
	Competitive			x			x	x			x			x	
	Symmetric roles	x	x	x	x	x			x	x		x	x		x
	Asymmetric roles						x	x			x			x	
	Discrete Actions	x	x	x	x	x	x	x	x	x	x	x	x	x	x
	Continuous Actions											x	x	x	x
	Discrete Observations	x	x	x	x	x	x	x	x	x	x				
	Continuous Observations											x	x	x	x
	Pixel Observations					x	x	x	x	x	x				
Concepts	Temporal Coordination	x			x	x			x	x		x	x		x
	Spacial Coordination				x	x			x	x		x	x		x
	Reciprocity					x									
	Fair Resource Sharing					x				x		x	x		
	Deception				x		x	x			x			x	
	Convention following								x			x			
	Nested-Reasoning		x	x			x	x			x			x	

All together, the API provides enough control that the policy can be used for evaluation using the `step` method, or for planning using the finer-grained control methods like `get_next_state` and `sample_action`.

6.4.2 *Environments*

POSGGym currently offers a collection of 14 environments. These environments have been used in multi-agent research in various forms, with most existing in paper descriptions, across disparate programming languages, some in unmaintained research code, and others in MARL libraries with no model support. Table 6.2 shows the complete list of environments, along with some of their properties and the multi-agent concepts they involve. For detailed explanations of each environment, we refer the reader to POSGGym's documentation[3]. Our intention is to expand the range of environments based on community demand, and actively encourage contributions from

3 Documentation available at: https://posggym.readthedocs.io/

the community. To facilitate this, we have developed comprehensive documentation and tutorials to streamline the process of adding new environments.

6.4.2.1 *Classic*

POSGGym includes several well-known problems that have been used extensively in planning and multi-agent research over the years. These include *Multi-Access Broadcast Channel* (MABC) [OW96; HBZ04], *Multi-Agent Tiger* [GD05], and *Rock-Paper-Scissors*. These problems encompass cooperative, mixed, and competitive scenarios, respectively, and support discrete actions and observations. Due to their smaller size and well-defined characteristics, some versions of these problems have known provably optimal solutions. This makes them useful for debugging and for fine-grained analysis of algorithms. POSGGym offers full model definitions for all the classic problems currently implemented in the library.

6.4.2.2 *Grid-World*

Seven widely used discrete grid-world problems are also provided by POSGGym. Many of these problems have been used throughout this thesis and encompass a variety of scenarios. The collection includes *Cooperative Reaching* (CR) [Rah+23], *Level-Based Foraging* (LBF) [CSA20; Pap+21], *Two Paths* [SZK22], *Unmanned Aerial Vehicle* (UAV) [PG17], *Driving* [McK+22; LP19], *Predatory Prey* [Tan93; Lei+17] and *Pursuit Evasion* [SvW18; SZK22]. The current selection of problems was chosen to provide a diverse range of cooperative, mixed and competitive environments, as well as symmetric and asymmetric roles. Each environment is represented as a grid-world with discrete observations and actions.

6.4.2.3 *Continuous*

POSGGym also offers four 2D continuous problems. This includes adaptations of three grid-world environments: *Driving*, *Predator Prey* and *Pursuit Evasion*. For these environments the dynamics of the agents are modeled using a simple non-holonomic unicycle model, where agents are controlled by both angular and linear velocities. Observations incorporate sensors that emit from an agents position in a circular pattern at a fixed distance. The fourth environment is *Drone Team Capture* (DTC) [De +21], which simulates a cooperative pursuit-evasion scenario. POSGGym's implementation of DTC closely adheres to the original paper, with enhancements to accommodate partial observability, such as limited sight distance.

6.4.3 *Reference Agents*

POSGGym currently offers reference agents for the majority of its environments. These agents encompass a combination of handcrafted heuristic policies and policies trained using various MARL algorithms. For instance, a number of grid-world and continuous environments include deep RL policies trained via self-play and K-Level Reasoning [Cui+21]. Additionally, handcrafted policies previously used in Level-Based Foraging [AS17; PCA21] and DTC [Ang12; Jan+17; dCV22] are included. To access the comprehensive and up-to-date list of available policies, consult the library documentation. In the following we cover the details of the general methods employed for generating reference policies.

6.4.3.1 *Reinforcement Learning Policy Training*

Every RL policy included with POSGGym to-date uses a LSTM actor-critic neural network architecture and was trained using Proximal Policy Optimization (PPO) [Sch+17]. LSTM's allow each policy to be conditioned on histories of action and observations, which is important for the majority of partially observable environments. The specific architecture used consisted of a fully-connected network (FCN) trunk, followed by a single layer LSTM, and then separate FCN actor and critic heads. The specific neural network architecture and training hyperparameters for each environment are shown in Table 6.3. For the grid-world problems, hyperparameters values were chosen based on commonly used values in the literature, since we found these worked well in general. For the continuous environments, some hyperparameter tuning was conducted to select appropriate values. The number of training steps was chosen such that policies could train until convergence, as indicated by their learning curves.

We used different multi-agent training schemes depending on the environment, while the same RL algorithm and neural network architecture was used for each individual policy. The two schemes we used were K-Level Reasoning (KLR) [Cui+21] and independent self-play with best response (SP-BR, commonly referred to as Independent PPO when using PPO as the RL algorithm) [de +20]. We used these two methods as they have been extensively studied [Lan+17; Cui+21; Tes94; de +20], are simple to implement (and thus replicate), and produced diverse populations of policies when combined with policy pruning to remove similar policies. Figure 6.6 provides a visualization of the training schemas used. The following sections contain a high level overview of each scheme.

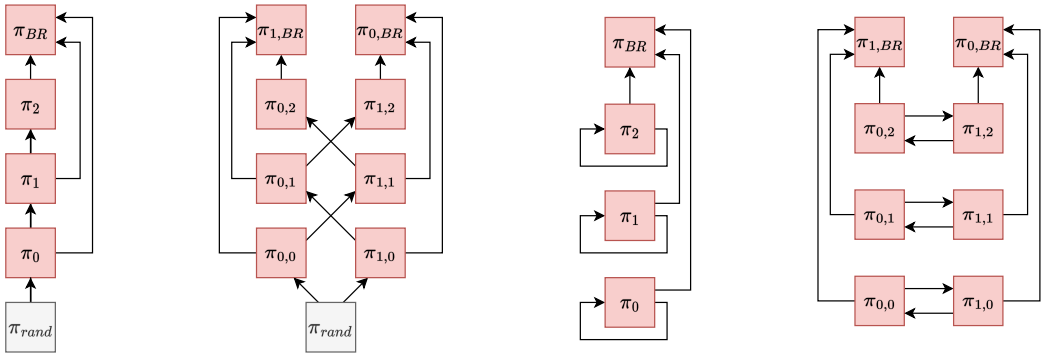
K-LEVEL REASONING In K-Level Reasoning (KLR) policies are trained in a hierarchy, the level $K = 0$ policy is trained against a uniform random policy, level $K = 1$ is trained

Table 6.3: Training hyperparameters for POSGGym RL policies in grid-world and continuous environments

Hyperparameter	Grid-world				Continuous
	Level-Based Foraging	Driving	Predator Prey	Pursuit Evasion	
Training steps	100M	32M	100M	10M	100M
Trunk layer sizes	[64, 64]	[64, 64]	[64, 64]	[64, 32]	[256, 256]
LSTM size	64	64	64	256	256
Head layer sizes	[64]	[64]	[64]	-	-
γ	0.99	0.99	0.99	0.99	0.99
Learning rate	0.0003	0.0003	0.0003	0.0003	0.0003
GAE λ	0.95	0.95	0.95	0.95	0.95
Batch size	6144	6144	6144	2048	65,536
Mini-batch size	2048	2048	2048	256	2048
Rollout horizon	64	64	64	100	100
Update epochs	2	2	2	2	2
BPTT sequence length	10	10	10	20	20
Entropy bonus	0.01	0.01	0.01	0.001	0.001
Value function coeff.	0.5	0.5	0.5	1.0	1.0
Clip parameter	0.2	0.2	0.2	0.3	0.5
Global gradient clipping	10	10	10	10	10

against the level $K = 0$, and so on with the level K policy trained as a best response to the level $K - 1$ policy for $K > 0$. Finally, the best-response policy K_{BR} is trained against all K level policies, excluding the random policy and the K_{BR} policy itself. In our implementation we used the Synchronous KLR Best-Response (SyKLRBR) training method [Cui+21] which trains all policies synchronously and was shown to converge in less total wall time and lead to generally more robust policies.

