Multiagent Stochastic Planning with Bayesian Policy Recognition

by

Alessandro Panella
B.A. (Dipartimento di Elettronica e Informazione, Politecnico di Milano) 2006
M.S. (Department of Computer Science, University of Illinois at Chicago) 2008
M.S. (Dipartimento di Elettronica e Informazione, Politecnico di Milano) 2009

Thesis submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2016

Chicago, Illinois

Defense Committee:
Piotr Gmytrasiewicz, Chair and Advisor
Tanya Berger-Wolf
Barbara Di Eugenio
Ryan Martin, Mathematics, Statistics, and Computer Science
Brian Ziebart

To my family, for their unfailing support.

ACKNOWLEDGMENTS

First and foremost, I would like to thank the person that has been at my side throughout the last year of my PhD program: my fiancée Becky. She has supported me with love, patience, and food. I have no doubt in my mind that without her I could not have achieved this.

A huge heartfelt thank you goes to my amazing family, because I would not be who I am and where I am without them. They have always supported me in all my decisions, and have always been a source of comfort and encouragement. I have the most amazing parents a person could ask for, and I feel unbelievably lucky for that.

I would like to thank my thesis committee, for their help and feedback during the most decisive phase of my PhD program. Special thanks to my academic advisor Prof. Pitor Gmytrasiewicz, for his guidance throughout the years. Additional thanks go to the people at the Computation Institute at the University of Chicago, who gave me the opportunity to expand my research interests.

There are so many great people that directly or indirectly helped me achieve this goal. To all my Italian friends in Chicago, some of which shared with me the joys and pains of the long-term graduate student life, thank you. I would like to give a special acknowledgement to my friend Alessandro Gnoli, once roommate and then everlasting source of peer support. To all my American friends in Chicago, thank you. You have made the last 7 years an amazing time of my life. I will forever be grateful to all of you for being the best friends one could find in a new place. To all my friends form my hometown, thank you. It is always amazing to come

back and have a great time with you all. To all the other people whom I met throughout my years as a student, thank you for helping me build the academic path that culminates in this dissertation.

AP

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

CHAPTER **PAGE**

TABLE OF CONTENTS (Continued)

CHAPTER **PAGE**

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (Continued)

FIGURE **PAGE**

LIST OF ABBREVIATIONS

CRP	Chinese Restaurant Process
δ_D	Dirac delta function
δ_K	Kronecker delta function
DP	Dirichlet Process
FSC	Finite State Controller
FSC	Griffiths-Engen-McCloskey (used to refer to the stick-breaking process)
I-POMDP	Interactive POMDP
LHS	Left-Hand Side
MCMC	Markov Chain Monte Carlo
MCTS	Monte Carlo Tree Search
MH	Metropolis-Hastings
PDFC	Probabilistic Deterministic Finite Controller
POMCP	Partially Observable Monte Carlo Planning
POMDP	Partially Observable Markov Decision Process
RHS	Right-Hand Side

SUMMARY

We consider an autonomous agent facing a stochastic, partially observable, multiagent environment. In order to compute an optimal plan, the agent must accurately predict the actions of the other agents, since they influence the state of the environment and ultimately the agent's utility. To do so, we propose a special case of interactive partially observable Markov decision process (I-POMDP), in which the agent does not explicitly model the other agents' beliefs and intentions, and instead models the other agents as stochastic processes implemented by probabilistic deterministic finite state controllers (PDFCs).

The agent maintains a probability distribution over the PDFC models of the other agents, and updates this distribution using Bayesian inference. Since the number of nodes of these PDFCs is unknown and unbounded, the agent places a Bayesian nonparametric prior distribution over the infinitely dimensional set of PDFCs. This allows the size of the learned models to adapt to the complexity of the observed behavior. Deriving the posterior distribution is in this case too complex to be amenable to analytical computation; therefore, we provide a Markov chain Monte Carlo (MCMC) algorithm that approximates the posterior beliefs over the other agents PDFCs, given a sequence of (possibly imperfect) observations about their behavior. Experimental results show how the learned models converge behaviorally to the true ones.

Moreover, we describe how the learned PDFCs can be embedded in the learning agent's own decision making process. We consider two settings, one in which the agent first learns, then interacts with other agents, and one in which learning and planning are interleaved. We show

how the agent's performance increases as a result of learning in both situations. Moreover, we analyze the dynamics that ensue when two agents are simultaneously learning about each other while interacting, showing in an example environment that coordination emerges naturally from our approach. Moreover, we demonstrate how an agent can exploit the learned models to complement its possibly noisy observations about the environment.

CHAPTER 1

INTRODUCTION

In artificial intelligence, an agent is defined as an autonomous entity that interacts with an environment by performing actions and receiving observations, and is characterized by an *agent function*, a mapping from the agent's history of observations to actions (1). A rational agent aims at performing a course of actions that maximizes a given optimality criterion. The work presented in this thesis considers an agent facing an environment that is:

- **sequential:** the environment evolves as a discrete sequence of configurations;
- **stochastic:** when the agent performs an action, the environment goes from one configuration to the next according to some probability distribution; and
- **partially observable:** the agent only receives partial observations stochastically related to the current configuration.

In this kind of environment, a *rational* agent maximizes its *expected utility*, where the expectation is taken with respect to its *belief*, defined as the agent's subjective probability distribution over the current state of the environment. This concept of rationality prescribes that at each step a rational agent updates its belief according to Bayesian inference, on the basis of newly collected evidence. Finding the strategy that maximizes the expected utility in this scenario is known as *partially observable stochastic planning*, and has a mathematical formulation given by the partially observable Markov decision process (POMDP) (2). Computing the optimal

solution to this problem is computationally very intensive, due to the uncertainty both in the environment transitions and in the agent's observations.

Yet an additional layer of complexity is introduced when there are other agents that also influence the environment with their own actions. We say in this case that the environment is *multiagent*. Traditionally, problems in which multiple agents interact within an environment have been approached using tools from economics and game-theory, that are mostly based one the concept of Nash equilibrium.

In this work, we depart from equilibrium solution concepts, and build upon the body of work that uses subjective game theory and decision theory as its core paradigms (3; 4; 5). In particular, we propose an approach to multiagent stochastic planning that continues on the path opened by the introduction of the interactive partially observable Markov decision process (I-POMDP) (6). Within this framework, an agent maintains a belief not only on the state of the environment, but also on the models of other agents, and its decisions depend solely on this subjective belief.

In I-POMDP literature, it is predominant to consider *intentional* models (7; 8) of other agents, which consist in ascribing to them beliefs and preferences, and simulating their decision making process in order to predict their actions. This intentional stance is reciprocated by the other agents, yielding an infinite hierarchy of nested beliefs, which is cut off at a finite level for implementation purposes.

In this work, we propose the use of *subintentional* models to represent the other agents' behavior. A subintentional model is intended here as a stochastic process over the modeled

agent's sequence of actions, that depends on the observations it receives as input. Note that a subintentional model does not explicitly take into account another agent's preferences or beliefs.

Our interest in subintentional models has several motivations. The first is computational: since subintentional models do not explicitly represent the other agents' beliefs, there is no explicit reciprocal modeling. This means that an agent does not have to recursively solve the other agents' models in order to enable its own optimal decision making. This may lead to substantial computational savings. Moreover, we are interested in situations where an agent has weak prior knowledge about others. In particular, we consider the case in which their preferences are completely unknown to the modeling agent. In this scenario, the agent's predictions of the other agents' actions can be solely based on recognizing regular patterns in their behavior, which is what subintentional models directly attempt to do. Furthermore, subintentional modeling makes no assumption about the rationality of the other agents. On the other hand, intentional modeling relies on the fact that other agents are rational, since their actions are predicted by assuming that they maximize their expected utility. While this may usually be a safe assumption, it is not universally true. For example, an agent's rationality may be bounded or compromised.

Among all possible subintentional models, we consider the probabilistic deterministic finite controller (PDFCs) representation, in which the transitions between the nodes are deterministic, while actions are generated stochastically in each node. Intuitively, the nodes of a PDFC represent discrete "mental states" of the modeled agents. However, the other agent's beliefs are never represented explicitly.

In line with the decision-theoretic approach to planning discussed above, the modeling agent places a prior distribution over the space of PDFCs, and updates this belief in the Bayesian way, based on the observed behavior of other agents. We do not assume that the agent is given a restricted set of possible PDFCs of the other agents a priori. Instead, every possible PDFC must be considered. This implies that the size of the PDFC, albeit finite, is unbounded. This is desirable, since we do not wish to bound, a priori, the complexity of other agents' models. However, constructing a probability distribution over PDFCs of unbounded size presents some challenges. In this work, we tackle this problem by using a *nonparametric Bayesian* prior distribution (9). Note that the term "nonparametric" does not indicate in this case the absence of parameters, but quite the opposite situation in which the posterior number of parameters (here, the size of the PDFC) scales with the complexity of the observed data (here, another agent's behavior.)

1.1 Summary of Contributions and Results

The goal of this thesis is to provide a multiagent planning methodology that specializes the interactive POMDP framework by considering subintentional models of other agents, which are represented as probabilistic deterministic finite controllers (PDFC). This is achieved by means of the following contributions:

- We design a suitable nonparametric prior probability distribution for the infinite-dimensional space of PDFCs, based on the Dirichlet process. We motivate the use of this prior distribution and characterize it by defining the probability it induces over the number of nodes of the PDFCs.

- Performing Bayesian inference using nonparametric distributions is not amenable to analytical solutions. We develop an ad-hoc Markov chain Monte Carlo algorithm (MCMC) that learns a sample-based approximation of the posterior distribution from a sequence of observed behavior. In order to allow a more efficient exploration of the state space, the algorithm occasionally splits and merges whole PDFC nodes, along with performing Gibbs moves on individual transitions. The modeling agent's observations may not reveal completely the behavior of the modeled agents; therefore, the inference procedure must deal with partial observability. We show, for three experimental domains, that the posterior distribution concentrates around PDFCs that are progressively more similar to the true models generating the behavior as longer training trajectories are used. While learning the true model of another agent exactly with probability one is only possible with an infinite amount of observations, our method is able to pick up regularities in the modeled agent's behavior even from limited observations, and the prior distribution naturally compensates for data scarcity in the Bayesian way.
- We formalize the framework of subintentional I-POMDP with PDFC models, yielding a mechanism that allows the agent to exploit the learned PDFCs during interaction. We present two modalities of use of this framework.
  - *Learn, then plan.* The modeling agent first accumulates observations about other agents' behavior, performs inference about their models offline, and then exploits such models during interaction. Our results demonstrate that, even though learning the exact model of another agent is unattainable, especially with realistic observabil-

ity assumptions, the agent can still largely improve its performance by recognizing behavioral patterns that are statistically significant and encode them in a compact model. This allows the agent's performance to greatly improve even when the true model is not actually discovered.

– *Interleaved learning and planning.* The agent periodically updates the models of other agents during interaction. In this context, we show experimentally that our methodology is robust with respect to the actual process that generates the other agents' behavior, even when their true model is not implemented as a finite-state policy. Moreover, we analyze the dynamics that ensue when two agents simultaneously learn about each other in a simple but meaningful interactive domain. We show that coordination between two agents can emerge naturally from our methodology without requiring any prior knowledge of the agents' about each others' preferences. Furthermore, we provide an example of how an agent can exploit the learned models to augment its own observation capabilities, enabling it to achieve higher payoffs as a result.

1.2 Structure of the Thesis

The rest of this thesis is organized as follows:

- In Chapter 2 we provide the necessary background to the presented work and discuss some of the related work in multiagent learning.

- Chapter 3 describes the prior distribution over PDFCs that constitutes the basis for our work, and provides its characterization in terms of induced number of nodes.
- Chapter 4 presents the MCMC inference algorithm that performs approximate Bayesian inference on the space of PDFCs from perfectly observed behavior trajectories, followed by the description and discussion of experimental results.
- In Chapter 5, we describe how the MCMC algorithm can be expanded to deal with partially observed behavior, and validate its effectiveness.
- In Chapter 6, we propose a two-phase methodology in which an agent first learn about other agents, and then interacts with them. We show that, as a result of learning, the agent increases its rewards.
- Chapter 7 provides an approach for interleaving learning and planning, and provides experimental results for an example domain, by analyzing different types of interactions between two agents.
- Chapter 8 concludes the thesis and describes possible future work.

CHAPTER 2

BACKGROUND

2.1 Partially Observable Markov Decision Processes

A general framework for (single-agent) stochastic planning under partial observability is the partially observable Markov decision process (POMDP) (2). Formally, a POMDP is a 6-tuple

$$P = (S, A, \Omega, T, O, R), \tag{2.1}$$

where:

- S is the set of the possible **states of the world**.
- A is the set of the agent's possible **actions**.
- Ω is the set of the agent's possible **observations**,
- $T : S \times A \to \Delta(S)$ is the environment's **transition function** (or model), that maps each state-action pair to a distribution over the next state of the environment[1]. If the environment is in state s and the agent executes action a, the next state of the environment s' is drawn with probability $p(s'|s, a) = T(s, a, a')$.
- $O : A \times S \to \Delta(\Omega)$ is the agent's **observation function** (or model), that maps each action-state pair to a probability distribution over the set observation received by the

[1]The symbol 'Δ' denotes the probability simplex over a set

agent. If the agent executed action a resulting in state s', then the observation to the agent is drawn with probability $p(\omega|a, s') = O(a, s', \omega)$.

- $R(s, a) \to \mathbb{R}$ is the agent's **reward function** (or model), that maps each state-action pair to a real number representing the reward awarded to the agent.

A solution to a POMDP is a function that maps any history of observations and actions to distribution over the agent's actions. Formally, we define a history, or trajectory, of length t as $h_{1:t} = (a_{1:t-1}, \omega_{2:t})$.[1] We denote as $H_{1:t}$ the set of all possible histories of length t. A solution to a POMDP is therefore a function $f : H_{1:t} \to \Delta(A)$, for any relevant t, that optimizes a given optimality criterion.

The optimality criterion depends on the planning horizon, that is, the length of interaction with the environment considered by the agent. If the horizon is finite, say T, then the agent can aim at optimizing the total sum of expected rewards received during that time, and the agent's policy will only have to consider histories of length at most T. If the horizon is infinite, indicating that the agent assumes that the interaction has an unbounded duration, then we need to carefully design on optimality criterion. The usual criterion in this case is the *expected sum of discounted rewards*, defined as:

$$E\left[\sum_{t=1}^{\infty} \gamma^t \, R_t\right], \tag{2.2}$$

---

[1]The notation $x_{1:t}$ indicates the sequence $(x_1, x_2, \ldots, x_t)$.

Where R_t is the random variable representing the reward obtained at time t, and $0 < \gamma < 1$ is the **discount factor**, that indicates how future rewards are discounted, so that the sum above converges. For infinite-horizon POMDP, the discount factor is usually included in the tuple of Equation 2.1.

Instead of explicitly considering the infinite set of possible histories, a POMDP agent usually summarizes this information into a *belief state* $b \in \Delta(S)$, a probability distribution over states of the world that is a sufficient statistics of the past history of the agent, i.e. $b(s)$ is the probability that the agent assigns to the world being in state s. Since it is a sufficient statistics, it is possible to obtain a closed formula for the "updated" belief b' upon executing an action a and receiving an observation ω, starting from belief b. The belief update formula can be derived via application of Bayes' rule and is:

$$b'(s') = p(s'|\omega, a, b) = \beta \; O(s', a, \omega) \sum_{s \in S} b(s) \; T(s, a, s'), \tag{2.3}$$

where $\beta^{-1} = p(\omega|a, b)$ is the Bayes normalization factor, that is constant with respect to s'.

2.1.1 Solving POMDPs

We can view a policy for a POMDP agent as a conditional plan implemented by a policy tree π: from a starting node, the agent executes a prescribed action[1] and receives an observation.

---

[1]Note that here we consider deterministic policies. The discussion is still valid in the case of stochastic policies with minimal variations.

This makes the agent transition to a child of the original node, and this process is repeated at each timestep. The value of a given policy tree π in a state s is given by the recursive equation:

$$V_\pi(s) = R(s, a(\pi)) + \gamma \sum_{s' \in S} T(s, a(\pi), s') \sum_{\omega \in \Omega} O(a, s', \omega) V_{\pi.\omega}(s'), \tag{2.4}$$

where $a(\pi)$ indicates the action prescribed at the root of policy tree pi, and $\pi.\omega$ is the subtree reached when traversing the edge labeled by ω. We call V_π the **value function** induced by π. Since the equation above is linear, the value of following a plan given a belief b is:

$$V_\pi(b) = \sum_{s \in S} b(s) V_\pi(s) = b \cdot V_\pi, \tag{2.5}$$

using vector notation. Given Π_t, the set of all the policy trees of depth t, the optimal plan π^* for belief b is then $\pi^* = \text{argmax}_{\pi \in \Pi_t} V_\pi$. From this equation we can derive the insight that a policy tree π corresponds to a value function that is linear in b, and that the optimal value function V_t^* corresponds to the piece-wise linear convex surface of the collection of value functions induced by policy trees in Π_t. A qualitative example is depicted in Figure 1, for a POMDP with two states.

From what is described above, it follows that the optimal infinite-horizon value function is the solution to the set of recursive relations:

$$V^*(s) = \max_{a \in A} \left[R(s, a(\pi)) + \gamma \sum_{s' \in S} T(s, a(\pi), s') \sum_{\omega \in \Omega} O(a, s', \omega) V^*(s') \right], \tag{2.6}$$

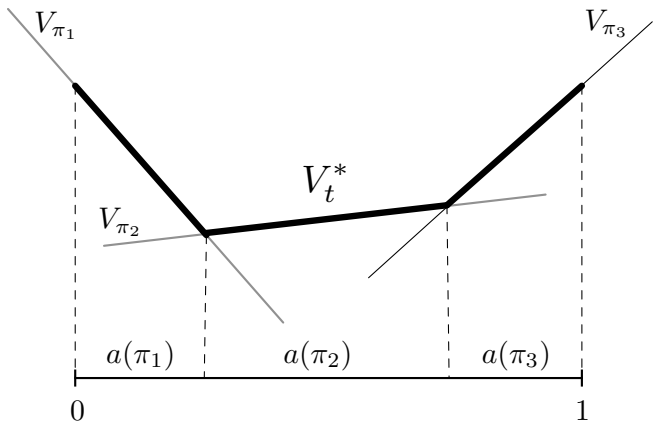

Figure 1. POMDP value function. Adapted from (2).

for all $s \in S$, which usually referred to as the **Bellman equation**.

Value iteration is a dynamic programming algorithm and the earliest known method for solving POMDPs exactly (10). It applies the Bellman equation recursively until convergence to an ϵ-optimal value function, while pruning completely dominated strategies. In some cases, the number of optimal non-dominated plans stops growing long before reaching convergence and the resulting value function is composed by a bounded number of linear segments; these policies are called *finitely transient* (2). This corresponds to the situation in which the belief regions induced by the optimal value function (see Figure 1) are mapped into each other exactly by the belief update formula. In this case the optimal policy can be represented as a finite state controller (see next section.)

Since its publication in (10), many different versions of value iteration have been proposed. In (2), the authors provide a more efficient form of value iteration called the "witness algorithm."

However, the problem of solving a POMDP exactly has been proven to be PSPACE-complete (11). Therefore, some versions of value iteration give up optimality in favor of convergence speed. One of the most prominent class of such approximations is the so-called **point-based value iteration** (12; 13), in which only a finite set of reachable beliefs is considered instead of the whole belief simplex. Other methods include Monte Carlo approximation (14) and factored representations of the state space (15; 16).

An alternative class of algorithms is **policy iteration** (17; 18). While also employing dynamic programming, this algorithms work directly on the space of finite state controllers in order to derive high-quality policies.

The types of POMDP algorithms mentioned above aim at solving a problem offline, once and for all. On the other hand, in **online POMDP algorithms** (19), an agent starts from a given belief state, and uses a forward search procedure in the belief space in order to select its current optimal action; at each step, this process is repeated. Recently, a sample-based online POMDP solver was proposed in (20), that uses a Monte-Carlo tree search (MCTS) (21) procedure for implementing the forward search, instead of exploring the full-width search tree. Such algorithm, called partially observable Monte Carlo planning (POMCP), is of primary importance in this work and is described more in detail in the next section.

2.1.2 Partially Observable Monte Carlo Planning

Traditional online POMDP planners work by executing a "full-width" search of the forward belief tree. This means that the agent considers every possible belief state that can be reached from the current belief. This procedure has a complexity exponential in the time horizon. In

contrast, POMCP is an online randomized algorithm that performs a finite set of randomized simulations through the belief search tree instead of executing a full-width search. The algorithm has two main features:

- Instead of running a set of random simulations, the algorithm uses Monte Carlo tree search (MCTS) to orient the search towards more promising regions of the belief search tree. This is done by considering the choice of action at each hypothetical belief node as a multi-armed bandit problem, and applying the UCB1 algorithm (22) in order to select the next action in the simulation. This provides an optimal trade-off between exploring new future belief paths and focusing the search on branches that seem promising.
- Each belief node is represented empirically as a set of unweighted particles, each corresponding to a possible state of the world. During the execution of MCTS to select the next action, the sampled future states of the world are stored at each belief node along the simulation. Once the agent executes a real action and receives an observation, its updated belief node is obtained by following the corresponding branch in the lookahead belief tree created by MCTS: the set of particles that were stored in such node during the simulations constitute the new belief of the agent.

Due to its high performance, POMCP has been gathering growing interest in the since its inception, and its use has become widespread in the POMDP community.

2.2 Probabilistic Deterministic Finite-state Controllers

As mentioned in the Section 2.1.1, finite-state controllers (FSCs) represent in some cases the optimal solution to POMDP; indeed, some algorithms search directly in the space of finite state

controllers in order to find a good policy. Therefore, our choice of modeling an observed agent's behavior as a finite state controller is perhaps not surprising. As we will see, finite controllers are suitable to capture regularities in an observed agent's behavior even when the latter is not prescribed by an actual finite controller. In other words, finite state controllers are a powerful statistical model that yields to effective learning of input/output stochastic processes.

In this work, we consider a special case of FSCs, called probabilistic deterministic finite-state controllers (PDFCs). A PDFC is a 6-tuple:

$$C = (\Omega, A, Q, \tau, \theta, \tau_0), \tag{2.7}$$

where:

- Ω is the set of observations of the agent.
- A is the set of actions the agent can execute.
- Q is the set of nodes.
- $\tau : Q \times A \times Q \to Q$ is the deterministic node transition function. If q is the current node, from which an agent executes an action a and receives an observation ω, the next node is $q' = \tau(q, a, \omega)$, that we usually abbreviate as $\tau_{qa\omega}$.
- $\theta : Q \to \Delta(A)$ is the stochastic emission (action generation) function. If q is the current node, the agent will execute action a with probability $p(a|q) = \theta(q, a)$, sometimes abbreviated as θ_{qa}; similarly, θ_q denotes the probability vector $(\theta_{q1}, \theta_{q2}, \ldots, \theta_{q|A_j|})$.
- τ_0 is the starting node.

It is straightforward to see how a PDFC can implement a POMDP agent function: conditional on the observed history $h_{1:T} = (a_{1:T-1}, \omega_{2:T})$, the current PDFC node is defined recursively for $t = 1, 2, \ldots$, starting from q_1, as $q_t = \tau(q_{t-1}, a_{t-1}, \omega_t)$, and will generate an action drawing from the probability distribution $\theta(q_t, \cdot)$. Intuitively, the nodes of a PDFC implementing a conditional policy can be viewed as internal mental states of the agent that behaves according to such policy. In POMDPs, this is indeed the case, since each node of a PDFC (and in general, of a FSC) can be mapped to a region of the belief space.

Given a sequence of observations $\omega_{2:T}$, a PDFC induces a probability distribution over the action sequence $a_{1:T}$ defined as follows:

$$\begin{aligned} & q_1 = \tau_0 \\ & p(a_{1:T}|\omega_{2:T}) = \theta(q_1, a_1) \prod_{t=2:T} \sum_{q \in Q} \delta_K(q, \tau(q_{t-1}, a_{t-1}, \omega_t))\, \theta(q, a_t). \end{aligned} \tag{2.8}$$

Here and in the remainder of this thesis, δ_K denotes the Kronecker delta function[1].

PDFCs are related to transducers in the context of grammar learning (23). In (24), the authors propose a heuristic algorithm for learning deterministic transducers based on state-merging moves. More recent work (25) attempts at learning stochastic transducers using spectral methods.

---

[1]The Kronecker delta function is equal to 1 if its arguments are equal, 0 otherwise.

2.3 Interactive POMDPs

The interactive POMDP (I-POMDP) (6) is an extension of the single-agent POMDP framework to multiagent settings. It is important to notice that in the I-POMDP framework each agent maintains full autonomy, and performs its planning in isolation and without any sort of centralized control. In the following and in the rest of this thesis, we consider an environment containing two agents, namely agent i and agent j. All the descriptions and methods easily generalize to environments with more than two agents. In its most general form (26), an I-POMDP for agent i is defined as a 6-tuple:

$$I - POMDP_i = (IS_i, A, \Omega_i, T_i, O_i, R_i), \tag{2.9}$$

where:

- IS_i is the set of **interactive states**, defined as $IS_i = S \times M_j$, where M_j is the set of possible models of the other agent. Each model $m_j \in M_j$ is a triple $m_j = (O_j, h_j, f_j)$, where $O_j \in \mathcal{O}_j$ is an observation function[1], $h_j \in H_j$ is a history, and $f_j \in F_j$ is an agent function of the form $f_j : H_j \to \Delta(A_j)$.
- $A = A_i \times A_j$ is the set of joint actions for the agents.
- Ω_i is the set of observations for agent i.

---

[1] O_j here also implicitly contains information about agent j's observation set Ω_j.

- $T_i : S \times A \to \Delta(S)$ is the stochastic transition function. $T_i(s, a_i, a_j, s') = p(s'|s, a_i, a_j)$ is the probability of landing in state S given that the agents execute actions (a_i, a_j) when the environment is in state s.
- $O_i : S \times A \to \Delta(\Omega_i)$ is the stochastic observation function. $O_i(s, a_i, a_j, \omega)$ corresponds to the probability that agent i receives observation o, given that the joint action (a_i, a_j) was executed and the resulting state is s.
- $R_i : IS_i \times A \to \mathbb{R}$ is the reward function. $R_i(is, a_i, a_j)$ is the payoff to agent i when joint action (a_i, a_j) is executed in interactive state is. In line with I-POMDP literature, we restrict the model by assuming that $R_i : S \times A \to \mathbb{R}$, that is, the reward depends only on the physical state of the world, and the agent does not have preferences over the models of the opponent.

Similarly to single-agent POMDPs, any history can be summarized with a belief over the interactive state space. However, the belief update equation is more complicated. First, the agent, say i, needs to infer what action agent j has executed, in that it influences the state of the environment and agent i's own observation. Second, agent i needs to speculate over j's observation in order to update its belief about j's model. We assume in the following that the

model m_j of the other agent is stationary, except obviously for the observation history, which grows as time passes. The belief update is defined as:

$$b'(is') = p(is'|a_i, \omega_i, b) = \\ \beta \sum_{is \in IS \,:\, (O'_j, f'_j) = (O_j, f_j)} b(is) \times \left[\sum_{a_j \in A_j} f_j(h_j, a_j)\; O_i(a_i, a_j, s', \omega_i)\; p(is'|is, a_i, a_j) \right]. \tag{2.10}$$

The quantity $p(is'|is, a_i, a_j)$ is the *interactive transition model* and is defined as:

$$p(is'|is, a_i, a_j) = T_i(s, a_i, a_j, s') \sum_{\omega_j \in \Omega_j} O_j(s, a_i, a_j, \omega_j)\; \delta_K(h'_j, \text{APPEND}(h_j, (a_j, \omega_j))), \tag{2.11}$$

where APPEND is a function that returns a sequence resulting from the concatenation of its second argument to the first.

The formalization above is mathematically sound, but of little practical use if we do not restrict the set of possible agent functions F_j. Simply considering every possible agent function, even limited to the Turing-computable ones, is not viable. Beyond the apparent impracticality of such approach, there are actual theoretical limits, as described in (26). In fact, instead of every possible agent function, the original definition of I-POMDP introduced in (6) considered the–still very broad–set of intentional models, and is described in the next section.

2.3.1 Intentional I-POMDPs

By intentional I-POMDPs, we consider the framework in which the set of policies F_j of the agent j is itself implicitly specified as an I-POMDP. This means that an I-POMDP agent i of this kind models another agent j in terms of beliefs and intentions, i.e. it takes an "intentional

stance" in modeling the other agent. This includes the possibility that agent j models agent i back, and so on, generating an infinite hierarchy of nested beliefs. Because of this, in intentional I-POMDPs only a finite number of nesting levels is taken into account.

Formally, a finitely nested I-POMDP is described with a 6-tuple similar to Equation 2.9, with the addition of an index indicating the nesting level, i.e.:

$$I - POMDP_{i,l} = (IS_{i,l}, A, \Omega_i, T_i, O_i, R_i). \tag{2.12}$$

All the elements are the same as before, with the exception of the interactive state space, which is now more specific and defined as $IS_{i,l} = S \times M_{j,l-1}$. Here, $M_{j,l-1} = \{\Theta_{j,l-1} \cup SM_j\}$ for $l \geq 1$, and $M_{j,l-1} = \varnothing$ for $l = 0$. $\Theta_{j,l-1}$ is the set of computable *intentional models* of agent j, while SM_j is the set of *subintentional models*. An intentional model is a pair $\theta_{j,l-1} = \langle b_{j,l-1}, \hat{\theta}_j \rangle$, where $b_{j,l-1}$ is a belief over $IS_{j,l-1}$, and $\hat{\theta}_j = \langle A_i, T_j, \Omega_j, O_j, R_j \rangle$ is a *frame* of agent j, which is assumed to be rational. A subintentional model sm_j can be as simple as a probability distribution over j's action or a more complex nondeterministic finite state machine. The interactive state space with l levels of nesting can be recursively constructed as:

$IS_{i,0}$	$= S,$	$\Theta_{j,0}$	$= \{\langle b_{j,0}, \hat{\theta}_j \rangle \mid b_{j,0} \in \Delta(IS_{j,0})\},$
		$M_{j,0}$	$= \Theta_{j,0} \cup SM_j$
$IS_{i,1}$	$= S \times M_{j,0},$	$\Theta_{j,1}$	$= \{\langle b_{j,1}, \hat{\theta}_j \rangle \mid b_{j,1} \in \Delta(IS_{j,1})\},$
		$M_{j,1}$	$= \Theta_{j,1} \cup SM_j$
	$\vdots$		$\vdots$
$IS_{i,l}$	$= S \times M_{j,l-1},$	$\Theta_{j,l}$	$= \{\langle b_{j,l}, \hat{\theta}_j \rangle \mid b_{j,l} \in \Delta(IS_{j,l})\}$

The I-POMDP belief update defined in Equation 2.10 can be specialized to this case. Although we do not provide all the details here, it is evident that in order to compute the probability over agent j's action, agent i must simulate j's own decision making process, that in turn might require to solve j's model of i's decision making process, and so on down to level 0. Therefore, it is clear that solving an intentional I-POMDP at any level of nesting is at least as complex as solving a single-agent POMDP. On top of this computational hardness, there is a more theoretical limitation. While we may wish to maintain a belief over every possible belief at the subsequent level, it can be shown (27) that it is impossible to have a probability distribution over every possible belief for nesting level larger than two.

Nevertheless, several approaches have been proposed that alleviate these difficulties, usually by compromising for an approximate solution. Some of these methods mirror advances proposed for single-agent POMDP algorithms, although their generalization is far from straightforward. Among these, we find Monte Carlo methods (27), policy iterations algorithms (28), point-based value iteration (29), and nested dynamic influence diagrams (30).

2.4 Bayesian Inference and Nonparametric Priors

2.4.1 Bayesian Learning

Bayesian inference is a powerful paradigm that allows to "learn" hypotheses from data by transforming a previously held belief over the hypothesis space, the prior, into a posterior belief filtered by the data that are observed. The idea is that the hypotheses that better "explain" the data will have a higher posterior probability (of being true) than the ones that do not appear to support the observed data. Let us denote our hypothesis space as Θ, usually called

the "parameter space" because $\theta \in \Theta$ parameterizes a stochastic generative model $p(X|\theta)$, for some observed variable X. Assume that we have a previously held belief over the values of θ, the **prior** probability $p(\theta)$. After observing some realizations of the generative process, $x_{1:N} = (x_1, x_2, \ldots, x_N)$, the **evidence**, we can update our belief over the hypothesis space using Bayes' rule, as:

$$p(\theta|x) = \frac{p(x|\theta)\ p(\theta)}{p(x)}. \tag{2.13}$$

Above, the term $p(\theta|x)$ is the **posterior** distribution, that represents our updated belief *after* observing the evidence, while $p(x|\theta)$ is the **likelihood**, that is a measure of how likely it is that we are observing the evidence, given that the parameter of the generative process is θ. The denominator $p(x) = \int_\Theta p(x|\theta)p(\mathrm{d}\theta)$ represents the probability of observing the evidence given the prior belief, and serves as a normalization factor since it is independent from the value of θ on the left-hand side.