SELF-PLAY Self-play training involves training independent policies against themselves [Tes94; de +20]. For asymmetric environments this meant training a set of policies; one for each agent in the environment for each training seed. While for symmetric environments a single policy is used by all agents in the environment. In self-play Best-Response (SP-BR) an additional best-response policy π_{BR} is trained against a uniform distribution over all the independent policies trained.



(a) (b) (c) (d)

Figure 6.6: Multi-agent training schemas used for generating RL policies for POSGGym. (a) KLR in symmetric environment, (b) KLR in asymmetric environment with two agents, (c) self-play in symmetric environment, (d) self-play in asymmetric environment with two agents. Each box is an independent policy and arrows indicate which policy a given policy was trained against. Figure adapted from [Cui+21]

6.5 CONCLUSION

In this chapter we presented POSGGym, a library designed to facilitate decision-theoretic planning and RL research in partially observable, multi-agent environments. The key contributions of POSGGym are twofold: its provision of a diverse set of discrete and continuous environments complete with their dynamics models suitable for planning, and its collection of diverse reference policies. By designing the API to be similar to popular MARL libraries, users of POSGGym are easily able to utilize the growing ecosystem of tools for deep RL and apply them to planning problems. POSGGym is fully open-source and accepting contributions of new multi-agent, planning benchmarks.

CODE AND DOCUMENTATION

POSGGym is open-source and can be accessed at https://github.com/RDLLab/posggym, while all the documentation is available at https://posggym.readthedocs.io/.

7

CONCLUSION

7.1 SUMMARY

This thesis presented a set of improvements and insights for planning in partially observable, multi-agent environments:

- In Chapter 3 we developed a new online planning algorithm for more efficient planning using nested reasoning under the I-POMDP framework [GD05]. We empirically demonstrated the advantage of our algorithm, I-NTMCP, over a state-of-the-art offline solver ([HL13]) in terms of scalability to both larger environments and deeper reasoning levels.
- In Chapter 4 we presented POTMMCP: an online planning algorithm for the type-based reasoning setting. The key insight of POTMMCP was the use of a meta-policy to guide search during planning. In experiments we showed that POTMMCP outperformed existing state-of-the-art online planning methods ([Eck+20; Kak+22]), and was able to perform near-optimal in much larger environments than prior works. Additionally, we proved convergence to the optimal policy in the planning limit, assuming an accurate model of the environment and other agent types.
- In Chapter 5 we conducted an extensive empirical analysis on the benefits of integrating reinforcement learning (RL) and planning. From our experiments we were able to demonstrate that the combination of RL and planning can lead to improved performance and robustness in multi-agent, partially observable environments. Our experiments also lead to novel insights regarding where existing planning methods and their combination with RL do poorly, namely when dealing with model inaccuracy, long horizons, and large, dense belief spaces.
- In Chapter 6 we presented a new software library for facilitating planning and RL research in partially observable, multi-agent environments. Our library,

POSGGym, provides implementations of a diverse set of discrete and continuous environments, as well as a collection of reference policies. Importantly, each environment implementation includes a dynamics model to support planning.

7.2 FUTURE WORK AND OPEN PROBLEMS

In this thesis we presented a number of contributions towards imbuing agents with the ability to plan in complex, uncertain, environments. However, there are still many obstacles that must be overcome before we have agents that are capable of operating safely and robustly in more real-world applications. In this section we discuss some of the most important open problems and promising future directions for planning research, based on the lessons learned from this thesis.

7.2.1 *Model inaccuracy*

For the most part, this thesis has focused on techniques for improving planning as it is applied to more complex environments. Scaling to problems with more states, observations, and actions is vital for planning to be practical for real-world applications. The contributions of this thesis have mainly been in the setting where we have a perfect model of the environment dynamics, in order to focus on improving scaling.

Another key challenge for planning in real-world domains is dealing with model inaccuracy, which is almost inevitable when modeling any domain of sufficient complexity. This is especially true in multi-agent environments where modeling other agents can be particularly difficult since the space of possible behaviors is essentially infinite if we consider stochastic policies. We demonstrated the negative effect of inaccurate other agent models in Chapter 5, where an inaccurate model led to a large drop in performance and, in some cases, even worsening performance as planning time increased.

Developing more robust models of the other agents, and better methods for handling model inaccuracy are both important open problems for multi-agent planning. An area which shows promise for improving other agent models is research into techniques for creating diverse populations of agents [Lan+17; Lup+21; Xin+21; Rah+23]. So far, much of the research in this direction has been done within the area of RL. Integrating existing results into planning is a future avenue we believe is particularly promising for improving robustness to inaccuracy of other agent models.

7.2.2 *Better belief representations*

Finding ways to efficiently represent beliefs for planning in partially observable environments has been an active area of research for the field for many years. In Chapter 5, we found that even if we have a perfect model of the environment and the other agent, planning can still suffer due to poor belief accuracy under more complex belief spaces. Our experiments also found belief updates can require significant computation, depending on the methods used. For these reasons, there is a real need for more efficient, general methods to represent beliefs. One promising direction is learning belief representations using deep learning methods, with recent work in this direction showing positive results [Hu+21; Mor+21]. An open question is how to learn general belief models that can handle complex state spaces, and which can be reliably sampled from in a way that can be used in planning. There may also be other types of belief representations beyond particle beliefs that help improve the efficiency specifically in multi-agent settings. We need planning agents that can scale to real-world applications with complex state and observation spaces. A core part of creating these is finding better belief representations.

7.2.3 *Leveraging compute for planning*

The ability to leverage large amounts of compute has been a key driver of the success of deep learning. This ability stems from deep learning's use of batched computations which can be efficiently performed by hardware accelerators such as GPUs and TPUs [Jou+17]. Compared to areas like computer vision, natural language processing, and RL, much less work has been done on hardware accelerated planning. While developing more sample efficient planning techniques is important for reducing the required compute to find good solutions, fundamentally there is no escaping that scaling planning to larger environments will require greater compute. Fortunately, there is an opportunity to leverage many of the tools from the modern deep learning stack for planning. Indeed there has already been some work in this direction, taking advantage of general hardware accelerator libraries such as JAX [Bra+18] which can be applied beyond just deep learning work loads. For example, JAX has been applied to more classical planning [Tai+23], while hybrid planning under uncertainty methods have been developed that use GPUs for some computations in the single agent setting [Cai+21]. More work along these lines would be a huge contribution for the planning research community.

7.3 CONCLUSION

A key motivation for doing research on artificial intelligence is the creation of autonomous agents that help humans achieve their goals in the real-world. For these agents to be useful, they must be capable, robust, and safe. These agents must be able to handle the messiness and uncertainty of the real-world, in particular uncertainty stemming from partial observability, inherent stochasticity, and the actions of other independent agents. Furthermore, to be capable of safely performing more than basic, memorized, repeated tasks, agents will require the ability to plan, that is given a new task, use a model of the world to formulate the best sequence of actions to take to complete the task.