Note that in general, each data point x_i is a vector and not just a scalar. Similarly, the parameter space Θ is in general multi-dimensional, e.g. $\Theta = \mathbb{R}^D$ for some $D \in \mathbb{N}^+$. Despite making the computation more involved, this does not pose any difficulty. Moreover, the parameter space can be infinite-dimensional, as we see in Section 2.4.3.

2.4.2 Markov Chain Monte Carlo Inference

It is often the case that the posterior distribution in Equation 2.13 cannot be computed analytically, and approximate numerical solutions must be adopted. Markov chain Monte Carlo (MCMC) (31) is very popular class of methods that allow to draw a set of samples from the posterior distribution, without actually having to compute it analytically. This is done by

constructing an aperiodic, irreducible Markov chain over the parameter space Θ, for which the stationary distribution is the posterior distribution $p(\theta|x)$. MCMC methods differ on how to build the *jumping probability* of going from a state $\theta^{(n)}$ of the chain the the next state $\theta^{(n+1)}$.

Every MCMC algorithm starts by drawing an initial value of θ such that $p(\theta|x) > 0$, and then generates a sequence of states using the jumping probability. At some point, referred to as the *mixing time*, this chain will reach its stationary distribution. The samples generated form there on will be as if drawn from the stationary distribution of the chain, that is, the target posterior distribution. In the following two sections, we briefly describe two of the most used MCMC methods: Metropolis-Hastings and Gibbs sampling.

2.4.2.1 Metropolis-Hastings

A method for constructing a suitable Markov chain is the Metropolis-Hastings (MH) algorithm (32; 33). The method iterates through the following steps:

1. Sample an initial value $\theta^{(1)}$ from some distribution g, such that $p(\theta^{(1)}|x) > 0$. Set $n = 1$.
2. Sample a candidate value θ^* from a *proposal distribution* $q(\cdot|\theta^{(n)})$.
3. Compute the MH acceptance ratio:

$$a = \min\left[1, \frac{q(\theta^{(n)}|\theta^*)}{q(\theta^*|\theta^{(n)})}\frac{p(\theta^*|x)}{p(\theta^{(n)}|x)}\right] = \min\left[1, \frac{q(\theta^{(n)}|\theta^*)}{q(\theta^*|\theta^{(n)})}\frac{p(x|\theta^*)\ p(\theta^*)}{p(x|\theta^{(n)})\ p(\theta^{(n)})}\right]. \tag{2.14}$$

4. Accept the proposed value θ^* as the next value $\theta^{(n+1)}$ with probability a.
5. Set $n = n + 1$ and repeat from steps 2-5 until convergence.

The method requires the choice of a *proposal distribution* $q(\cdot|\theta)$. Such distribution can be arbitrary, as far as it generates an aperiodic and irreducible chain. However, different choices can greatly influence the mixing time and the convergence of the chain.

2.4.2.2 Gibbs Sampling

Gibbs sampling (34) is another method that generates a Markov chain over the parameter space whose stationary distribution is the target posterior, and is designed to cope with multi-dimensional parameter spaces. Suppose that θ is a d-dimensional vector $\theta = (\theta_1, \theta_2, \ldots, \theta_d)$. While we might not be able to sample directly from the multivariate posterior $p(\theta|x)$, it is often the case that the univariate conditional distributions $p(\theta_i|\theta_{-i}, x)$ are easier to sample from. The notation θ_{-i} indicates all values of θ except θ_i, i.e. $\theta_{-i} = (\theta_1, \ldots, \theta_{i-1}, \theta_{i+1}, \ldots, \theta_d)$. By exploiting this fact, the Gibbs sampler builds a chain by iteratively sampling from the univariate conditional distributions, and is composed by the following steps:

1. Sample an initial value $\theta^{(1)}$ from some distribution g, such that $p(\theta^{(1)}|x) > 0$. Set $n = 1$.
2. For $i = 1, 2, \ldots, d$, sample the next values $\theta_i^{(n+1)} \sim p(\theta \mid \theta_1^{(n+1)}, \ldots, \theta_{i-1}^{(n+1)}, \theta_{i+1}^{(n)}, \ldots, \theta_d^{(n)})$.
3. Set $n = n + 1$ and repeat steps 2-3 until convergence.

There exist many variations of the basic Gibbs sampling schema described above. A simple variation is to consider a random ordering of the individual parameters instead of a pre-defined one. Another modification consists in sampling groups of variables at a time instead of just one, whenever this is possible. This is known as *blocking*. Yet another modification is to include a MH step within Gibbs in order to sample the value of one or more variables, resulting in a

hybrid MCMC sampler. Sometimes, one or more variables can be integrated out analytically. When this is done, the resulting algorithm is referred to as *collapsed* Gibbs sampler.

2.4.3 Dirichlet Process, Stick-Breaking Process, Chinese Restaurant Process

The fact that we can consider an infinite-dimensional space is somewhat surprising: how can we perform inference over an infinite amount of parameters if we will observe a finite amount of observations? The catch is that, usually, infinite-dimensional parameter spaces are used to describe spaces that are actually finite-dimensional, but whose dimensionality is unknown and unbounded. While there are several ways to design a prior distribution $p(\theta)$ over an infinite-dimensional space, we focus here on the class of Bayesian nonparametric (BNP) priors, and in particular on distributions based on the Dirichlet process (DP).

A Dirichlet process (35; 9) is a probability distribution over measures, characterized by a *mean probability distribution* G_0 over a parameter space Θ, and a concentration parameter α. We denote this as $\mathrm{DP}(H, \alpha)$. The following hierarchical construction yields a distribution over an infinite-dimensional space that has some nice mathematical properties:

$$\begin{aligned} G &\sim \mathrm{DP}(H, \alpha) \\ \theta_k \mid G &\sim G \qquad\qquad k = 1, 2, \ldots \end{aligned} \tag{2.15}$$

A draw G from $\mathrm{DP}(H, \alpha)$ is an infinite discrete distribution over Θ that has the following form (36):

$$G(\theta) = \sum_{k=1}^{\infty} \pi_k \delta_K(\theta, \theta_k), \tag{2.16}$$

where:

- For each $k = 1, 2, \ldots \infty$, θ_k is drawn according to the mean measure of the process, i.e. $\theta_k \sim H$.
- The infinite-dimensional *weight vector* π is drawn from the stick-breaking distribution denoted as GEM(α)[1], i.e. $\pi \sim \text{GEM}(\alpha)$. This means that each component $\pi_1, \pi_2, \ldots$ is drawn recursively as:

$$\pi_k = \mu_k \prod_{h=1}^{k-1} (1 - \mu_h), \qquad \text{where } \mu_k \sim \text{Beta}(1, \alpha). \tag{2.17}$$

A draw from GEM(α) can be thought as being generated by the following intuitive procedure, that gives it the name "stick-breaking process": imagine to break a stick of length 1 at a random point π_1 drawn according to Beta$(1, \alpha)$. We then take the remaining part (of length $1 - \pi_1$) and break it using the same procedure, and so on infinite times.

We can motivate the terms "mean (or base) measure" for H and "concentration parameter" for α. When $\alpha \to 0$, the entire stick is broken off and allocated to π_1, resulting in a degenerate draw G corresponding to just one θ_1 being drawn from the base measure H. On the other hand, when $\alpha \to \infty$, we break the stick into infinite infinitesimally small segments, each associated to a $\theta_k \sim H$. The discrete distribution G is then an approximation of the (in general continuous) mean distribution H. In this sense, α controls how G "concentrates" around the mean measure H.

---

[1]The acronym stands for Griffiths, Engen, and McCloskey.

Yet another characterization of the Dirichlet process comes from the fact that π can be integrated out analytically (37) from the posterior distribution of Equation 2.15; after observing a sequence of N draws $\theta_1, \theta_2, \ldots, \theta_N$ that assume K distinct values, the next value θ_{N+1} is distributed according to:

$$p(\theta_{N+1}|\theta_1, \theta_2, \ldots, \theta_N, \alpha) = \sum_{k=1}^{K} \frac{n_k}{\alpha + N}\delta_D(\theta_k) + \frac{\alpha}{\alpha + N}H, \tag{2.18}$$

where δ_D indicates the Dirac delta function, and n_K is the number of times θ_k has been encountered during the N previous samples. This means that a value of θ_k that has already been observed will be drawn with a probability proportional to the number of times it has been observed, while with probability proportional to α we will draw a brand new value for θ_{N+1} from the base measure H. This formula is called the "Chinese restaurant process" (CRP) because of the following analogy: imagine a restaurant with infinite tables, each big enough to sit an infinite number of people. When the first customer comes in, she will be seated at the first table, and a menu item θ_1 will be picked for that table from distribution H. When the second customer enters the restaurant, the host will seat him at the first table with probability $\frac{1}{\alpha+1}$, or to a new table with probability $\frac{\alpha}{\alpha+1}$, and in this case a menu item θ_2 for that table will be drawn from H. In general, when the $(N+1)$-th customer enters the restaurant, he will be seated at an already existing table k with probability proportional to the number of customers n_k already seated at that table, and will be served menu item θ_k associated to that table, or

she will be seated at a new table $K+1$ with probability proportional to α, and will be served an item θ_{K+1} drawn from H.

Equation 2.15 can be re-written in terms of an infinite sequence $z = (z_1, z_2, \ldots)$, with $z_i \in \mathbb{N}^+$ for each $k = 1, 2, \ldots, \infty$. We have:

$$\begin{aligned} \pi &\sim \text{GEM}(\alpha) \\ z_i \mid \pi &\sim \pi \qquad\qquad i = 1, 2, \ldots, N \\ \theta_k &\sim H \qquad\qquad k = 1, 2, \ldots \end{aligned} \tag{2.19}$$

Intuitively, z_i is the number of the table that customer i gets seated at, which is assigned to menu item θ_{z_i}. The CRP rule can now be expressed as:

$$\begin{aligned} p(z_{N+1} = k | z_1, z_2, \ldots, z_N, \alpha) &\propto n_k \qquad \text{for existing table } k, \text{ and} \\ p(z_{N+1} = K+1 | z_1, z_2, \ldots, z_N, \alpha) &\propto \alpha \qquad \text{for new table } K+1. \end{aligned} \tag{2.20}$$

2.4.4 Dirichlet Process Mixture Models

The construction above can be used to place a prior probability distribution over mixture models with an infinite number of components. A mixture model generates a draw x_i with probability:

$$p(x_i | \pi, \theta_1, \theta_2, \ldots) = \sum_{k=1}^{\infty} \pi_k \; f(x_i | \theta_k), \tag{2.21}$$

where $f(\cdot|\cdot)$ is the density of some probability distribution parameterized by θ_k. Above, π_k represents the weight, or "importance" of component k. Equation 2.19 naturally provides a

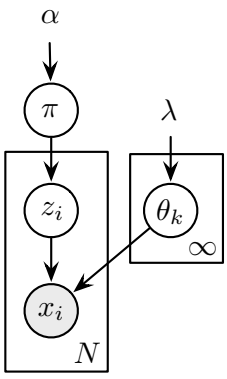

Figure 2. Graphical model of a DPMM.

hierarchical prior distribution over the parameters $(\pi, \theta_1, \theta_2, \ldots)$ of an infinite mixture model. The overall construction is called Dirichlet process mixture model (DPMM) (38):

$$\begin{aligned} \pi &\sim \text{GEM}(\alpha) \\ z_i \mid \pi &\sim \pi \qquad\qquad i = 1, 2, \ldots, N \\ \theta_k &\sim H(\lambda) \qquad\quad k = 1, 2, \ldots \\ x_i \mid z_i &\sim F(\theta_{z_i}) \qquad i = 1, 2, \ldots, N, \end{aligned} \tag{2.22}$$

and can be depicted as the Bayesian network of Figure 2.

DPMMs are important in clustering applications when the number of clusters is not known a-priori. The task is to assign a cluster index, z_i to each observation x_i, $i = 1, 2, \ldots, N$. From a Bayesian perspective, we want to provide a posterior probability over the cluster assignments $p(z_1, z_2, \ldots, z_N | x_1, x_2, \ldots, z_N, \alpha, \lambda)$. Since the computation of this quantity cannot be carried out analytically, approximate methods must be employed. Gibbs sampling is widely used to

tackle this problem. Since the elements of the sequence $x_{1:N}$ are i.i.d., they are also exchangeable. This makes it possible to treat every element in the sequence *as if it were the last one*, and devise its Gibbs conditional distribution: for an observation x_i, given the current value of all the other cluster assignments $z_{-i} = z_1, x_2, \ldots, z_{n-1}, z_{n+1}, \ldots, z_N$ and of the parameters $(\theta_1, \theta_2, \ldots)$, the conditional probability of the cluster assignment z_i can be derived, by applying Bayes' rule and using the CRP of Equation 2.20:

$$\begin{aligned} p(z_i = k | x_i, z_{-i}, \alpha, \lambda, \theta_1, \theta_2, \ldots) &\propto n_k^- \, F(x_i|\theta_k) && \text{for existing table } k, \text{ and} \\ p(z_i = K+1 | x_i, z_{-i}, \alpha, \lambda, \theta_1, \theta_2, \ldots) &\propto \alpha \int_\Theta f(x_i|\theta) dH(\theta|\lambda) && \text{for new table } K+1, \end{aligned} \tag{2.23}$$

where n_k^- is the number of samples currently assigned to cluster k, not considering z_i. It is interesting to observe that the CRP rule embodies the so-called "rich get richer" paradigm: larger clusters are more likely to attract even more samples, and are therefore self-reinforcing.

In some cases, F and H can be picked from conjugate families, so that the integral in Equation 2.23 can be computed analytically, and the θ_k's can be integrated out altogether by employing predictive distributions based on sufficient statistics. Whenever this is not possible, we have to use some more complex algorithm to get around the complexity of computing the integral. Neal (39) provides and compares several algorithms to deal with the non-conjugate case.

It is obvious that a finite amount of samples $x_{1:N}$ can only be grouped finite number of clusters $K \leq N$, although in theory Equation 2.22 defines a probability over infinite clusters. This discrepancy is resolved by assuming that there are indeed infinite potential clusters, but

only a finite amount gets instantiated, i.e. its size is larger than zero. Antoniak (40) derives a formula for the probability of the number of clusters K when N samples are observed (note that this quantity depends only on the number of samples, not their value:)

$$P(K = k|\alpha, N) = s(N, k)\alpha^k \frac{\Gamma(\alpha)}{\Gamma(\alpha + N)}, \tag{2.24}$$

where $s(N, k)$ is the absolute value of the Stirling number of the first kind for N, k. This formula is useful in case one would like to place a hyperprior distribution over the concentration parameter α in order to make the prior more flexible.

2.5 Model-based Multiagent Learning

The concept of learning in multiagent systems is complex and multi-faceted (41), and has been the subject of extensive research both in the field of game theory and agent-based artificial intelligence. In the following, we briefly review some approaches to learning explicit models of other agents. The goal is not to provide a comprehensive survey, but to present some of the related research in order to better contextualize our work.

One of the first approaches to opponent's policy modeling to have been proposed is *fictitious play.* (42). According to this method, an agent, say i, ascribes to another agent, j, a probability over its actions given by the observed empirical distribution of j's previous actions. Despite being a very simple method, fictitious play has been extensively studied in its relation with Nash equilibrium, with a number of theoretical results (43). Fictitious play is applicable to

repeated games with perfect monitoring, that is, the same stage game is played by two agents repeatedly, and the agents can observe each other actions.