The aim of this thesis was to contribute towards imbuing agents with the ability to plan in uncertain, real-world environments. In pursuit of this aim, we developed methods that allowed planning to be tractably applied to larger and more complex problems. However, the challenge of imbuing agents with the ability to plan is far from solved. Our hope is the ideas presented in this thesis will help along the path to creating capable and safe autonomous agents, that are deployed to benefit as much of humanity as possible.

BIBLIOGRAPHY

[ACR16] Stefano Albrecht, Jacob Crandall, and Subramanian Ramamoorthy. "Belief and Truth in Hypothesised Behaviours". In: *Artificial Intelligence* 235 (2016), pp. 63–94.

[ALS17] Stefano Albrecht, Somchaya Liemhetcharat, and Peter Stone. "Special Issue on Multiagent Interaction without Prior Coordination: Guest Editorial". In: *Autonomous Agents and Multiagent Systems* 31.4 (2017), pp. 765–766.

[AR13] Stefano Albrecht and Subramanian Ramamoorthy. "A Game-Theoretic Model and Best-Response Learning Method for Ad Hoc Coordination in Multiagent Systems". In: *Autonomous Agents and Multiagent Systems*. IFAAMAS, 2013, pp. 1155–1156.

[AR14] Stefano Albrecht and Subramanian Ramamoorthy. "On Convergence and Optimality of Best-Response Learning with Policy Types in Multiagent Systems". In: *Uncertainty in Artificial Intelligence*. 2014, pp. 12–21.

[AS17] Stefano Albrecht and Peter Stone. "Reasoning about Hypothetical Agent Behaviours and Their Parameters". In: *Autonomous Agents and Multiagent Systems*. IFAAMAS, 2017, pp. 547–555.

[AS18] Stefano Albrecht and Peter Stone. "Autonomous Agents Modelling Other Agents: A Comprehensive Survey and Open Problems". In: *Artificial Intelligence* 258 (2018), pp. 66–95.

[ABZ10] Christopher Amato, Daniel S. Bernstein, and Shlomo Zilberstein. "Optimizing Fixed-Size Stochastic Controllers for POMDPs and Decentralized POMDPs". In: *Autonomous Agents and Multiagent Systems* 21.3 (2010), pp. 293–320.

[ADZ09] Christopher Amato, Jilles Dibangoye, and Shlomo Zilberstein. "Incremental Policy Generation for Finite-Horizon DEC-POMDPs". In: *International Conference on Automated Planning and Scheduling*. Vol. 19. 2009.

[AO14] Christopher Amato and Frans A. Oliehoek. "Scalable Planning and Learning for Multiagent POMDPs: Extended Version". In: *AAAI Conference on Artificial Intelligence*. Vol. 29. 1. 2014.

[Ang12] Luca Angelani. "Collective Predation and Escape Strategies". In: *Physical Review Letters* 109.11 (2012).

[AD10] Raghav Aras and Alain Dutech. "An Investigation into Mathematical Programming for Finite Horizon Decentralized POMDPs". In: *Journal of Artificial Intelligence Research* 37 (2010), pp. 329–396.

[APF22] Pouyan Asgharian, Adina M. Panchea, and François Ferland. "A Review on the Use of Mobile Service Robots in Elderly Care". In: *Robotics* 11.6 (2022), p. 127.

[Ast65] Karl J. Astrom. "Optimal Control of Markov Decision Processes with Incomplete State Estimation". In: *J. Math. Anal. Applic.* 10 (1965), pp. 174–205.

[ACF02] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. "Finite-Time Analysis of the Multiarmed Bandit Problem". In: *Machine learning* 47.2 (2002), pp. 235–256.

[Aum99] Robert J. Aumann. "Interactive Epistemology I: Knowledge". In: *International Journal of Game Theory* 28 (1999), pp. 263–300.

[Bak+20] Bowen Baker et al. "Emergent Tool Use From Multi-Agent Autocurricula". In: *International Conference on Learning Representations*. 2020.

[Bak+22] Anton Bakhtin et al. "Mastering the Game of No-Press Diplomacy via Human-Regularized Reinforcement Learning and Planning". In: *International Conference on Learning Representations*. 2022.

[Bam21] Christopher Bamford. "Griddly: A Platform for AI Research in Games". In: *Software Impacts* 8 (2021), p. 100066.

[Bar+20] Nolan Bard et al. "The Hanabi Challenge: A New Frontier for Ai Research". In: *Artificial Intelligence* 280 (2020), p. 103216.

[BS15] Samuel Barrett and Peter Stone. "Cooperating with Unknown Teammates in Complex Domains: A Robot Soccer Case Study of Ad Hoc Teamwork". In: *AAAI Conference on Artificial Intelligence*. 2015.

[BSK11] Samuel Barrett, Peter Stone, and Sarit Kraus. "Empirical Evaluation of Ad Hoc Teamwork in the Pursuit Domain". In: *Autonomous Agents and Multiagent Systems*. IFAAMAS, 2011, pp. 567–574.

[Bar+14] Samuel Barrett et al. "Communicating with Unknown Teammates." In: *Autonomous Agents and Multiagent Systems*. IFAAMAS, 2014, pp. 1433–1434.

[Bea+20] Charles Beattie et al. "Deepmind Lab2d". In: *arXiv preprint arXiv:2011.07027* (2020). arXiv: 2011.07027.

[Bel57] Richard Bellman. "A Markovian Decision Process". In: *Journal of mathematics and mechanics* (1957), pp. 679–684.

[BSF94] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. "Learning Long-Term Dependencies with Gradient Descent Is Difficult". In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166.

[Ber+19] Christopher Berner et al. "Dota 2 with Large Scale Deep Reinforcement Learning". In: *arXiv preprint arXiv:1912.06680* (2019). arXiv: 1912.06680.

[Ber+02] Daniel S. Bernstein et al. "The Complexity of Decentralized Control of Markov Decision Processes". In: *Mathematics of Operations Research* 27.4 (2002), pp. 819–840.

[BC99] Dimitri P. Bertsekas and David A. Castanon. "Rollout Algorithms for Stochastic Scheduling Problems". In: *Journal of Heuristics* 5 (1999), pp. 89–108.

[Bha+09] Shalabh Bhatnagar et al. "Natural Actor–Critic Algorithms". In: *Automatica* 45.11 (2009), pp. 2471–2482.

[Bly99] Jim Blythe. "Decision-Theoretic Planning". In: *AI magazine* 20.2 (1999), pp. 37–37.

[Bom+22] Rishi Bommasani et al. *On the Opportunities and Risks of Foundation Models*. 2022. arXiv: 2108.07258.

[BC08] Abdeslam Boularias and Brahim Chaib-Draa. "Exact Dynamic Programming for Decentralized POMDPs with Lossless Policy Compression". In: *International Conference on Automated Planning and Scheduling*. 2008, pp. 20–27.

[Bou96] Craig Boutilier. "Planning, Learning and Coordination in Multiagent Decision Processes". In: *Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge*. 1996, pp. 195–210.

[BDH99] Craig Boutilier, Thomas Dean, and Steve Hanks. "Decision-Theoretic Planning: Structural Assumptions and Computational Leverage". In: *Journal of Artificial Intelligence Research* 11 (1999), pp. 1–94.

[BM05] Michael Bowling and Peter McCracken. "Coordination and Adaptation in Impromptu Teams". In: *AAAI Conference on Artificial Intelligence*. Vol. 5. 2005, pp. 53–58.

[Bra+18] James Bradbury et al. *JAX: Composable Transformations of Python+NumPy Programs*. 2018.