A considerably more sophisticated method of learning that applies to the same setting is *rational learning* (44). Given a set of possible strategies S for the agents, and a set of possible histories of a game H, the probability that i ascribes to j's strategy $s_j \in S$, given a history h, is derived by applying Bayes' rule:

$$p(s_j|h) = \frac{p(h|s_j)\, p(s_j)}{\int_S p(h|s_j)p(\mathrm{d}s_j)}. \tag{2.25}$$

A remarkable result for rational learning is that, provided that the probability induced on future plays by an agent's belief over the other agent's strategy is *absolutely continuous* with respect to the distribution induced by the true strategy, the game will converge to a subjective equilibrium. The requirement above is called *absolute continuity condition* (ACC), and is often referred to as the "grain of truth" property: roughly speaking, an agent's belief needs to assign a nonzero probability to the true state of affairs. Unfortunately, assessing the ACC condition is very hard for most but the simplest cases. Moreover, if the set of strategies S is large, infinite, or even uncountable, there is little hope that Equation 2.25 can be implemented analytically.

The interactive POMDP model described in Section 2.3 implements rational learning within the belief update formula (Equation 2.10). Importantly, it extends rational learning to sequential, stochastic, partially observable settings. In (26), the authors extend the subjective equilibrium convergence of rational learning to this more general setting, by re-defining the ACC in

terms of probability over future sequences of observations. However, they also notice how it is impossible for two agents to recursively maintain beliefs over every possible computable strategy. This result suggests that in I-POMDPs the space of strategies of another agents needs to be limited to some extent. Another work that expands on rational learning is (45), that considers environments that evolve stochastically (but are perfectly observable,) in which each agent maintains a belief over the other agents' finite set of types.

In (46), the authors suggest the use of deterministic finite automata (DFA) for opponent modeling, and propose a heuristic algorithm that derives a DFA consistent with past plays. This approach assumes that the environment is a (single-state) repeated game with perfect monitoring, without allowing the modeled agent being itself adaptive. The same authors write in (47): "With this assumption [that the opponent is static] removed, we may want to look at windows of the input rather than the complete history". Interestingly, this is the same method that we follow in Section 6.1.2 for interleaving learning and planning.

One approach to multiagent learning that aims at explicitly modeling adapting opponents is the one in (48). As before, the authors focus on repeated games with perfect monitoring, and establish a series of desirable properties for opponent modeling: (*i*) if the opponent is a member of the target class, the algorithm should yield an ϵ-best response; (*ii*) in self-play, it should converge to an ϵ-Nash equilibrium; and (*iii*) against any opponent, it should stay within ϵ of the security value of the game. The algorithm therein provided satisfies these properties, and targets a class of opponents whose policy only depends on a history of finite length. The work in (49) makes similar assumptions, and proposes to solve the game in the space of the

"adversary induced MDP", whose state space is the set of all possible joint histories of a specific length, that can be solved with MDP methods.

2.5.1 Relations to Our Approach

We can recognize commonalities between some of the related work surveyed above and the approach we propose in this paper. Like (44) and related methods, we also use the Bayesian update as the fulcrum of learning. For obvious reasons, our work is related to the analysis of convergence in I-POMDPs (26), and in particular shares the same generality about the type of environment considered. A trait in common with (46) is to use finite-state models of other agents, and that we address their adaptability by implementing an approach similar to what was hinted at by the same authors in (47). On a related note, (48) and (49) consider models of opponents whose actions are based on a limited history, and whose adaptability can hence be modeled by considering a finite amount of observations.

2.5.2 The Case for Explicit Opponent Modeling

In the work presented in this paper, as well as in the literature surveyed above, an agent forms *explicit* models of other agents' policies, against which to provide a best response. However, there has been considerable effort in multiagent learning towards algorithms that model the opponents *implicitly* (50; 51; 52). The choice of which of these two paradigms is best suited to solve the problem at hand depends on assumptions about the prior knowledge that the agent has about the environment.

In our case, we assume that that an agent, say i, knows its own I-POMDP parameters. This means that i knows its reward function, its observation function, and the world's transition

function as a response to both agents' actions. Note that this knowledge induces a problem that is different from the one solved by reinforcement learning (RL), where such assumptions are usually not made. Another subtle difference with RL, sometimes overlooked, is that in POMDPs (and hence in I-POMDPs) the agent does not observe its rewards at each timestep, unless this is explicitly encoded in the observation function.

Since agent i knows the world's transition function, the missing piece of information is a mapping $g_j : \Omega_i^* \times A_i^* \rightarrow a_j^{t+1}$ from i's observations to j's actions. In this paper, we further assume that j's observation function is known, allowing i to speculate about j's observations and actions, given its own. Therefore, instead of inferring g_j, the agent aims at directly learning j's agent function $f_j : \Omega_j^* \times A_j^* \rightarrow A_j$. This is precisely the type of opponent modeling that takes place in I-POMDPs, using either intentional or subintentional models.

Surely, agent i could still treat agent j's policy implicitly, and fold it into the transition function as noise, but that would imply that j's actions depend at most on just the current state of the world, which is unrealistic since j's decision making is in general more involved. We claim that using finite state models is an appropriate hypothesis space for j's policies, especially since we do not bound the number of nodes, hence allowing j's actions to depend on a history of unbounded length.

Without the prior knowledge assumptions made above, modeling the other agent explicitly might be ineffective, whereas POMDP reinforcement learning techniques, such as utile suffix memory (53), Bayes-adaptive POMDPs (54), infinite generalized policy representation (55),

infinite POMDPs (56), and others would be more suitable to solve the problem, by learning a model of the environment that implicitly contains the other agent's policy.

CHAPTER 3

A PRIOR DISTRIBUTION FOR PDFCS

As mentioned in Chapter 1, the goal of this work is the design and implementation of an I-POMDP agent, henceforth i, that operates in a stochastic, partially observable, multiagent environment, and models the other agent, j, explicitly as a PDFC. Since the exact model of the other agent is not known a-priori, agent i needs to infer an accurate model of j from its own observations.

We propose a Bayesian methodology that yields learning over the class of possible PDFCs, denoted as C_j, given an observed trajectory (or history) $h_{1:T}$ that provides some information about an agent j's behavior. We want to compute the posterior distribution:

$$p(c_j|h_{1:T}) \propto p(h_{1:T}|c_j)\, p(c_j). \tag{3.1}$$

Crucial to this task is providing a suitable prior distribution $p(c_j)$ over PDFCs. Since agent j's complexity is unknown to agent i, we do not wish to bound, a priori, the number of nodes of j's PDFC. Instead, we want to provide a prior probability that allows the complexity of the learned PDFC to scale with the complexity of the observed behavior: if agent j follows a simple policy, we want to learn a small PDFC that best captures our observations; on the other hand, if j's behavior exhibits complex patterns, we want to be able to learn a more complex model that explains such regularities.

Recall from Section 2.2 that each node k of a PDFC is associated to a continuous parameter $\theta_k \in \Delta(A_j)$. Because of this, the set of PDFCs (of unbounded size) C_j is an infinite-dimensional sample space. We described in Section 2.4.3 a prior probability over infinite dimensional spaces based on the Dirichlet process, along with its application as a prior for infinite mixture models. In this chapter, we propose a variation of such prior distribution that applies to the space of PDFCs.

In a related work (57), a hierarchical stick-breaking prior has been proposed for probabilistic deterministic finite automata (PDFA) in the context of language modeling. The transition function of a PDFA depends on the previous node and on the action there executed, and not on an external input signal (the observation) as in the case of PDFCs, which are in fact a generalization of PDFAs. Nevertheless, the methodology described by the authors bears some similarities with our work. Moreover, stick-breaking priors have been used over the space of policies for decentralized POMDPs (58). While there are similarities in the way that prior probabilities over controllers are defined, the cited work presents a "planning as inference" methodology for decentralized POMDPs, targeting a very different problem than the one we consider in our work.

3.1 Dirichlet Process Prior for PDFCs

In this section, we describe the prior distribution over PDFCs that we adopt. The main idea is to view the PDFC inference problem as clustering the transitions of a PDFC into groups, each assigned to a node of the PDFC that represents the destination of the transitions belonging to that group. Recall from section 2.2 that each transition is identified by a starting node, an

action, and an observation. By determining the destination node of each transition, we have defined the PDFC's transition function τ.

For each starting node $k = 1, 2, \ldots, \infty$, action $g = 1, 2, \ldots, |A_j|$, and observation $h = 1, 2, \ldots, |\Omega_j|$, the destination node is drawn from a discrete infinite probability vector π. This vector is in turn distributed according to the stick-breaking process with concentration parameter α defined in Equation 2.17. For each node $k = 1, 2, \ldots, \infty$, the corresponding emission distribution over A_j is drawn from a symmetric Dirichlet distribution with total parameter λ. Intuitively, the parameter λ encodes our prior belief about the entropy of the emission distributions of the PDFC: a large value of λ reflects a bias towards more stochastic action generation functions, while lower values favor emission distribution peaked around one action. Setting $\lambda = |A_j|$ yields a non-informative (flat) prior distribution. Note that the Dirichlet distribution is conjugate to the multinomial distribution from which actions are generated in each node. Summarizing, we define the following hierarchical prior distribution over the space of PDFCs C_j:

$$\begin{array}{rcll} \pi \mid \alpha & \sim & \mathrm{GEM}(\alpha) & \\ \tau_0 \mid \pi & \sim & \pi & \\ \tau_{kgh} \mid \pi & \sim & \pi & k = 1..\infty;\ \ g = 1..|A_j|;\ \ h = 1..|\Omega_j| \\ \theta_k \mid \lambda & \sim & \mathrm{Dir}\left(\frac{\lambda}{|A|}, \ldots, \frac{\lambda}{|A|}\right) & k = 1..\infty \end{array} \tag{3.2}$$

An important property of the stick-breaking distribution is that the vector π can be integrated out analytically.

The derivation of the Chinese restaurant process formula carries over to this application, making it possible to define the conditional probability of τ_{kgh} given all other previously allocated transitions $\tau_{-(kjh)}$ as:

$$\begin{aligned} p(\tau_{kgh} = i | \alpha, \tau_{-(kgh)}) &\propto v_i \quad \text{for existing node } i \\ p(\tau_{kgh} = \bar{i} | \alpha, \tau_{-(kgh)}) &\propto \alpha \quad \text{for new node } \bar{i}, \end{aligned} \tag{3.3}$$

where v_i is the number of transitions in $\tau_{-(kgh)}$ that point to node i, not considering τ_{kgh}, and including the starting node τ_0, i.e.:

$$v_i = \delta_K(\tau_0, i) + \sum_{(k',g',h') \neq (k,g,h)} \delta_K(\tau_{kgh}, i). \tag{3.4}$$

As we noted in Section 2.4.4 for Dirichlet process mixture models, the CRP induces a "rich get richer" property on the nodes of the PDFC: transitions are biased to point to nodes that already receive a large amount of incoming transitions. This bias is not undesirable in our case: if we imagine that each node of a PDFC is associated to a unique region of agent j's belief space (as is the case for POMDP controllers,) then the self-reinforcing property of the CRP reflects the fact that some belief regions represent more common "mental states" for the agent, such as reset states. The histograms in Figure 3 depict the number of incoming transitions (v_i, sorted in descending order) for a sample from the PDFC prior with different values of α, with $|A_j| = 3$ and $|\Omega_j| = 2$.

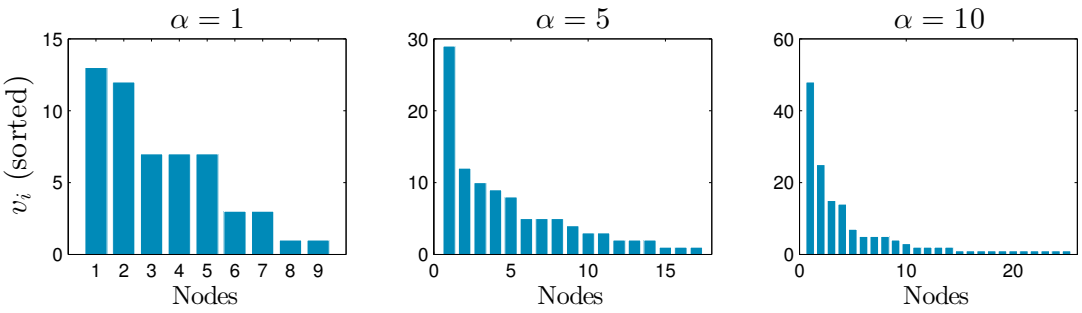

Figure 3. Number of incoming transition per node, sorted, with different values of α.

Strictly speaking, Equation 3.2 defines a distribution over PDFCs with infinite nodes, and not an unbounded finite number of nodes. However, we are interested only in the finite "connected component" containing the nodes that are reachable from the initial node, ignoring the infinite subset of nodes that are not connected. Conceptually, this is akin to more conventional DPMM priors used for clustering, where the result of inference is a finite amount of clusters, even though the prior contemplates infinite components (see Section 2.4.3). It is useful to determine analytically the probability over the effective number of nodes K induced by our prior distribution.

3.1.1 Induced Probability of the Number of Nodes

We want to obtain the probability of K, the number of nodes that gets "instantiated" when drawing from the prior in Equation 3.2 as a function of the concentration parameter α, the number of actions $|A_j|$ and observations $|\Omega_j|$. We can view the process of sampling a PDFC from the prior recursively, starting from one single node and drawing its outgoing transitions according to Equation 3.3, some of which may point to new nodes; we then do the same with

the second node, if any, and so on. By "instantiated" nodes, we refer to the nodes drawn as a result of this procedure. Since the prior is an exchangeable probability distribution, there is no loss of generality in interpreting a draw from $p(\tau|\alpha)$ sequentially as above.

Let us now derive the probability over the number of nodes K induced by this sequential drawing procedure. We observe that K is the index of the first node whose outgoing transitions $\tau_{K\cdot\cdot}$ all point to already existing nodes (including node K itself.) We will start from $K = 1$, $K = 2$, $K = 3$, and then derive a general rule. Let us denote as $Z = |A_j||\Omega_j|$ the number of outgoing transitions from each node. In the following, we index the transitions in the order that they are sampled in our schema, so that transitions $1 \leq l \leq Z$ are from the first node, transitions $(Z + 1) \leq i \leq 2Z$ are from the second node, and so on. From what we described above, we know that $K = 1$ if and only if all of the first node's outgoing transitions point to itself, i.e., no new node is generated besides the first, which is created with probability one $(\frac{\alpha}{\alpha})$. According to the CRP rule, the probability of this happening is:

$$p(K = 1|\alpha) = \frac{\alpha}{\alpha}\frac{1}{(1+\alpha)}\frac{2}{(2+\alpha)}\cdots\frac{Z}{(Z+\alpha)} = \frac{\alpha Z!}{\alpha^{(Z+1)}}, \tag{3.5}$$

where $\alpha^{(Z+1)}$ is the Pochhammer symbol indicating the rising factorial $\alpha^{(Z+1)} = \alpha(\alpha+1)(\alpha+2)...(\alpha+Z)$.

For $K = 2$, it must be the case that at least one of the first node's outgoing transitions points to the second node, and the second node's transitions all point to the first or second node. The transition from the first to the second node with the lowest index, that is, the one

that "generated" the second node when sampled, can be any of the first node's Z outgoing transitions, therefore:

$$p(K=2|\alpha) = \frac{\alpha}{\alpha^{(2Z)}}\frac{(2Z)!}{Z!}\left(\alpha\cdot 2\cdot \ldots \cdot Z \;+\; 1\cdot\alpha\cdot\ldots\cdot Z \;+\; \ldots \;+\; 1\cdot 2\cdot\ldots\cdot\alpha\right). \tag{3.6}$$

The sum of products between round brackets is the combinatorial quantity whose computation is critical in the general case.

Let us now consider $K = 3$: we know that there must be one transition from the first node, having index say $l \leq Z$, that points to the second node (and contributes "one α") and one transition indexed $l < l' \leq 2Z$ that goes to the third node. This transition may originate from either the first or second node. The sum of products resulting from all such possible configurations of new transitions to the second and third node is needed to compute $p(K=3|\alpha)$. For a generic K, we have to consider all the "legal" configurations of the $(K-1)$ "α's" that occur in the nodes previously sampled. We formalize this concept by introducing some definitions.

Definition 1. *A **configuration** for a PDFC with K nodes is a binary vector $w^K = (w_1^K, w_2^K, \ldots, w_{(K-1)Z}^K)$ of length $(K-1)Z$, containing exactly $(K-1)$ zeros. Intuitively, the position of the first zero in this sequence identifies the first transition that was sampled to point to the second node, the second zero indicates the transition that first points to the third node, and so on. We denote as L_k the position of the k^{th} zero in a configuration. By convention, $L(0) = 0$.*

Therefore, L_{K-1} is the first transition that points to node K in a PDFC with K nodes. We know that this transition must be drawn after the first transition to node $(K-1)$ is drawn. This leads to the definition of "legal" configuration.