[Bro+16] Greg Brockman et al. "Openai Gym". In: *arXiv preprint arXiv:1606.01540* (2016). arXiv: 1606.01540.

[BS18] Noam Brown and Tuomas Sandholm. "Superhuman AI for Heads-up No-Limit Poker: Libratus Beats Top Professionals". In: *Science* 359.6374 (2018), pp. 418–424.

[BS19] Noam Brown and Tuomas Sandholm. "Superhuman AI for Multiplayer Poker". In: *Science* 365.6456 (2019), pp. 885–890.

[Bro+20a] Noam Brown et al. "Combining Deep Reinforcement Learning and Search for Imperfect-Information Games". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 17057–17069.

[Bro+20b] Tom Brown et al. "Language Models Are Few-Shot Learners". In: *Neural Information Processing Systems*. Vol. 33. 2020, pp. 1877–1901.

[BBD08] Lucian Busoniu, Robert Babuska, and Bart De Schutter. "A Comprehensive Survey of Multiagent Reinforcement Learning". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.2 (2008), pp. 156–172.

[Cai+21] Panpan Cai et al. "HyP-DESPOT: A Hybrid Parallel Algorithm for Online Planning under Uncertainty". In: *The International Journal of Robotics Research* 40.2-3 (2021), pp. 558–573.

[CHC04] Colin F. Camerer, Teck-Hua Ho, and Juin-Kuan Chong. "A Cognitive Hierarchy Model of Games". In: *The Quarterly Journal of Economics* 119.3 (2004), pp. 861–898.

[Car+21] Steven Carr et al. "Safe Policies for Factored Partially Observable Stochastic Games". In: *Robotics: Science and Systems*. 2021.

[Cas98] Anthony R. Cassandra. "A Survey of POMDP Applications". In: *Working Notes of AAAI 1998 Fall Symposium on Planning with Partially Observable Markov Decision Processes*. Vol. 1724. 1998.

[Cho+22] Shushman Choudhury et al. "Scalable Online Planning for Multi-Agent MDPs". In: *Journal of Artificial Intelligence Research* 73 (2022), pp. 821–846.

[CSA20] Filippos Christianos, Lukas Schäfer, and Stefano Albrecht. "Shared Experience Actor-Critic for Multi-Agent Reinforcement Learning". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 10707–10717.

[Cog+06] Randy Cogill et al. "An Approximate Dynamic Programming Approach to Decentralized Control of Stochastic Systems". In: *Control of Uncertain Systems: Modelling, Approximation, and Design*. Ed. by Bruce A. Francis, Malcolm C. Smith, and Jan C. Willems. Vol. 329. Berlin/Heidelberg: Springer-Verlag, 2006, pp. 243–256. ISBN: 978-3-540-31754-8.

[Cou06] Rémi Coulom. "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search". In: *International Conference on Computers and Games*. 2006, pp. 72–83.

[CPW12] Peter I. Cowling, Edward J. Powley, and Daniel Whitehouse. "Information Set Monte Carlo Tree Search". In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.2 (2012), pp. 120–143.

[Cui+21] Brandon Cui et al. "K-Level Reasoning for Zero-Shot Coordination in Hanabi". In: *Advances in Neural Information Processing Systems* 34 (2021).

[CO21] Aleksander Czechowski and Frans A. Oliehoek. "Decentralized MCTS via Learned Teammate Models". In: *International Joint Conference on Artificial Intelligence*. 2021, pp. 81–88.

[dCV22] C. de Souza, Pedro Castillo, and B Vidolov. "Local Interaction and Navigation Guidance for Hunters Drones: A Chase Behavior Approach with Real-Time Tests". In: *Robotica* 40.8 (2022), pp. 2697–2715.

[De +21] Cristino De Souza et al. "Decentralized Multi-Agent Pursuit Using Deep Reinforcement Learning". In: *IEEE Robotics and Automation Letters* 6.3 (2021), pp. 4552–4559.

[dVV13] Harmen de Weerd, Rineke Verbrugge, and Bart Verheij. "How Much Does It Help to Know What She Knows You Know? An Agent-Based Simulation Study". In: *Artificial Intelligence* 199–200 (2013), pp. 67–92.

[de +20] de Witt, Christian Schroeder et al. "Is Independent Learning All You Need in the StarCraft Multi-Agent Challenge?" In: *arXiv preprint arXiv:2011.09533* (2020). arXiv: 2011.09533.

[DGB09] Antonio Del Giudice, Piotr J. Gmytrasiewicz, and Josh Bryan. "Towards Strategic Kriegspiel Play with Opponent Modeling." In: *Autonomous Agents and Multiagent Systems*. 2009, pp. 1265–1266.

[Dev+19] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *North American Chapter of the Association for Computational Linguistics*. 2019.

[Dib+16] Jilles Steeve Dibangoye et al. "Optimally Solving Dec-POMDPs as Continuous-State MDPs". In: *Journal of Artificial Intelligence Research* 55 (2016), pp. 443–497.

[do +22] Matheus Aparecido do Carmo Alves et al. "AdLeap-MAS: An Open-source Multi-Agent Simulator for Ad-hoc Reasoning". In: *Autonomous Agents and Multiagent Systems*. IFAAMAS, 2022, pp. 1893–1895.

[DG09] P. Doshi and P. J. Gmytrasiewicz. "Monte Carlo Sampling Methods for Approximating Interactive POMDPs". In: *Journal of Artificial Intelligence Research* 34 (2009), pp. 297–337.

[DP08] Prashant Doshi and Dennis Perez. "Generalized Point Based Value Iteration for Interactive POMDPs." In: *AAAI Conference on Artificial Intelligence*. 2008, pp. 63–68.

[Dos+10] Prashant Doshi et al. "Modeling Recursive Reasoning by Humans Using Empirically Informed Interactive POMDPs". In: *Autonomous Agents and Multiagent Systems*. 2010, pp. 1223–1230.

[DZ13] Edmund Durfee and Shlomo Zilberstein. "Multiagent Planning, Control, and Execution". In: *Multiagent systems* 11 (2013), pp. 485–545.

[Eck+20] Adam Eck et al. "Scalable Decision-Theoretic Planning in Open and Typed Multiagent Systems". In: *AAAI Conference on Artificial Intelligence*. Vol. 34. 2020, pp. 7127–7134.

[Ell+24] Benjamin Ellis et al. "SMACv2: An Improved Benchmark for Cooperative Multi-Agent Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. Vol. 36. 2024.

[Fic+21] Arnaud Fickinger et al. "Scalable Online Planning via Reinforcement Learning Fine-Tuning". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 16951–16963.

[Fou22] Farama Foundation. *Gymnasium*. 2022.

[Gha+15] Mohammad Ghavamzadeh et al. "Bayesian Reinforcement Learning: A Survey". In: *Foundations and Trends® in Machine Learning* 8.5-6 (2015), pp. 359–483.

[Gle+19] Adam Gleave et al. "Adversarial Policies: Attacking Deep Reinforcement Learning". In: *International Conference on Learning Representations*. 2019.

[GD05] Piotr J. Gmytrasiewicz and Prashant Doshi. "A Framework for Sequential Planning in Multi-Agent Settings". In: *Journal of Artificial Intelligence Research* 24 (2005), pp. 49–79.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning Book*. MIT Press, 2016.

[GSD13] Arthur Guez, David Silver, and Peter Dayan. "Scalable and Efficient Bayes-adaptive Reinforcement Learning Based on Monte-Carlo Tree Search". In: *Journal of Artificial Intelligence Research* 48 (2013), pp. 841–883.

[HG18] Yanlin Han and Piotr Gmytrasiewicz. "Learning Others' Intentional Models in Multi-Agent Settings Using Interactive POMDPs". In: *Advances in Neural Information Processing Systems* 31 (2018).

[HG19] Yanlin Han and Piotr Gmytrasiewicz. "IPOMDP-Net: A Deep Neural Network for Partially Observable Multi-Agent Planning Using Interactive POMDPs". In: *AAAI Conference on Artificial Intelligence* 33.1 (2019), pp. 6062–6069.

[HBZ04] Eric A. Hansen, Daniel S. Bernstein, and Shlomo Zilberstein. "Dynamic Programming for Partially Observable Stochastic Games". In: *National Conference on Artifical Intelligence*. 2004, pp. 709–715.

[HS15] Matthew Hausknecht and Peter Stone. "Deep Recurrent Q-Learning for Partially Observable MDPs". In: *AAAI Fall Symposium Series*. 2015.

[He+16] He He et al. "Opponent Modeling in Deep Reinforcement Learning". In: *International Conference on Machine Learning*. PMLR, 2016, pp. 1804–1813.

[HLS15] Johannes Heinrich, Marc Lanctot, and David Silver. "Fictitious Self-Play in Extensive-Form Games". In: *International Conference on Machine Learning*. PMLR, 2015, pp. 805–813.

[HL13] Trong Nghia Hoang and Kian Hsiang Low. "Interactive POMDP Lite: Towards Practical Planning to Predict and Exploit Intentions for Interacting with Self-Interested Agents". In: *International Joint Conference on Artificial Intelligence*. 2013.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[HK21] Marcus Hoerger and Hanna Kurniawati. "An On-Line POMDP Solver for Continuous Observation Spaces". In: *International Conference on Robotics and Automation*. IEEE, 2021, pp. 7643–7649.

[HB19] Karel Horák and Branislav Bošanský. "Solving Partially Observable Stochastic Games with Public Observations". In: *AAAI Conference on Artificial Intelligence* 33.01 (2019), pp. 2029–2036.

[Hu+20] Hengyuan Hu et al. ""Other-Play" for Zero-Shot Coordination". In: *International Conference on Machine Learning*. PMLR, 2020, pp. 4399–4410.

[Hu+21] Hengyuan Hu et al. "Learned Belief Search: Efficiently Improving Policies in Partially Observable Settings". In: *arXiv preprint arXiv:2106.09086* (2021). arXiv: 2106.09086.

[Hua+22] Shengyi Huang et al. "CleanRL: High-quality Single-File Implementations of Deep Reinforcement Learning Algorithms". In: *The Journal of Machine Learning Research* 23.1 (2022), pp. 12585–12602.

[Hut04] Marcus Hutter. *Universal Artificial Intelligence: Sequential Decisions Based on Algorithmic Probability*. Springer Science & Business Media, 2004.

[Hut09] Marcus Hutter. "Feature Reinforcement Learning: Part I. Unstructured MDPs". In: *Journal of Artificial General Intelligence* 1 (2009), pp. 3–24.

[Jad+19] Max Jaderberg et al. "Human-Level Performance in 3D Multiplayer Games with Population-Based Reinforcement Learning". In: *Science* 364.6443 (2019), pp. 859–865.

[Jan+17] Milán Janosov et al. "Group Chasing Tactics: How to Catch a Faster Prey". In: *New Journal of Physics* (2017).

[Jou+17] Norman P. Jouppi et al. "In-Datacenter Performance Analysis of a Tensor Processing Unit". In: *The Annual International Symposium on Computer Architecture*. ACM, 2017, pp. 1–12. ISBN: 978-1-4503-4892-8. DOI: 10.1145/3079856.3080246.

[Jul+18] Arthur Juliani et al. "Unity: A General Platform for Intelligent Agents". In: *arXiv preprint arXiv:1809.02627* (2018). arXiv: 1809.02627.

[KLC98] L.P. Kaelbling, M.L. Littman, and A.R. Cassandra. "Planning and Acting in Partially Observable Stochastic Domains". In: *Artificial Intelligence* 101.1-2 (1998), pp. 99–134.

[Kak+22] Anirudh Kakarlapudi et al. "Decision-Theoretic Planning with Communication in Open Multiagent Systems". In: *Uncertainty in Artificial Intelligence*. PMLR, 2022, pp. 938–948.

[Kap+20] Jared Kaplan et al. "Scaling Laws for Neural Language Models". In: *arXiv preprint arXiv:2001.08361* (2020). arXiv: 2001.08361.

[KOA17] Sammie Katt, Frans A. Oliehoek, and Christopher Amato. "Learning in POMDPs with Monte Carlo Tree Search". In: *International Conference on Machine Learning*. PMLR, 2017, pp. 1819–1827.

[KMN02] Michael Kearns, Yishay Mansour, and Andrew Y. Ng. "Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes". In: *Machine Learning* 49.2/3 (2002), pp. 193–208.

[KS06] Levente Kocsis and Csaba Szepesvári. "Bandit Based Monte-Carlo Planning". In: *European Conference on Machine Learning*. 2006, pp. 282–293.

[Koy+24] Sotetsu Koyamada et al. “Pgx: Hardware-Accelerated Parallel Game Simulators for Reinforcement Learning”. In: *Advances in Neural Information Processing Systems*. Vol. 36. 2024.

[KZ09] Akshat Kumar and Shlomo Zilberstein. “Dynamic Programming Approximations for Partially Observable Stochastic Games”. In: *International FLAIRS Conference*. 2009, pp. 547–552.

[Kur+20] Karol Kurach et al. “Google Research Football: A Novel Reinforcement Learning Environment”. In: *AAAI Conference on Artificial Intelligence*. Vol. 34. 2020, pp. 4501–4510.

[Kur22] Hanna Kurniawati. “Partially Observable Markov Decision Processes and Robotics”. In: *Annual Review of Control, Robotics, and Autonomous Systems* 5.1 (2022), pp. 253–277.

[KHL08] Hanna Kurniawati, David Hsu, and Wee Sun Lee. “SARSOP: Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces.” In: *Robotics: Science and Systems*. Vol. 2008. 2008.

[KY16] Hanna Kurniawati and Vinay Yadav. “An Online POMDP Solver for Uncertainty Planning in Dynamic Environment”. In: *Robotics Research*. Springer, 2016, pp. 611–629.

[Lan+17] Marc Lanctot et al. “A Unified Game-Theoretic Approach to Multiagent Reinforcement Learning”. In: *Advances in Neural Information Processing Systems* 30 (2017).

[Lan+19] Marc Lanctot et al. “OpenSpiel: A Framework for Reinforcement Learning in Games”. In: *arXiv preprint arXiv:1908.09453* (2019). arXiv: 1908.09453.

[Lec+23] Mathias Lechner et al. “Gigastep - One Billion Steps per Second Multi-agent Reinforcement Learning”. In: *Neural Information Processing Systems Datasets and Benchmarks*. 2023.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep Learning”. In: *Nature* 521.7553 (2015), pp. 436–444.

[Lei+17] Joel Leibo et al. “Multi-Agent Reinforcement Learning in Sequential Social Dilemmas”. In: *Autonomous Agents and Multiagent Systems*. Vol. 16. IFAAMAS, 2017, pp. 464–473.

[Lei+21] Joel Leibo et al. “Scalable Evaluation of Multi-Agent Reinforcement Learning with Melting Pot”. In: *International Conference on Machine Learning*. PMLR, 2021, pp. 6187–6199.

[LP19] Adam Lerer and Alexander Peysakhovich. "Learning Existing Social Conventions via Observationally Augmented Self-Play". In: *AAAI/ACM Conference on AI, Ethics, and Society*. 2019, pp. 107–114.