Definition 2. *A configuration w^K is **legal** if, for all $0 < k < K$, we have that $L_{k-1} < L_k < Z(K-1)$. We denote as W^K the set of all legal configurations for a PDFC with K nodes.*

Each legal configuration w^K is associated to a quantity $z(w^K)$, that is the product of the positions of ones in the configuration, i.e. $z(w^K) = \prod_{i=1}^{Z(K-1)} i\, w_i^K$. The combinatorial quantity that we need for computing the probability of having K nodes, denoted as $q(K)$, is the sum of such quantities for all legal configurations, i.e.

$$q(K) = \sum_{w^K \in W^K} z(w^K). \tag{3.7}$$

If $q(K)$ is known, then the probability of having K nodes is given by

$$p(K|\alpha) = \frac{\alpha^K}{\alpha^{(KZ+1)}} \frac{(KZ)!}{((K-1)Z)!} q(K), \tag{3.8}$$

where:

- α^K are the numerators of the CRP terms corresponding to transition draws that resulted in the creation of new nodes, including the α in the first vacuous term $\frac{\alpha}{\alpha}$ that "creates" the first node;

- $\alpha^{(KZ+1)} = \alpha(\alpha+1)(\alpha+2)...(\alpha+KZ)$ is the rising factorial, resulting from the product of the denominators of the CRP conditional distributions;
- $\frac{(KZ)!}{((K-1)Z)!} = \big((K-1)Z \cdot ((K-1)Z+1) \cdot \ldots \cdot KZ\big)$ are the numerators of CRP terms for transitions outgoing from the last node K, that did not result in the creation of any new node;
- $q(K)$ is the sum of products of legal configurations, described above.

Figure 4 shows the probability over the number of nodes computed from Equation 3.8 when $Z = 6$, for different values of α. It is clear that the expected number of nodes increases with α. This is not surprising: a higher value of α implies a higher likelihood of sitting customers (transitions) at a new table (node) instead of an existing one, in the Chinese restaurant analogy. Indeed, the prior in Equation 3.2 can be intuitively described with an extension of the restaurant analogy: instead of receiving a number N of customers, our new restaurant process starts by receiving one customer and sitting her at the first table. This causes Z more customers to enter the restaurant, which are seated according to the rule of the CRP. For every new table that gets allocated, Z more customers enter the restaurant, until no new tables are created.

Figure 5 shows a histogram representing the empirical distribution of the number of nodes K of 10000 PDFCs drawn from the prior distribution with $\alpha = 5$ and $Z = 6$, along with a line representing the probability computed analytically. We can see that the two distributions converge.

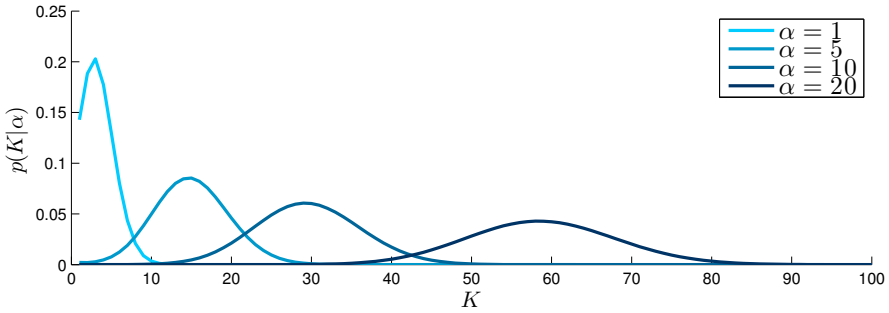

Figure 4. Probability of PDFC size, for different values of α.

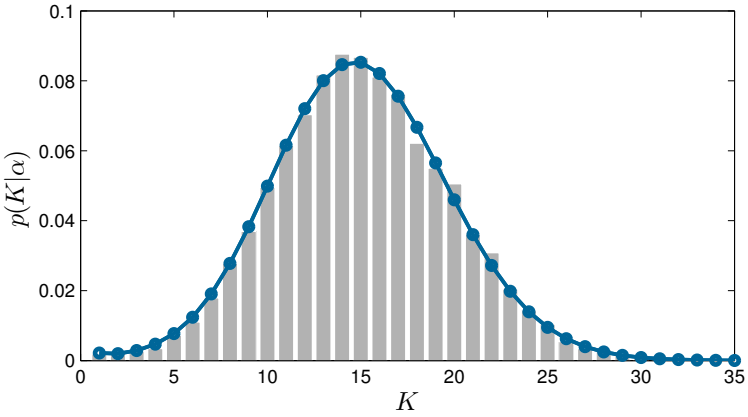

Figure 5. Empirical (bars) and exact (line) probability of PDFC size.

3.1.1.1 Efficient Computation

A brute-force computation of the q terms in Equation 3.8, according to the formula in Equation 3.7, would have exponential complexity. In the following, we instead describe a way to compute $q(K)$ more efficiently. Let us introduce the quantity $q(K, l)$, that represents the sum of products $z(w^K)$ for legal configurations having the last zero in position l, i.e. $L_{K-1} = l$. Since

the last zero in a legal configuration for a PDFC with K nodes can occur between positions $K-1$ and $Z(K-1)$, we have that $q(K) = \sum_{l=K-1}^{Z(K-1)} q(K,l)$. In order to make its manipulation easier, we decompose $q(K,l)$ into the sum of products of the configurations truncated at index l included, denoted as $\bar{q}(K,l)$, and the remaining product of the configuration (which does not contain any zero), i.e.:

$$q(K,l) = \bar{q}(K,l)\ (l+1)(l+2)\dots((K-1)Z). \tag{3.9}$$

It follows that:

$$q(K) = \sum_{l=K-1}^{Z(K-1)} \bar{q}(K,l)\frac{((K-1)Z)!}{l!}. \tag{3.10}$$

We can now derive a recursive relation for $\bar{q}(K,l)$ from $\bar{q}(K,l-1)$. When "moving" the position of the last α from $(l-1)$ to l, we have to multiply the previous $\bar{q}$ by $(l-1)$, since in the corresponding configuration the element w_{l-1}^{K} switched from 0 to 1. Moreover, by shifting the position of the last α to l, we must acknowledge that there are now potentially more configurations that are legal for the first $(K-2)$ α's, that is to say L_{K-2} can now take the value $l-1$. This is only true when $l-1$ is a legal value for L_{K-2}, i.e. when $(l-1) < (K-2)Z$. Putting all this together, we have:

$$\bar{q}(K,l) = (l-1)\ \bar{q}(K,l-1) + \begin{cases} \bar{q}(K-1,l-1) & \text{if } (l-1) < (K-2)Z \\ 0 & \text{otherwise.} \end{cases} \tag{3.11}$$

Computing the values of q in this way has a complexity of $O(K^2)$, much lower than $O(2^K)$ that results from direct computation of Equation 3.7. Moreover, these values can be pre-computed and stored, since they are not dependent on α, and used when needed.

CHAPTER 4

PDFC INFERENCE FROM FULLY OBSERVED BEHAVIOR

4.1 Learning Setup

In this chapter, we assume that agent i perfectly observes agent j's history, i.e. the sequence of j's actions and observations $h^j_{1:T} = (\omega^j_{1:T}, a^j_{1:T})$. This assumption is sometimes unrealistic, but it allows to evaluate the PDFC learning algorithm under ideal conditions. In Chapter 5, this assumption will be relaxed and the learning algorithm extended to cope with the partially observable case. Moreover, we assume throughout this work that agent j's observation function O_j is known by agent i. However, we never assume any knowledge about j's reward function R_j; this makes the proposed approach very general, since it reflects agent i's lack of knowledge about the nature of the opponent, that could be competitive, cooperative, indifferent, or else.

The Bayesian learning of agent j's PDFC c_j, with perfect observability of j's behavior trajectory, can be formally defined as computing the posterior distribution:[1]

$$p(c_j|h^j_{1:T}) = p(\tau, \theta|\omega^j_{1:T}, a^j_{1:T}) = \frac{p(a^j_{1:T}|\tau, \theta, \omega^j_{1:T})\; p(\tau, \theta)}{p(a^j_{1:T}|\omega^j_{1:T})}, \tag{4.1}$$

where θ is a shorthand for the sequence $(\theta_1, \theta_2, \ldots)$. Notice that the product of Bayesian learning is not one single PDFC, but instead a distribution (the posterior) over all possible PDFCs.

---

[1]Here and in the remainder of this document, we interpret τ as including the information about the initial node τ_0, unless otherwise stated.

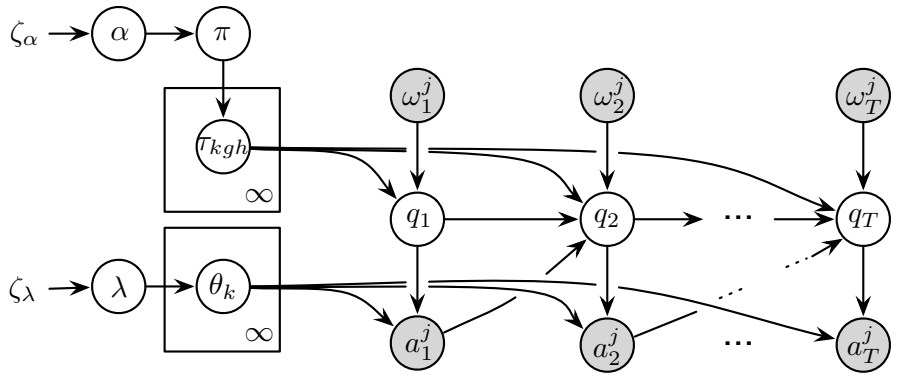

Figure 6. Graphical model representation of PDFC learning setup, with observable observation/action pairs.

The dynamic Bayesian network in Figure 6 depicts the learning scenario graphically. We place the prior distribution defined in Equation 3.2 over the PDFC's parameters (τ, θ). The rest of the conditional distributions of the model are given by the dynamics (transition and emission function) of the PDFC being learned:

$$\begin{aligned} q_1 \mid \tau &= \tau_0 \\ a_t \mid q_t, \theta &\sim \theta_{q_t} \qquad\qquad t = 1, \ldots, T \\ q_{t+1} \mid q_t, a_t^j, \omega_{t+1}^j, \tau &= \tau_{q_t a_t^j \omega_{t+1}^j} \qquad t = 1, \ldots, T-1. \end{aligned} \tag{4.2}$$

Moreover, we place exponential hyperpriors over the concentration parameter α of the stick-breaking distribution and the parameter λ of the symmetric Dirichlet over θ_k. This makes the

learning more flexible with respect to the number of nodes and the entropy of the emission distributions of the PDFC. Specifically, we have:

$$\begin{aligned} \alpha \mid \zeta_\alpha &\sim \exp(\zeta_\alpha) \\ \lambda \mid \zeta_\lambda &\sim \exp(\zeta_\lambda). \end{aligned} \tag{4.3}$$

4.2 MCMC Sampler for PDFC Inference

Computing the posterior distribution in Equation 4.1 is not analytically tractable. This is usually the case for complex models in Bayesian learning. In this work, we adopt a Markov-chain Monte Carlo (MCMC) algorithm to approximate the posterior distribution, inspired by previous research on DPMM inference. Using a Monte Carlo method means that we will obtain an empirical distribution that approximates the true posterior. In other words, we will obtain an ensemble of candidate PDFCs of agent j, and not just a single model.

In general, the state of the Markov chain is a value assignment to all the PDFC parameters that we need to learn, including the hyperparameters α and λ. However, we have seen in Section 2.4.3 that the weight vector $\pi \sim \text{GEM}(\alpha)$ can be integrated out analytically via the CRP construction. Moreover, we placed a Dirichlet prior over each node's emission parameter θ_k, that is conjugate to the multinomial action generation. Therefore, the emission parameters θ can also be integrated out analytically. Lastly, since in a PDFC the next node depends deterministically on the value of the previous node, its action, and the new observation, there is no need to include it explicitly in the state: the sequence $q_{1:T}$ can be derived at will when

needed. The resulting *collapsed* state is therefore a tuple $(\tau, \lambda, \alpha)^{(n)}$, where n indicates the n-th iteration of the MCMC algorithm.

We can devise a Gibbs sampling procedure on this space that, at each iteration, uniformly draws a transition and re-samples its destination. Recall that a transition in a PDFC with K nodes is identified by a starting node $1 \leq k \leq K$, an action $1 \leq g \leq |A^j|$, and an observation $1 \leq h \leq |\Omega^j|$, and takes value τ_{kgh}. According to this schema, given a triple (k, g, h) identifying a transition, the next state in the Markov chain is obtained by:

$$\tau_{kgh}^{(n+1)} \sim p(\tau_{kgh} | \tau_{-(kgh)}^{(n)}, \alpha^{(n)}, \lambda^{(n)}, \omega_{1:T}^{j}, a_{1:T}^{j}), \tag{4.4}$$

where we note that $\tau_{-(kgh)}$ also includes the value of the initial node τ_0. We call this jumping distribution an **incremental move**. The details on how to draw the new value for a single transition are provided in Section 4.2.1.

Despite being relatively easy to implement, allowing only one transition to change at each iteration may negatively affect the mixing time, and potentially cause the MCMC algorithm to get stuck in local modes of the posterior distribution for long periods of time, a problem not new in the context of Gibbs sampling for mixture models (59; 60). This is because in order to go from a local mode to a state with higher probability, the incremental Gibbs sampler might need to pass through a sequence of states of low probability, effectively preventing the state from ever reaching the global mode of the posterior distribution. In DPMMs, this might prevent the

creation of new components. In our case, incremental moves might not be sufficient to reach the region of the space containing PDFC configurations with an adequate number of nodes.

In order to tackle this problem, we implement **split-merge moves** (61), that split a whole node or collapse two nodes in a single step, thus enabling a more effective exploration of the sample space. Split-merge moves are computationally more expensive than incremental moves, therefore they are applied only every R^{th} iterations, where R is a parameter of our MCMC algorithm. A detailed description of this step is provided in Section 4.2.2.

Additionally, the algorithm resamples the hyperparameters α and λ at each iteration. This is done via a Metropolis-Hastings (MH) step, as detailed in Section 4.2.3.

The overall MCMC algorithm, whose structure is provided in Algorithm 1, employs a general Gibbs sampling schema, and incorporates Metropolis-Hastings moves for splitting and merging nodes and sampling hyperparameters. It is therefore an instance of *hybrid MCMC sampling*.

4.2.1 Incremental Moves

In order to perform an incremental Gibbs move, we first sample a single transition source uniformly at random out of the $K|A^j||\Omega^j|+1$ transitions of the PDFC in the current state, where K is the current number of PDFC nodes. Note that the initial node τ_0 is just a special case of transition, hence the '+1' above. Let the sampled transition be indexed as kgh, indicating respectively the source node, the action, and the observation it corresponds to; the special index '0' indicates the initial node.