[Ler+20] Adam Lerer et al. "Improving Policies via Search in Cooperative Partially Observable Games". In: *AAAI Conference on Artificial Intelligence* 34.05 (2020), pp. 7187–7194.

[LS08] Kevin Leyton-Brown and Yoav Shoham. "Essentials of Game Theory: A Concise Multidisciplinary Introduction". In: *Synthesis lectures on artificial intelligence and machine learning* 2.1 (2008), pp. 1–88.

[Li+23] Zun Li et al. "Combining Tree-Search, Generative Models, and Nash Bargaining Concepts in Game-Theoretic Reinforcement Learning". In: *arXiv preprint arXiv:2302.00797* (2023). arXiv: 2302.00797.

[Lia+18] Eric Liang et al. "RLlib: Abstractions for Distributed Reinforcement Learning". In: *International Conference on Machine Learning*. PMLR, 2018, pp. 3053–3062.

[Lop+18] Pablo Alvarez Lopez et al. "Microscopic Traffic Simulation Using Sumo". In: *International Conference on Intelligent Transportation Systems*. 2018, pp. 2575–2582.

[Low+17] Ryan Lowe et al. "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments". In: *Advances in Neural Information Processing Systems* 30 (2017).

[Lup+21] Andrei Lupu et al. "Trajectory Diversity for Zero-Shot Coordination". In: *International Conference on Machine Learning*. PMLR, 2021, pp. 7204–7213.

[McK+22] Kevin McKee et al. "Quantifying the Effects of Environment and Population Diversity in Multi-Agent Reinforcement Learning". In: *Autonomous Agents and Multiagent Systems* 36.1 (2022), pp. 1–16.

[Mei11] C. Meissner. "A Complex Game of Cat and Mouse". In: *Lawrence Livermore National Laboratory Science and Technology Review* (2011), pp. 18–21.

[Mni+15] Volodymyr Mnih et al. "Human-Level Control through Deep Reinforcement Learning". In: *Nature* 518.7540 (2015), pp. 529–533.

[Mor+21] Pol Moreno et al. "Neural Recursive Belief States in Multi-Agent Reinforcement Learning". In: *arXiv preprint arXiv:2102.02274* (2021). arXiv: 2102.02274.

[Nai+05] Ranjit Nair et al. "Networked Distributed POMDPs: A Synthesis of Distributed Constraint Optimization and POMDPs". In: *AAAI Conference on Artificial Intelligence*. Vol. 5. AAAI Press, 2005, pp. 133–139.

[Ng+10] Brenda Ng et al. "Towards Applying Interactive POMDPs to Real-World Adversary Modeling". In: *Innovative Applications of Artificial Intelligence*. 2010.

[Ng+12] Brenda Ng et al. "Bayes-Adaptive Interactive POMDPs". In: *AAAI Conference on Artificial Intelligence*. Vol. 26. 2012.

[OA16] Frans A. Oliehoek and Christopher Amato. *A Concise Introduction to Decentralized POMDPs*. Springer International Publishing, 2016.

[OSV08] Frans A. Oliehoek, Matthijs TJ Spaan, and Nikos Vlassis. "Optimal and Approximate Q-value Functions for Decentralized POMDPs". In: *Journal of Artificial Intelligence Research* 32 (2008), pp. 289–353.

[Oli+13] Frans A. Oliehoek et al. "Incremental Clustering and Expansion for Faster Optimal Planning in Dec-POMDPs". In: *Journal of Artificial Intelligence Research* 46 (2013), pp. 449–509.

[OW96] James Ooi and Gregory Wornell. "Decentralized Control of a Multiple Access Broadcast Channel: Performance Bounds". In: *IEEE Conference on Decision and Control*. Vol. 1. 1996, pp. 293–298. DOI: 10.1109/CDC.1996.574318.

[OR94] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory*. MIT press, 1994.

[PG16] Alessandro Panella and Piotr Gmytrasiewicz. "Bayesian Learning of Other Agents' Finite Controllers for Interactive POMDPs". In: *AAAI Conference on Artificial Intelligence*. 2016, pp. 2530–2536.

[PG17] Alessandro Panella and Piotr Gmytrasiewicz. "Interactive POMDPs with Finite-State Models of Other Agents". In: *Autonomous Agents and Multiagent Systems* 31.4 (2017), pp. 861–904.

[PT87] Christos H. Papadimitriou and John N. Tsitsiklis. "The Complexity of Markov Decision Processes". In: *Mathematics of Operations Research* 12.3 (1987), pp. 441–450.

[PCA21] Georgios Papoudakis, Filippos Christianos, and Stefano Albrecht. "Agent Modelling under Partial Observability for Deep Reinforcement Learning". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 19210–19222.

[Pap+21] Georgios Papoudakis et al. "Benchmarking Multi-Agent Deep Reinforcement Learning Algorithms in Cooperative Tasks". In: *Neural Information Processing Systems Datasets and Benchmarks* (2021).

[PTC05] Sébastien Paquet, Ludovic Tobin, and Brahim Chaib-draa. "An Online POMDP Algorithm for Complex Multiagent Environments". In: *Autonomous Agents and Multiagent Systems*. ACM, 2005, pp. 970–977.

[Pen+21] Bei Peng et al. "Facmac: Factored Multi-Agent Centralised Policy Gradients". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 12208–12221.

[Per+22] Julien Perolat et al. "Mastering the Game of Stratego with Model-Free Multiagent Reinforcement Learning". In: *Science* 378.6623 (2022), pp. 990–996.

[PS08] Jan Peters and Stefan Schaal. "Natural Actor-Critic". In: *Neurocomputing* 71.7-9 (2008), pp. 1180–1190.

[Pet+20] Aleksei Petrenko et al. "Sample Factory: Egocentric 3d Control from Pixels at 100000 Fps with Asynchronous Reinforcement Learning". In: *International Conference on Machine Learning*. PMLR, 2020, pp. 7652–7662.

[PGT06] Joelle Pineau, Geoffrey Gordon, and Sebastian Thrun. "Anytime Point-Based Approximations for Large POMDPs". In: *Journal of Artificial Intelligence Research* 27 (2006), pp. 335–380.

[Put14] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

[PT02] David V. Pynadath and Milind Tambe. "The Communicative Multiagent Team Decision Problem: Analyzing Teamwork Theories and Models". In: *Journal of Artificial Intelligence Research* 16 (2002), pp. 389–423.

[Raf+21] Antonin Raffin et al. "Stable-Baselines3: Reliable Reinforcement Learning Implementations". In: *The Journal of Machine Learning Research* 22.1 (2021), pp. 12348–12355.

[Rah+23] Arrasy Rahman et al. "Generating Diverse Teammates to Train Robust Agents For Ad Hoc Teamwork". In: *Transactions on Machine Learning Research* (2023).

[RDG06] Bharanee Rathnasabapathy, Prashant Doshi, and Piotr Gmytrasiewicz. "Exact Solutions of Interactive POMDPs Using Behavioral Equivalence". In: *Autonomous Agents and Multiagent Systems*. IFAAMAS, 2006, pp. 1025–1032.

[Res+18] Cinjon Resnick et al. "Pommerman: A Multi-Agent Playground". In: *arXiv preprint arXiv:1809.07124* (2018). arXiv: 1809.07124.

[Ros11] Christopher D. Rosin. "Multi-Armed Bandits with Episode Context". In: *Annals of Mathematics and Artificial Intelligence* 61.3 (2011), pp. 203–230.

[RC07] Stéphane Ross and Brahim Chaib-Draa. "AEMS: An Anytime Online Search Algorithm for Approximate Policy Refinement in Large POMDPs." In: *International Joint Conference on Artificial Intelligence*. 2007, pp. 2592–2598.