We then proceed to sample the destination of this transition from its conditional distribution, given the current values of all other state variables. Such destination node can be one of the

Algorithm 1 LearnPDFC-PO

Input: $\omega^i_{1:T}, a^i_{1:T}, M, R, S, N_{iter}$
Output: $\tau^{(1:N_{iter})}, \alpha^{(1:N_{iter})}, \lambda^{(1:N_{iter})}$

▷ Initialize PDFC

$\tau_0^{(1)} \leftarrow 1$
$\tau_{1gh}^{(1)} \leftarrow 1 \quad \forall\, g = 1..|A^j|, h = 1..|\Omega^j|$
$\alpha^{(1)} \sim exp(\zeta_\alpha)$
$\lambda^{(1)} \sim exp(\zeta_\lambda)$

▷ Initialize hidden sequences

$(s_{1:T}, a^j_{1:T}, \omega^j_{1:T})^{(1)} \leftarrow$ sample-seq$(\tau, \lambda, \omega^i_{1:T}, a^i_{1:T})$

▷ MCMC iterations

for $n = 2..N_{iter}$ **do**
  if mod $(n, R) \neq 0$ **then**
    $\tau^{(n)} \leftarrow$ incremental-move$(\tau^{(n-1)}, \alpha^{(n-1)}, \lambda^{(n-1)}, (a^j_{1:T}, \omega^j_{1:T})^{(n)}, M)$
  else
    $\tau^{(n)} \leftarrow$ split-merge$(\tau^{(n-1)}, \alpha^{(n-1)}, \lambda^{(n-1)}, (a^j_{1:T}, \omega^j_{1:T})^{(n)}, S)$
  end if
  $(s_{1:T}, a^j_{1:T}, \omega^j_{1:T})^{(n)} \leftarrow$ sample-seq$(\tau^{(n)}, \lambda^{(n)}, \omega^i_{1:T}, a^i_{1:T})$
  $(\alpha^{(n)}, \lambda^{(n)}) \leftarrow$ sample-hyperpars$(\tau^{(n)}, \alpha^{(n-1)}, \lambda^{(n-1)}, (a^j_{1:T}, \omega^j_{1:T})^{(n)})$
end for

existing nodes or an entirely new one, that is added to the PDFC. By using Bayes rule and applying the conditional independence encoded in the model's structure (see Figure 6), we obtain:

$$p(\tau_{kgh}|\tau_{-(kgh)}, \omega^j_{1:T}, a^j_{1:T}, \alpha, \lambda) \propto p(\tau_{kgh}|\alpha, \tau_{-(kgh)})\; p(a^j_{1:T}|\tau, \omega^j_{1:T}, \lambda), \tag{4.5}$$

where as before $\tau_{-(kgh)}$ denotes all current values of τ except the one being sampled, and $\tau = \tau_{kgh} \cup \tau_{-(kgh)}$.

The first term of the RHS side of Equation 4.5 is the conditional prior distribution, given by the Chinese restaurant process in Equation 3.3. The second term of the RHS is the likelihood

that the new assignment awards to the action sequence $a^j_{1:T}$, given j's observation sequence $\omega^j_{1:T}$. For existing nodes, this can be computed by considering that τ, $a^j_{1:T}$, and $\omega^j_{1:T}$ jointly determine the value of the node sequence $q_{1:T}$.

The likelihood can be then obtained as the expectation of Equation 2.8 with respect to the measure $p(\theta_k|\lambda)$, for each node k. Since this latter density is in our model a conjugate prior to the multinomial action generation of the PDFC, this expectation can be easily computed in closed form as follows. Let us introduce a count matrix d, where each element d_{kg} represents how many times action g is generated in node k in such sequence, that is:

$$d_{kg} = \sum_{t=1}^{T} \delta_K(q_t, k)\delta_K(a^j_t, g). \tag{4.6}$$

Given that each action is conditionally independent, we can use the properties of the Dirichlet-multinomial model (62), and marginalize over the parameters θ_k's, to obtain:

$$p(a^j_{1:T}|\omega^j_{1:T}, \tau, \lambda) = \prod_{k=1}^{K} \left[\frac{\Gamma(\lambda)}{\Gamma(d_{k\cdot} + \lambda)} \prod_{g=1}^{|A^j|} \frac{\Gamma(d_{kg} + \lambda/|A^j|)}{\Gamma(\lambda/|A^j|)} \right], \tag{4.7}$$

where the quantity $d_{k\cdot}$ is the number of times node k is visited, i.e. $d_{k\cdot} = \sum_{g=1}^{|A^j|} d_{kg}$.

Computing the likelihood term of assigning τ_{kgh} to a *new* node is more complicated. This is because, when a new node is considered, its own outgoing transitions need to be evaluated. According to the prior, such transitions can in turn point to some other new node, and so on recursively. It is therefore unfeasible to sum over all the countably infinite possible transition configurations that stem out of the new node. This situation is akin to DPMMs with non-

conjugate priors, where the components' parameters cannot be integrated analytically. In our case, the parameters θ_k's are given a conjugate Dirichlet prior, and hence θ can be integrated out analytically. However, the new node's outgoing transitions, that can be thought as additional parameters of the nodes, cannot.

A simple solution to this problem could be the use of a Metropolis-Hastings (MH) step instead of Gibbs to sample the new transition, similarly to the algorithm proposed in (57). In our case, however, such method leads to slow mixing rates. A better solution is to adapt the *auxiliary variables* algorithm described in (39), as described in the next section.

4.2.1.1 Incremental Gibbs Sampling with Auxiliary Variables

The key idea behind this method is to approximate the integration over possible new nodes by sampling M candidate transition configurations for the new node (i.e. the new node's own outgoing transitions) from the conditional prior distribution, which is obtained by recursively sampling from the CRP until no new node is generated. Once the likelihood of these candidates is evaluated, we sample the transition τ_{kgh} from Equation 4.5, distributing α uniformly among the M candidates, so that the total prior probability of generating a new node is still proportional to α. In the following, we provide the details of this procedure.

For this approach to work, we need to be able to sample a number M of candidate new nodes, that are the "auxiliary variables" the name of the method refers to. These candidate nodes are characterized by their own outgoing transitions, which are sampled as needed from the conditional prior distribution, given by the CRP rule in Equation 2.18. Sampling from the CRP might in turn create additional new nodes, whose outgoing transitions need to be sampled

Algorithm 2 sample-new-node(τ, K, k, g, h)

$\forall k = 1..K \quad v_k \leftarrow \left|\left\{k \ : \ \tau_{k'g'h'} = k\right\}\right|$
$v_{\tau_{kgh}} \leftarrow v_{\tau_{kgh}} - 1$
$\tau_{kgh} = K + 1$
$v \leftarrow (v_1, v_2, \ldots, v_K, 1)$
$K \leftarrow K + 1$
$k' \leftarrow K + 1$
while $k' \leq K$ **do**
    for $g' \in \{1, \ldots, |A|\}$ **do**
        for $h' \in \{1, \ldots, |\Omega^j|\}$ **do**
            $\tau_{k'g'h'} \leftarrow \text{Mult}((v_1, \ldots, v_K, \alpha))$ *//CRP*
            if $\tau_{k'g'h'} = K + 1$ **then**
                $v \leftarrow (v_1, \ldots, v_K, 1)$
                $K \leftarrow K + 1$
            else
                $v_{\tau_{k'g'h'}} \leftarrow v_{\tau_{k'g'h'}} + 1$
            end if
        end for
    end for
    $k' \leftarrow k' + 1$
end while
return τ

recursively, and so on until no new nodes are generated. Each candidate new "node", in fact, is in general a set of new nodes made reachable from the existing PDFC through the transition $\tau_{-(kgh)}$, rather than a single node. For this reason, we refer to each candidate configuration as $\tilde{\tau}^m$, where $m \in \{1, \ldots, M\}$, indicating the whole PDFC topology resulting from attaching the m^{th} candidate set of new nodes thus generated to the existing PDFC. The iterative procedure that generates each candidate new configuration is detailed in Algorithm 2.

Using this procedure, we sample the M new candidate nodes, that are used as possible values for τ_{kgh}, along with the set of already existing nodes. The new value for τ_{kgh} is then sampled using the following probabilities:

$$p(\tau_{khg} = k' | \tau_{-(khg)}, \omega^j_{1:T}, a_{1:T}) \propto \begin{cases} v_i \; p(a_{1:T} | \tau_{-(kgh)}, \tau_{kgh} = i, \omega^j_{1:T}) & \text{for } i = 1, \ldots, K \\ \frac{\alpha}{M} \; p(a_{1:T} | \tilde{\tau}^{i-M}, \omega^j_{1:T}) & \text{for } i = K + (1, \ldots, M) \end{cases}, \tag{4.8}$$

where the likelihood terms in the formula are computed as in Equation 4.7.

4.2.2 Splitting and Merging Nodes

This step starts by sampling two transitions uniformly at random. If these transitions point to the same node, a split of such node is proposed, otherwise a merge of the two destination nodes is proposed. Once a split or merge is proposed, it is accepted or rejected using the MH criterion. It is important to propose "high-quality" splits, since it is intuitive that just splitting the node randomly will probably not represent an improvement, and hence the move will likely be rejected by the MH criterion. In order to do so, the algorithm described in (61) is adapted to our case.

When splitting a node, its incoming transitions are re-directed towards either one of the two newly created nodes using S iterations of a *restricted Gibbs sampler* (S is a parameter of the algorithm,) that also samples the new nodes' outgoing transitions. This produces a split that reflects to some extent the observed data instead of being just randomly sampled, and

hence has a higher chance of being accepted. A merge is proposed using a similar method, that collapses two nodes into one and samples its outgoing transitions. The detailed description of this procedure is provided in the next section.

4.2.2.1 Split-Merge Proposals

We provide here the details of the procedure that implements split-merge moves in our MCMC algorithm. This procedure is inspired by (61) and adapted to our case.

Let us consider the transition function τ as a $K \times |A^j| \times |\Omega^j|$ matrix, where K is the current number of nodes. We will denote an element of such matrix as $\tau_u \in \{1, \ldots, K\}$, with $u = (k, g, h)$ indicating that the PDFC will transition to the node indexed by the value of τ_u starting from node indexed as $k \in \{1, \ldots, K\}$, after action indexed by $g \in \{1, \ldots, |A^j|\}$ is executed and observation $h \in \{1, \ldots, |\Omega^j|\}$ is received.

We will describe a procedure for proposing a new matrix τ^* from the current one. Such matrix can be the result of splitting a node into two nodes, or merging two nodes into one. Regardless of which type of move (split or merge) is performed, the proposed transition matrix τ^* is accepted stochastically according to the Metropolis-Hastings acceptance ratio, defined as (we omit the dependence on the hyperparameters α and λ for readability):

$$a(\tau^*, \tau) = \min \left[1, \frac{q(\tau|\tau^*)}{q(\tau^*|\tau)} \frac{p(\tau^*)}{p(\tau)} \frac{p(a^j_{1:T}|\tau^*, \omega^j_{1:T})}{p(a^j_{1:T}|\tau, \omega^j_{1:T})} \right]. \tag{4.9}$$

In the equation above, the prior probability terms can be computed from sequentially applying the CRP formula in Equation 2.18, and the likelihood terms are computed form Equa-

tion 4.7. The jumping distribution $q(\tau^*|\tau)$ is responsible for generating a good quality candidate configuration.

The split-merge procedure is composed by the following steps, adapted from (61), to which we refer for further explanation of the statistics details involved.

1. Sample a pair of indexes $u = (k', h', j')$ and $w = (k'', h'', j'')$ uniformly at random.

2. Let Z denote the set of indexes z such that $\tau_z = \tau_u$ or $\tau_z = \tau_w$ and $z \neq u$, $z \neq w$.

3. Define the *launch* states $\tau^{L_{split}}$ and $\tau^{L_{merge}}$ by performing the following operations.

   - To obtain $\tau^{L_{split}}$:

     – If $\tau_u = \tau_w$ (i.e. the proposed move is a split,) assign $\tau_u^{L_{split}} = K+1$ and $\tau_w^{L_{split}} = \tau_w$. Note that by $K+1$ we denote a new node that is not yet instantiated. For each $z \in Z$, assign $\tau_z^{L_{split}}$ to either $\tau_u^{L_{split}}$ or $\tau_w^{L_{split}}$ uniformly at random. Sample the new slice of the transition matrix $\tau_{(K+1,\cdot,\cdot)}^{L_{split}}$ according to the restricted conditional prior distribution.

     If $\tau_u \neq \tau_w$ (i.e. the proposed move will be a merge,) simply assign $\tau_u^{L_{split}} = \tau_u$ and $\tau_w^{L_{split}} = \tau_w$. For all indexes $z \in Z$, let $\tau_z^{L_{split}} = \tau_z$.

     – Perform S iterations of the **restricted Gibbs sampler** starting from this state to obtain $\tau^{L_{split}}$. The restricted Gibbs sampler is a method to propose high-quality splits, given the observed data. It is "restricted" because it considers only a subset of transitions on which to operate. Specifically, for each element of

Algorithm 3 restricted-gibbs($k', k'', Z, \omega^j_{1:T}, a^j_{1:T}, S$)

for $s \in \{1, \ldots, S\}$ **do**
  $q \leftarrow 1$
  for $z \in Z$ **do**
    Sample $\bar{k} \sim p(\tau_z = k | \tau_{-z}, \omega^j_{1:T}, a^j_{1:T})$ for $k \in \{k', k''\}$
    $\tau_z \leftarrow \bar{k}$
    $q \leftarrow q \times p(\tau_z = \bar{k} | \tau_{-z}, \omega^j_{1:T}, a^j_{1:T})$
  end for
  for $k \in \{k', k''\}$ **do**
    for $g \in \{1, \ldots, |A|\}$ **do**
      for $g \in \{1, \ldots, |A|\}$ **do**
        $u \leftarrow (k, g, h)$
        Sample $\bar{i} \sim p(\tau_u = i | \tau_{-u}, \omega^j_{1:T}, a^j_{1:T})$ for $i \in \{1, \ldots, K\}$
        $\tau_u \leftarrow \bar{i}$
        $q = q \times p(\tau_u = \bar{i} | \tau_{-u}, \omega^j_{1:T}, a^j_{1:T})$
      end for
    end for
  end for
end for
return τ, q

$z \in Z$, it samples the value of τ_z to be either node τ_w or $(K+1)$ from the conditional distribution of these assignments. Moreover, it samples the destinations for the outgoing transitions of the two nodes above from the current set of nodes $\{1, \ldots, K+1\}$. The details of the restricted Gibbs sampling can be found in Algorithm 3, that is invoked here as `restricted-gibbs`$(\tau, \tau_u, K+1, Z, \omega^j_{1:T}, a^j_{1:T}, S)$.

- To obtain $\tau^{L_{merge}}$:
  - Regardless of whether $\tau_u = \tau_w$ or $\tau_u \neq \tau_w$, assign $\tau_u^{L_{merge}} = \tau_w^{L_{merge}} = \tau_w$. For all indexes $z \in Z$, let $\tau_z^{L_{merge}} = \tau_w$.

– Perform S iterations of the restricted Gibbs sampler starting from this state, obtaining $\tau^{L_{merge}}$, by calling `restricted-gibbs`$(\tau, \tau_w, \tau_w, Z, \omega^j_{1:T}, a_{1:T}, S)$.

4. **Split move.**

- If $\tau_u = \tau_w$, a split move is proposed, by executing one final restricted Gibbs sampling sweep starting from $\tau^{L_{split}}$ to obtain τ^{split}. Note that, as a result of this move (and the intermediate Gibbs sampling sweeps performed in step 3,) the only values that are possibly altered are τ_z, for each $z \in Z$, and the values in the sub-matrices $\tau_{(\tau_u^{split},\cdot,\cdot)}$ and $\tau_{(\tau_w^{split},\cdot,\cdot)}$. Note that $\tau_u^{split} = K + 1$.

- Compute the proposal probability $q(\tau^{split}|\tau)$ as the restricted Gibbs sampling transition kernel from the final iteration. This is the product of the conditional probabilities of the sampled values as they were being sampled during the final sweep (the value `q` returned by Algorithm 3.) Compute the reverse proposal probability $p(\tau|\tau^{split})$ as the Gibbs sampling kernel from a *hypothetical* scan of the restricted Gibbs sampler from $\tau^{L_{merge}}$ to the original τ.

- Compute the Metropolis-Hastings acceptance probability $a(\tau^{split}, \tau)$ and evaluate accordingly. If the proposal is accepted, the state τ^{split} becomes the next state of the Markov chain, otherwise the state τ is left unchanged.

5. **Merge move.**

- If $\tau_u \neq \tau_w$, a merge move is proposed, by executing one final restricted Gibbs sampling sweep starting from $\tau^{L_{merge}}$ to obtain τ^{merge}. Note that in this case, the only values of τ that possibly gets altered are the elements of the sub-matrix $\tau_{(\tau_w^{merge})}$, i.e. the transitions departing from the new merged state.

- Compute the merge proposal probability $q(\tau^{merge}|\tau)$ as the transition kernel of the final scan of restricted Gibbs sampling. Moreover, compute the inverse proposal probability $q(\tau|\tau^{merge})$ as the transition kernel of a *hypothetical* iteration of the restricted Gibbs sampling, going from the split launch state $\tau^{L_{split}}$ to the original state τ.

- Compute the Metropolis-Hastings acceptance probability $a(\tau^{merge}, \tau)$ and evaluate accordingly. If the proposal is accepted, the state τ^{merge} becomes the next state of the Markov chain, otherwise the state τ is left unchanged.