[Ros+08] Stéphane Ross et al. "Online Planning Algorithms for POMDPs". In: *Journal of Artificial Intelligence Research* 32 (2008), pp. 663–704.

[RGT05] Nicholas Roy, Geoffrey Gordon, and Sebastian Thrun. "Finding Approximate POMDP Solutions through Belief Compression". In: *Journal of artificial intelligence research* 23 (2005), pp. 1–40.

[Rut+23] Alexander Rutherford et al. "JaxMARL: Multi-Agent RL Environments in JAX". In: *arXiv preprint arXiv:2311.10090* (2023). arXiv: 2311.10090.

[Sam+19] Mikayel Samvelyan et al. "The StarCraft Multi-Agent Challenge". In: *Autonomous Agents and Multiagent Systems*. IFAAMAS, 2019, pp. 2186–2188.

[Sch+20] Julian Schrittwieser et al. "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model". In: *Nature* 588.7839 (2020), pp. 604–609.

[Sch+17] John Schulman et al. "Proximal Policy Optimization Algorithms". In: *arXiv preprint arXiv:1707.06347* (2017). arXiv: 1707.06347.

[SK23] Jonathon Schwartz and Hanna Kurniawati. "Bayes-Adaptive Monte-Carlo Planning for Type-Based Reasoning in Large Partially Observable, Multi-Agent Environments". In: *Autonomous Agents and Multiagent Systems*. IFAAMAS, 2023, pp. 2355–2357.

[SKH23] Jonathon Schwartz, Hanna Kurniawati, and Marcus Hutter. "Combining a Meta-Policy and Monte-Carlo Planning for Scalable Type-Based Reasoning in Partially Observable Environments". In: *arXiv preprint arXiv:2306.06067* (2023). arXiv: 2306.06067.

[SZK22] Jonathon Schwartz, Ruijia Zhou, and Hanna Kurniawati. "Online Planning for Interactive-Pomdps Using Nested Monte Carlo Tree Search". In: *International Conference on Intelligent Robots and Systems*. IEEE, 2022, pp. 8770–8777.

[Sch+24] Jonathon Schwartz et al. "POSGGym: A Library for Decision-Theoretic Planning and Learning in Partially Observable, Multi-Agent Environments". In: *International Conference on Automated Planning and Scheduling PRL Workshop*. 2024.

[Sch+25] Jonathon Schwartz et al. "POSGGym: A Library for Decision-Theoretic Planning and Learning in Partially Observable, Multi-Agent Environments". In: *Autonomous Agents and Multi-Agent Systems* 39.2 (2025), p. 35. ISSN: 1573-7454. DOI: 10.1007/s10458-025-09716-6.

[SvW18] Iris Rubi Seaman, Jan-Willem van de Meent, and David Wingate. "Nested Reasoning About Autonomous Agents Using Probabilistic Programs". In: *arXiv preprint arXiv:1812.01569* (2018). arXiv: 1812.01569.

[SZ08] Sven Seuken and Shlomo Zilberstein. "Formal Models and Algorithms for Decentralized Decision Making under Uncertainty". In: *Autonomous Agents and Multi-Agent Systems* 17 (2008), pp. 190–250.

[SP09a] Richard Seymour and Gilbert L. Peterson. "A Trust-Based Multiagent System". In: *International Conference on Computational Science and Engineering*. Vol. 3. IEEE, 2009, pp. 109–116.

[SP09b] Richard S. Seymour and Gilbert L. Peterson. "Responding to Sneaky Agents in Multi-Agent Domains". In: *International FLAIRS Conference*. FLAIR, 2009.

[SPK13] Guy Shani, Joelle Pineau, and Robert Kaplow. "A Survey of Point-Based POMDP Solvers". In: *Autonomous Agents and Multi-Agent Systems* 27.1 (2013), pp. 1–51.

[Sha53] Lloyd S. Shapley. "Stochastic Games". In: *Proceedings of the national academy of sciences* 39.10 (1953), pp. 1095–1100.

[SPG07] Yoav Shoham, Rob Powers, and Trond Grenager. "If Multi-Agent Learning Is the Answer, What Is the Question?" In: *Artificial intelligence* 171.7 (2007), pp. 365–377.

[SV10] David Silver and Joel Veness. "Monte-Carlo Planning in Large POMDPs". In: *Advances in Neural Information Processing Systems* 23 (2010), pp. 2164–2172.

[Sil+16] David Silver et al. "Mastering the Game of Go with Deep Neural Networks and Tree Search". In: *Nature* 529.7587 (2016), pp. 484–489.

[Sil+18] David Silver et al. "A General Reinforcement Learning Algorithm That Masters Chess, Shogi, and Go through Self-Play". In: *Science* 362.6419 (2018), pp. 1140–1144.

[Skr+22] Alexey Skrynnik et al. "POGEMA: Partially Observable Grid Environment for Multiple Agents". In: *arXiv preprint arXiv:2206.10944* (2022). arXiv: 2206.10944.

[SS73] Richard D. Smallwood and Edward J. Sondik. "The Optimal Control of Partially Observable Markov Processes over a Finite Horizon". In: *Operations Research* 21.5 (1973), pp. 1071–1088.

[SS04] Trey Smith and Reid Simmons. "Heuristic Search Value Iteration for POMDPs". In: *Uncertainty in Artificial Intelligence*. 2004, pp. 520–527.

[SS05] Trey Smith and Reid Simmons. "Point-Based POMDP Algorithms: Improved Analysis and Implementation". In: *Uncertainty in Artificial Intelligence*. 2005, pp. 542–549.

[Som+13] Adhiraj Somani et al. "DESPOT: Online POMDP Planning with Regularization". In: *Advances in Neural Information Processing Systems*. Vol. 26. 2013.

[Son78] Edward J. Sondik. "The Optimal Control of Partially Observable Markov Processes over the Infinite Horizon: Discounted Costs". In: *Operations research* 26.2 (1978), pp. 282–304.

[Son71] Edward Jay Sondik. *The Optimal Control of Partially Observable Markov Processes.* Tech. rep. Stanford University, 1971.

[SCD15] Ekhlas Sonu, Yingke Chen, and Prashant Doshi. "Individual Planning in Agent Populations: Exploiting Anonymity and Frame-Action Hypergraphs". In: *International Conference on Automated Planning and Scheduling*. Vol. 25. 2015, pp. 202–210.

[SD12] Ekhlas Sonu and Prashant Doshi. "Generalized and Bounded Policy Iteration for Finitely-Nested Interactive Pomdps: Scaling Up". In: *Autonomous Agents and Multiagent Systems*. IFAAMAS, 2012, pp. 1039–1048.

[SD15] Ekhlas Sonu and Prashant Doshi. "Scalable Solutions of Interactive POMDPs Using Generalized and Bounded Policy Iteration". In: *Autonomous Agents and Multiagent Systems* 29.3 (2015), pp. 455–494.

[SO08] Matthijs TJ Spaan and Frans A. Oliehoek. "The MultiAgent Decision Process Toolbox: Software for Decision-Theoretic Planning in Multiagent Systems". In: *Proc. of the AAMAS Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains (MSDM)*. 2008, pp. 107–121.

[Sto+10] Peter Stone et al. "Ad Hoc Autonomous Agent Teams: Collaboration without Pre-Coordination". In: *AAAI Conference on Artificial Intelligence*. 2010.

[Sua+19] Joseph Suarez et al. "Neural MMO: A Massively Multiagent Game Environment for Training and Evaluating Intelligent Agents". In: *arXiv preprint arXiv:1903.00784* (2019). arXiv: 1903.00784.