4.2.3 Resampling Hyperparameters

Recall from Section 4.1 that the concentration parameter α and the Dirichlet parameter λ are distributed exponentially with parameters ζ_α and ζ_λ, respectively. We adopt a Metropolis-Hastings procedure in order to resample their values at each iteration of Algorithm 1; the details are described in the following.

4.2.3.1 Concentration Parameter

The concentration parameter α of the stick-breaking process employed by our prior is conditionally independent from all the other variables, given the current number of nodes K of

the PDFC. We use Metropolis-Hastings procedure to sample from the conditional distribution $p(\alpha|K)$. Given the current value α, the new α^* is proposed from a log-normal distribution with mean $\ln(\alpha)$ and unit variance, i.e. $\alpha^* \sim \ln\mathcal{N}(\ln(\alpha), 1)$. This proposal distribution is convenient in that it yields a simple formula for the MH acceptance ratio. By substituting the appropriate densities into Equation 2.14, we have the following derivation:

$$\begin{aligned} a(\alpha^*, \alpha) &= \min\left[1, \frac{q(\alpha|\alpha^*)}{q(\alpha|\alpha^*)}\frac{p(\alpha^*)}{p(\alpha)}\frac{p(K|\alpha^*)}{p(K|\alpha)}\right] \\ &= \min\left[1, \frac{\frac{1}{\alpha}e^{-\frac{1}{2}(\ln\alpha - \ln\alpha^*)^2}}{\frac{1}{\alpha^*}e^{-\frac{1}{2}(\ln\alpha^* - \ln\alpha)^2}}\frac{\zeta_\alpha e^{-\zeta_\alpha \alpha^*}}{\zeta_\alpha e^{-\zeta_\alpha \alpha}}\frac{p(K|\alpha^*)}{p(K|\alpha)}\right] \\ &= \min\left[1, \frac{\alpha^*}{\alpha}\frac{e^{-\zeta_\alpha \alpha^*}}{e^{-\zeta_\alpha \alpha}}\frac{p(K|\alpha^*)}{p(K|\alpha)}\right]. \end{aligned} \tag{4.10}$$

Above, the likelihood terms $p(K|\alpha)$ and $p(K|\alpha^*)$ can be computed from Equation 3.8.

4.2.3.2 Dirichlet Distribution Parameter

The Dirichlet parameter λ is conditionally independent from all the other variables, given the counts d_{kg} of the actions executed in each node, defined in Equation 4.6.

To sample λ, we use the same method as above, obtaining the MH acceptance ratio:

$$a(\lambda^*, \lambda) = \min\left[1, \frac{\lambda^*}{\lambda}\frac{e^{-\zeta_\lambda \lambda^*}}{e^{-\zeta_\lambda \lambda}}\frac{p(d|\lambda^*)}{p(d|\lambda)}\right], \tag{4.11}$$

where the likelihoods $p(d|\lambda)$ and $p(d|\lambda^*)$ are given by Equation 4.7.

4.3 Experimental Results

4.3.1 Description of the Experimental Domains

4.3.1.1 Tiger Problem

The (single-agent) *Tiger Problem* was first introduced in (2), and represents one of the most widely used example domains for POMDPs. Despite its simplicity, it incorporates all the element typical of stochastic, partially observable planning.

In the Tiger Problem, the agent faces two doors. Behind one door is a pot of gold, while the other hides a ferocious tiger. The state of the world is identified by the position of the tiger, i.e. $S = \{TL, TR\}$, corresponding to the tiger being behind the left or the right door, respectively. At each timestep, the agent can perform one of three actions: listen (L), open the door on the left (OL), or open the door on the right (OR), that is, $A = \{L, OL, OR\}$. Listening does not affect the position of the tiger. However, the agent receives upon listening one of two observations: a growl from the left (GL), or a growl from the right (GR), that is, $\Omega = \{GL, GR\}$. These observations are related to the position of the tiger; specifically, the agent perceives a growl from the direction where the tiger is with 0.85 accuracy, and receives a "wrong" signal with 0.15 accuracy. Upon opening either door, the position of the tiger is reset with uniform probability. When opening a door, the agent does not receive informative growls about the position of the tiger, that is, the growls are both received with probability 0.5. Each listening action costs the agent 1. When opening a door, the agent receives a reward of 10 if the door hides the gold, while he gets a penalty of -100 if instead the tiger is found. The description above is summarized in Table I.

TABLE I

SPECIFICATION OF THE TIGER PROBLEM.

Transition function:

	a :	**L**		**OL**		**OR**	
	s :	**TL**	**TR**	**TL**	**TR**	**TL**	**TR**
a' :	**TL**	1	0	0.5	0.5	0.5	0.5
	TR	0	1	0.5	0.5	0.5	0.5

Observation function:

	a :	**L**		**OL**		**OR**	
	s' :	**TL**	**TR**	**TL**	**TR**	**TL**	**TR**
ω :	**GL**	0.85	0.15	0.5	0.5	0.5	0.5
	GR	0.15	0.85	0.5	0.5	0.5	0.5

Reward function:

	a :	**L**	**OL**	**OR**
s :	**TL**	-1	-100	10
	TR	-1	10	-100

Since the Tiger Problem is quite simple, an exact solution can be computed. If we assume that the agent is initially completely uninformed about the position of the tiger (i.e. the initial belief is $b(TL) = 0.5$,) then the optimal infinite horizon policy can be represented by a 5-node deterministic finite-state controller, depicted in Figure 7, where the blue color identifies the initial node. The intuition behind the agent's policy is the following: in order to be confident

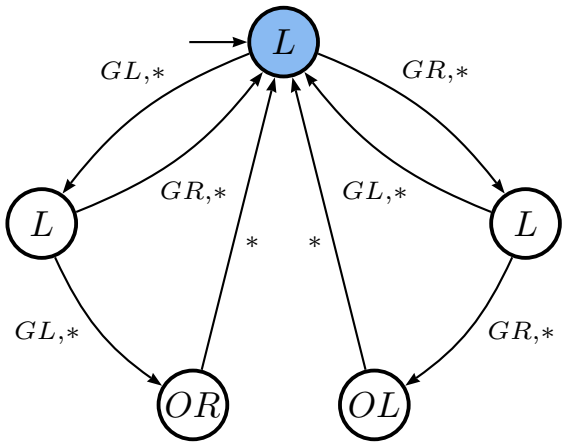

Figure 7. Optimal policy for the single-agent Tiger Problem.

enough about the position of the tiger, the agent listens until the number of growls from one side is at least two more than the growls from the other side.

4.3.1.2 Maze Problem

The *Maze Problem* is an adaptation of the 4×3 world described in (1), and is represented in Figure 8-a. The world is composed of the cells of a 4×3 grid, with one cell blocked. The agent's start position is drawn uniformly at random and the agent is unaware of its own position. The agent can move up, down left, or right at each timestep, i.e. $A = \{U, D, L, R\}$. The agent successfully moves in the intended direction with 0.8 probability, while with 0.1 probability it slips to either the left or the right, as represented in the right of Figure 8-a. When moving against a wall, the agent just remains in the current location. The goal of the agent is to reach the upper right corner of the maze, where it is awarded a reward of 1. Each move costs the agent 0.04. There is a monster lurking in cell 7 that inflicts the agent a penalty of -1. The

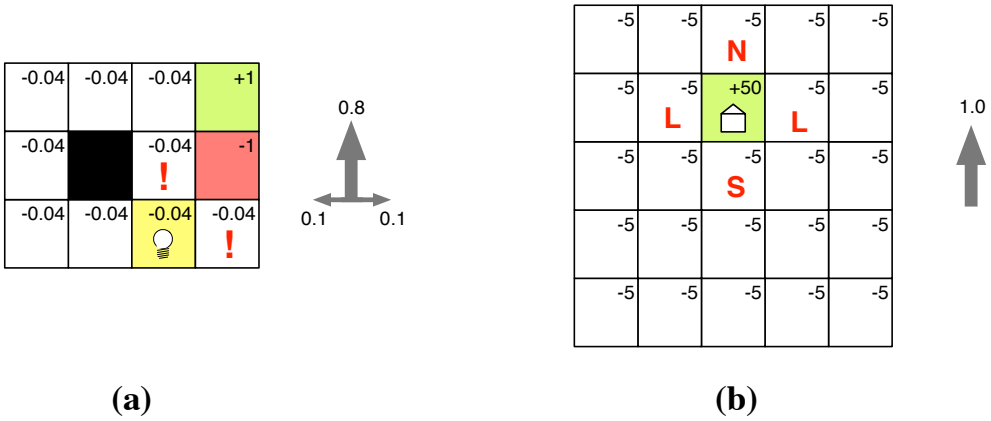

Figure 8. Representation of **(a)** the Maze domain, and **(b)** the AUAV domain.

agent senses the presence of the monster when in one of the two non-goal neighboring locations with probability 0.8. The agent does not perceive the presence of the monster anywhere else. Moreover, the agent knows with probability 1 when reaching cell 10, by perceiving a light that is not perceived anywhere else. After reaching the upper right corner or being caught by the monster, the agent's position is reset at random and the interaction repeats. The agent is notified with a "game over" signal when this happens.

Summarizing, there are $|S| = 11$ possible states of the environment, $|A| = 4$ actions the agent can execute, and $|\Omega| = 4$ possible observations. When solving this problem with a finite-grid approximation (63), we obtain a deterministic finite state controller with $|Q| = 42$ nodes.

4.3.1.3 Fugitive in the AUAV Domain

This problem models a fugitive agent trying to reach a safe house located on a 5×5 grid, depicted in Figure 8-b. In the multiagent version of this problem that we will introduce in

Chapter 7 and is described in (30), an autonomous unmanned aerial vehicle (AUAV) is tasked to intercept the fugitive, hence the name of the domain. As in the previous problem, the agent can move in four directions, but in this case the result of these actions is deterministic, meaning that the agent will always successfully move in the desired location if possible. The agent can sense its position with respect to the safe house, namely whether it is north of it, south of it, or at the same level. These observations are received with accuracy 0.8, and only when the agent is in the proximity of the house, that is, in the four locations adjacent to it. When the agent reaches the safe house, it receives an "end" signal and its position is reset uniformly at random for a new round.

Summarizing, there are $|S| = 25$ states of the world, $|A| = 4$ actions available to the agent, who can receive $|\Omega| = 4$ observations. The controller obtained using the same method as above has in this case $|Q| = 36$ nodes.

4.3.2 Experimental Analysis of MCMC Algorithm

In this section, we provide experimental results about the convergence of the MCMC algorithm through its execution. We consider the value of four variables of interest for a sample run of the algorithm in each of the three domains described above. The variables we consider are:

- The log-likelihood of the PDFC, i.e. $\log\left[p(a^j_{1:T}|\omega^j_{1:T}, \tau, \lambda)\right]$, computed from Equation 4.7, that reflects how well the PDFC fits the observation/action trajectory of agent j.
- The number of instantiated states K.
- The value of the concentration parameter α.

- The value of the Dirichlet distribution parameter λ.

The parameters of Algorithm 1 were set as follows: $T = 512$, $N_{iter} = 5000$, $M = 50$, $R = 50$, $S = 2$, $\zeta_\alpha = \zeta_\lambda = 0.1$. We note that the values of the ζ_α and ζ_λ imply a relatively vague distribution over α and λ, making the learning process more flexible with respect to the number of nodes and entropy of action emission distributions. On the other hand, this additional layer of parameters has the potential of negatively affect the mixing time. In our experiments, however, this has shown to be negligible.

The length of the input trajectories T was set to 512. The trajectories were generated by running a simulation of each POMDP domain with the agent acting as prescribed by the finite state controllers described in Section 4.3.1.

Figure 9 depicts, for each problem, the values observed during a sample run of the algorithm. For all domains, the likelihood of the models progressively increases and then stabilizes (first row of plots, in red,) confirming the expectation that the MCMC sampler moves towards regions of the sample space with high posterior probability, where the Markov chain reaches its stationary regime. The number of nodes K (second row, in blue) varies through the execution, hence demonstrating the ability of the proposed Bayesian methodology to consider PDFCs of different size. The third rows of plots (in green) shows the value of the concentration parameter α throughout the iterations of the algorithm. Specifically, the grey line shows the actual values, while the bold green line is the smoothed value, that makes it easier to see trends. We can see that the value of α follows the increase and decrease in the number of nodes K in the plots above, which is expected given the positive correlation between α and K from Equation 3.8.

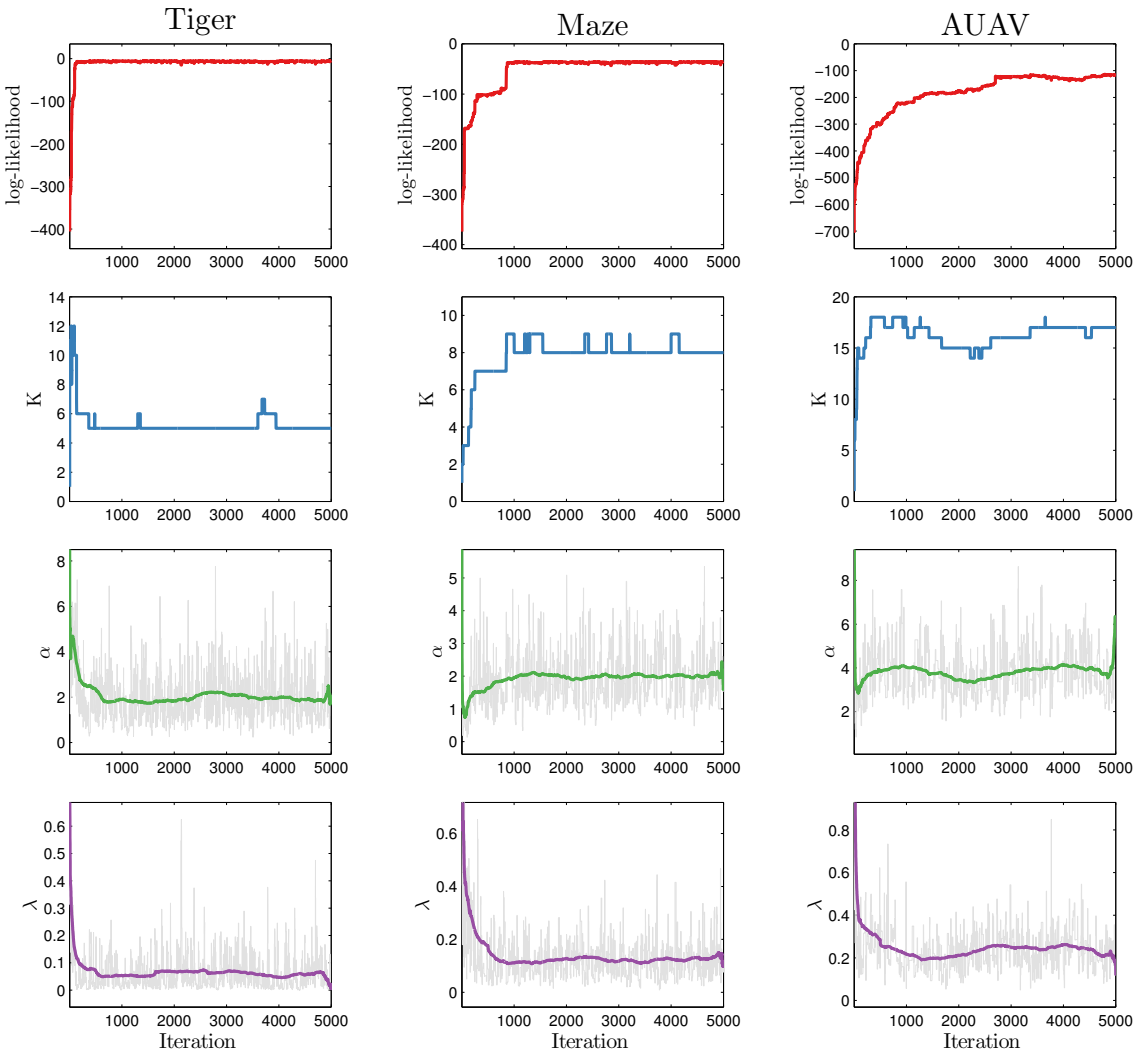

Figure 9. Log-likelihood of τ and value of other state variables during a sample run of Algorithm 1 for the three experimental domains.

The last row (purple) of plots shows the value of the Dirichlet parameter λ, which quickly decreases as the inference goes on. This result is meaningful: since we are using trajectories generated by deterministic controllers, the value of λ adapts to favor PDFCs with low entropy action generation distributions θ_k.