[SK18] Zachary Sunberg and Mykel Kochenderfer. "Online Algorithms for POMDPs with Continuous State, Action, and Observation Spaces". In: *International Conference on Automated Planning and Scheduling*. Vol. 28. 2018, pp. 259–263.

[SK22] Zachary Sunberg and Mykel J. Kochenderfer. "Improving Automated Driving Through POMDP Planning With Human Internal States". In: *Transactions on Intelligent Transportation Systems* 23.11 (2022), pp. 20073–20083.

[Sut90] Richard S. Sutton. "Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming". In: *Machine Learning Proceedings 1990*. Elsevier, 1990, pp. 216–224. (Visited on 03/22/2024).

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT press, 2018.

[SCZ05] Daniel Szer, François Charpillet, and Shlomo Zilberstein. "MAA*: A Heuristic Search Algorithm for Solving Decentralized POMDPs". In: *Uncertainty in Artificial Intelligence*. 2005.

[Tai+23] Ayal Taitler et al. "pyRDDLGym: From RDDL to Gym Environments". In: *International Conference on Automated Planning and Scheduling PRL Workshop*. 2023.

[Tam97] Milind Tambe. "Towards Flexible Teamwork". In: *Journal of Artificial Intelligence Research* 7 (1997), pp. 83–124.

[Tan93] Ming Tan. "Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents". In: *International Conference on Machine Learning*. PMLR, 1993, pp. 330–337.

[Tea+21] Open Ended Learning Team et al. "Open-Ended Learning Leads to Generally Capable Agents". In: *arXiv preprint arXiv:2107.12808* (2021). arXiv: 2107.12808.

[Ter+21] Jordan Terry et al. "Pettingzoo: Gym for Multi-Agent Reinforcement Learning". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 15032–15043.

[Tes94] Gerald Tesauro. "TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play". In: *Neural Computation* 6.2 (1994), pp. 215–219.

[TG96] Gerald Tesauro and Gregory Galperin. "On-Line Policy Improvement Using Monte-Carlo Search". In: *Neural Information Processing Systems*. Vol. 9. 1996.

[Tim+22] Finbarr Timbers et al. "Approximate Exploitability: Learning a Best Response". In: *International Joint Conference on Artificial Intelligence*. 2022, pp. 3487–3493.

[Tun+20] Saran Tunyasuvunakool et al. "Dm_control: Software and Tasks for Continuous Control". In: *Software Impacts* 6 (2020), p. 100022.

[TW12] Karl Tuyls and Gerhard Weiss. "Multiagent Learning: Basics, Challenges, and Prospects". In: *AI Magazine* 33.3 (2012), pp. 41–41.

[VD95] Guido Van Rossum and Fred L. Drake Jr. *Python Tutorial*. Vol. 620. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

[Vin+17] Oriol Vinyals et al. "Starcraft Ii: A New Challenge for Reinforcement Learning". In: *arXiv preprint arXiv:1708.04782* (2017). arXiv: 1708.04782.

[Vin+19] Oriol Vinyals et al. "Grandmaster Level in StarCraft II Using Multi-Agent Reinforcement Learning". In: *Nature* 575.7782 (2019), pp. 350–354.

[Wal+02] William E. Walsh et al. "Analyzing Complex Strategic Interactions in Multi-Agent Systems". In: *AAAI Workshop on Game-Theoretic and Decision-Theoretic Agents*. 2002, pp. 109–118.

[WKK18] Erli Wang, Hanna Kurniawati, and Dirk Kroese. "An On-Line Planner for Pomdps with Large Discrete Action Space: A Quantile-Based Approach". In: *International Conference on Automated Planning and Scheduling*. Vol. 28. 2018, pp. 273–277.

[Wan13] Fangju Wang. "An I-POMDP Based Multi-Agent Architecture for Dialogue Tutoring". In: *International Conference on Advanced ICT and Education*. 2013, pp. 486–489.

[Wat89] Christopher J. C. H. Watkins. "Learning from Delayed Rewards". PhD thesis. University of Cambridge, 1989.

[WD92] Christopher J. C. H. Watkins and Peter Dayan. "Q-Learning". In: *Machine Learning* 8.3-4 (1992), pp. 279–292.

[Wei99] Gerhard Weiss. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT press, 1999.

[Wel06] Michael P. Wellman. "Methods for Empirical Game-Theoretic Analysis". In: *AAAI Conference on Artificial Intelligence*. 2006, pp. 1552–1556.

[Wil92] Ronald J. Williams. "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". In: *Machine Learning* 8.3-4 (1992), pp. 229–256.

[WW12] Mark P. Woodward and Robert J. Wood. "Learning from Humans as an I-POMDP". In: *arXiv preprint arXiv:1204.0274* (2012). arXiv: 1204.0274.

[WZC11] Feng Wu, Shlomo Zilberstein, and Xiaoping Chen. "Online Planning for Ad Hoc Autonomous Agent Teams". In: *International Joint Conference on Artificial Intelligence*. 2011.

[Wun+11] Michael Wunder et al. "Using Iterated Reasoning to Predict Opponent Strategies." In: *Autonomous Agents and Multiagent Systems*. IFAAMAS, 2011, pp. 593–600.

[Wun+12] Michael Wunder et al. "A Framework for Modeling Population Strategies by Depth of Reasoning." In: *Autonomous Agents and Multiagent Systems*. IFAAMAS, 2012, pp. 947–954.

[WDM08] Peter R. Wurman, Raffaello D'Andrea, and Mick Mountz. "Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses". In: *AI magazine* 29.1 (2008), pp. 9–9.

[Xin+21] Dong Xing et al. "Learning with Generated Teammates to Achieve Type-Free Ad-Hoc Teamwork." In: *International Joint Conference on Artificial Intelligence*. 2021, pp. 472–478.

[Ye+17] Nan Ye et al. "DESPOT: Online POMDP Planning with Regularization". In: *Journal of Artificial Intelligence Research* 58 (2017), pp. 231–266.

[Yoo+08] Sung Wook Yoon et al. "Probabilistic Planning via Determinization in Hindsight." In: *AAAI Conference on Artificial Intelligence*. AAAI Press, 2008, pp. 1010–1016.

[You+18] Elnaz Shafipour Yourdshahi et al. "Towards Large Scale Ad-Hoc Teamwork". In: *International Conference on Agents*. 2018, pp. 44–49.

[Yu+22] Chao Yu et al. "The Surprising Effectiveness of Ppo in Cooperative Multi-Agent Games". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 24611–24624.

[Yur+20] Ekim Yurtsever et al. "A Survey of Autonomous Driving: Common Practices and Emerging Technologies". In: *IEEE access* 8 (2020), pp. 58443–58469. (Visited on 04/26/2024).

[Zha+19a] Daochen Zha et al. "RLCard: A Toolkit for Reinforcement Learning in Card Games". In: *arXiv preprint arXiv:1910.04376* (2019). arXiv: 1910.04376.

[Zha+19b] Huichu Zhang et al. "Cityflow: A Multi-Agent Reinforcement Learning Environment for Large Scale City Traffic Scenario". In: *The World Wide Web Conference*. 2019, pp. 3620–3624.

[Zha+23] Wayne Xin Zhao et al. *A Survey of Large Language Models*. 2023.

[Zhe+18] Lianmin Zheng et al. "Magent: A Many-Agent Reinforcement Learning Platform for Artificial Collective Intelligence". In: *AAAI Conference on Artificial Intelligence*. Vol. 32. 2018.

[Zil+02] Shlomo Zilberstein et al. "Decision-Theoretic Control of Planetary Rovers". In: *Advances in Plan-Based Control of Robotic Agents*. Ed. by G. Goos et al. Vol. 2466. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 270–289.