4.3.3 Convergence to the True Controllers

In this section we evaluate how "similar" the PDFCs inferred by Algorithm 1 are to the true controllers generating the data. Defining a measure of distance between two PDFCs is not straightforward. Here, similarity is measured with respect to the predictive action distribution, that is, how close the predictive probability over the next action of agent j is to the true probability. Computing the predictive probability can be formalized as a filtering task: given a PDFC c, our current distribution over the nodes of the controller $p(q_t)$, the action that j executes a_t^j, and the observation that j receives ω_t^j, the predictive distribution over the next action is:

$$p(a_{t+1}^j | p(q_t), a_t^j, \omega_{t+1}^j, c) = \sum_{q_t} p(q_t) \sum_{q_{t+1}} \delta_K(q_{t+1}, \tau_{q_t a_t^j \omega_{t+1}^j}) \, \theta_{q_{t+1} a_{t+1}^j} \quad . \tag{4.12}$$

4.3.3.1 Weighted Kullbak-Leibler Divergence Between two PDFCs

To derive a well-defined accuracy measure for the predictive probability, we build on the following intuitive idea: suppose that an agent operates according to the true controller c_T, and a second hypothetical agent does so according to the learned controller c_L. The environment only responds to the action of the first agent (the true one.) In this setting, let us denote as

$\eta_{q_L q_T}$ the co-frequency, or probability of the two agents being simultaneously in nodes q_L and q_T of the respective controllers. The co-frequency can be computed as the stationary distribution of a Markov chain whose transition function is the composition of the controllers' parameters and environment's dynamics. The details of this computation are provided in Appendix A.

We then define the **weighted Kullback-Leibler divergence** (wKL) of the learned controller c_L from the true controller c_T as:

$$wKL(c_L, c_T) = \sum_{(q_L, q_T) \in Q_L \times Q_T} \eta_{q_L q_T} \; KL(\theta_{q_L}, \theta_{q_T}), \tag{4.13}$$

where KL is the Kullback-Leibler divergence between two finite discrete distributions p and q over the same set, i.e.

$$KL(p, q) = \sum_i p(i) \log \frac{p(i)}{q(i)}. \tag{4.14}$$

4.3.3.2 Results

The following results show the values of the weighted KL divergence of the learned PDFCs from the true one, with respect to the length of the observation sequence T_{learn}. For each of the three domains and for each of the considered values of T_{learn}, 10 learning trials were performed. Each learning trial consists of generating a trajectory $(\omega^j_{1:T_{learn}}, a^j_{1:T_{learn}})$ from a simulator and calling Algorithm 1 on such sequence. The parameters of the MCMC sampler were set as above, that is, $N_{iter} = 5000$, $M = 50$, $R = 50$, $S = 2$, $\zeta_\alpha = \zeta_\lambda = 0.1$. In each trial, the second halves of the generated sample chains were subsampled every 100 iterations, resulting in ensembles of

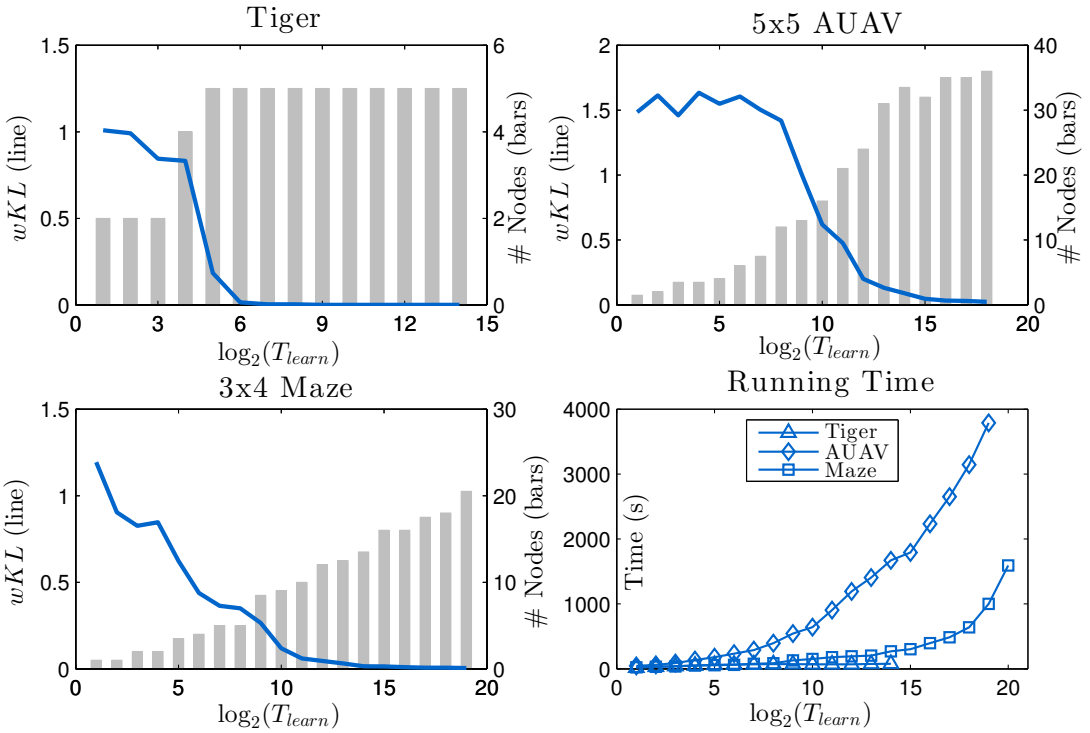

Figure 10. Weighted KL distance between the learned and the true controllers (line) and number of nodes of learned PDFC (bars). The fourth panel reports the timing results.

25 PDFCs per trial. We computed the overall mean wKL across trials, which is reported in Figure 10 along with the median size of the learned PDFCs.

For the Tiger Problem, we see that the wKL quickly approaches zero ($T_{learn} \geq 64$) and the number of nodes stabilizes at 5, the size of the true controller. This is evidence of the fact that for this simple case, the structure of the true controller is retrieved and the emission probability of the nodes converge to the action generation functions of the true controller.

For the other two problems, the wKL decreases more gradually, eventually converging towards zero. For the AUAV problem, the number of nodes approaches the size of the true controller (36 nodes) for long sequences. In the Maze problem, the size of the PDFCs grows steadily but remains lower than the true size (42 nodes), even when the wKL approaches zero. While it seems that for this problem we may need an impossibly long sequence to eventually learn the true number of nodes, the learned PDFCs are behaviorally very close to j's true controller.

In order to shed some light over this result, Figure 11 reports the proportion of time spent in each node of the true controller in the limit, sorted in decreasing order. This frequency can be computed as the marginalized stationary distribution of the Markov chain over $S \times Q$, whose transitions are given by the combined dynamics of the POMDP domain and the PDFC.

We can see that the distribution decreases rapidly for the Maze problem, with less than 1% of the time spent in more than 50% of the nodes. This means that most of the complexity of the observed agent's behavior can be captured with fewer nodes. For the AUAV problem, the distribution decreases more gradually, meaning that more nodes are required to accurately describe the observed behavior. The relation between node occupancy and convergence to the true controller is important to establish theoretical properties of learning, and will be explored more in depth in future work.

As a further argument that, even if with fewer nodes, the PDFCs learned for the Maze domain resemble the true controller, Figure 12 provides a co-frequency graph in which the set of nodes on the left are the nodes of the true controller, and the nodes on the right belong to

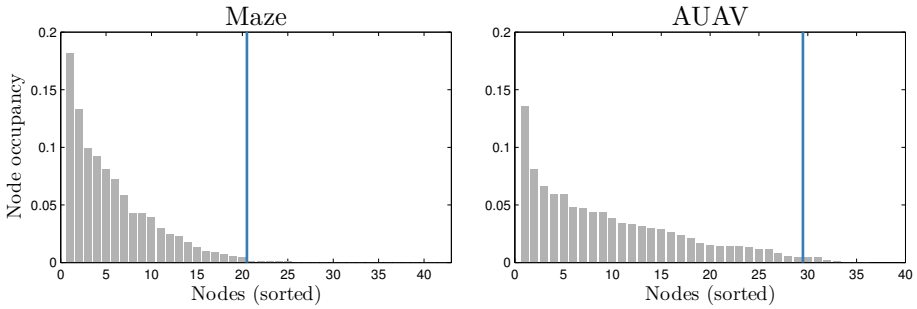

Figure 11. Node occupancy for Maze and AUAV problems. The vertical line indicates the 99% percentile.

a PDFC of size 16, learned with a training sequence of length $T_{learn} = 2^{12}$. Of the 42 nodes of the true controller, only 16 are represented explicitly, while all the others are collapsed into one meta-node, represented as a grey rectangle in the graph, which accounts for only 3.71% of total occupancy. The thickness of each edge (q_T, q_L) is proportional to the co-frequency $\eta_{q_L q_L}$. There is almost an exact one-to-one relation between the two sets of nodes, which is especially true for the nodes that are visited frequently. We can deduce that the smaller PDFCs that the algorithm learns capture most of the relevant topology of the true controller, even with a much lower number of nodes.

The bottom-right panel of Figure 10 reports the running time of the MCMC algorithm[1], which is at most linear in the amount of data considered. Notice that the growth rate for the AUAV problem is almost constant for large values of T_{learn}, while it increases for the Maze problem, indicating higher dependence on the PDFC's size than on T_{learn}. This is due to

[1]Implemented in MATLAB® and running on an Intel® Xeon® 2.27 GHz processor.

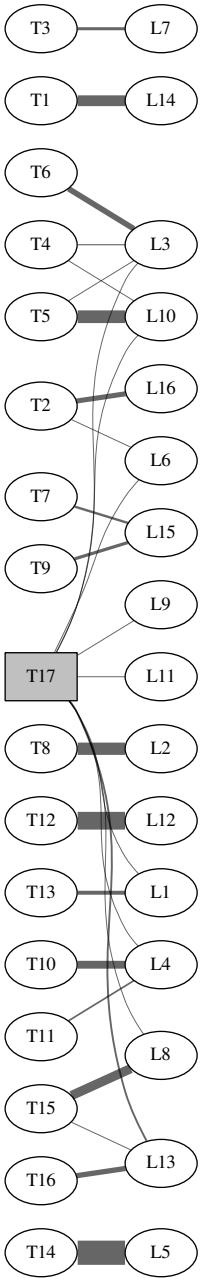

Figure 12. Co-frequency between nodes of the true controller for the Maze domain (left) and the learned PDFC (right).

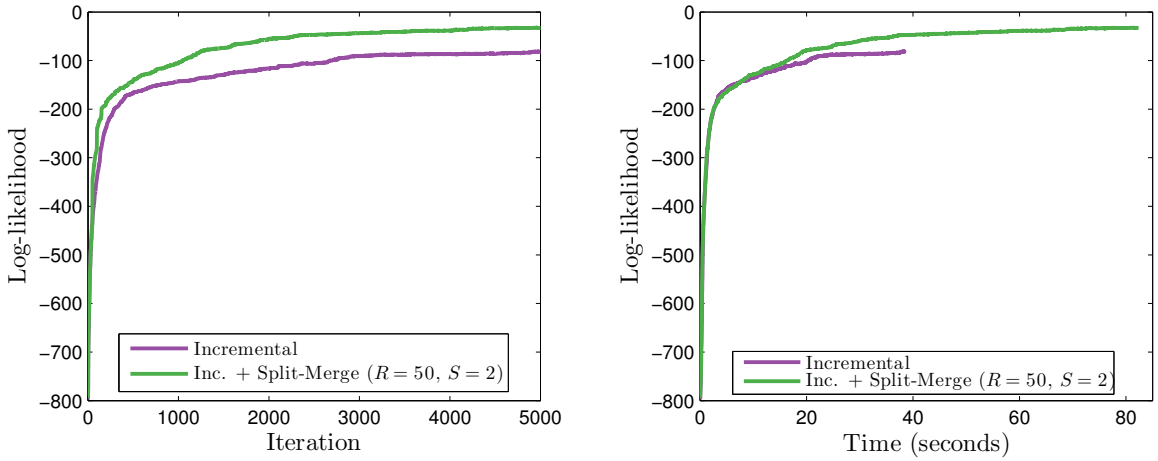

Figure 13. MCMC sampling log-likelihood when split-merge moves are used and when they are not, with respect to MCMC iteration (left) and c0mputation time (right).

optimizations in the computation of the quantities d used in Equation 4.7, which contains the only dependency of running time on T_{learn}.

4.3.4 Benefit of Split-Merge Moves

In order to show that split-merge moves increase the quality of the learning with respect to a purely incremental Gibbs sampling approach, we compare the log-likelihood obtained using the two methods in the Maze domain. 50 action/observation trajectories of length $T_{learn} = 2^{10}$ were generated, and for each such sequence Algorithm 1 was run both with parameters ($R = 50, S = 2$) as above, and with the split-merge procedure disabled. All other parameters were set as in the previous experiments. Figure 13-left shows the average log-likelihood throughout the execution of the algorithm for both cases. It is clear that using split-merge moves leads to better results in terms of mixing speed with respect to the number of iterations. However, splitting and

merging nodes is computationally intensive and makes the algorithm slower. For this reason, Figure 13-right reports the log-likelihood with respect to execution time (in seconds.) The plots show that, even when considering the actual computation time, using split-merge moves delivers superior results.

CHAPTER 5

PDFC INFERENCE FROM PARTIALLY OBSERVED BEHAVIOR

5.1 Learning with Partial Observability

In general, agent i may not be able to perfectly observe the trajectory of observations and actions $(\omega^j_{1:T}, a^j_{1:T})$ of agent j, and instead just receive its own sequence of observations from the environment, $\omega^i_{1:T}$. Moreover, agent i may not be just a passive observer. Its own actions need to be taken into consideration when learning j's model since they can affect j's behavior.

The sequences of world states $s_{1:T}$, actions $a^j_{1:T}$, and j's observations $\omega^j_{1:T}$ are related to i's received sequence of observations via the world's transition and both agent's observation functions, which are here assumed to be known. We hence need to infer the posterior distribution:

$$p(\tau, \theta | \omega^i_{1:T}, a^i_{1:T}) = \frac{p(\omega^i_{1:T} | \tau, \theta, a^i_{1:T})\ p(\tau, \theta)}{p(\omega^i_{1:T} | a^i_{1:T})}. \tag{5.1}$$

The extended graphical model that represents the learning scenario in this case is depicted in Figure 14. We place the same prior as before on the variables describing j's PDFC, and the conditional distributions in Equation 4.2 and Equation 4.3 still apply to this model. The

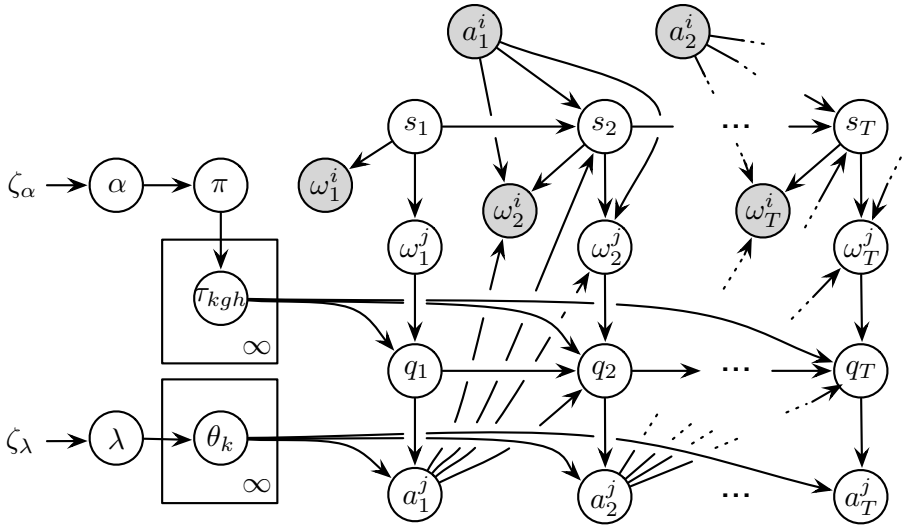

Figure 14. Graphical model representation of PDFC learning setup under partial observability of j's behavior.

conditional distributions involving the new variables are given by the dynamics of I-POMDP model:

$$\begin{aligned} s_t \mid s_{t-1}, a^i_{t-1}, a^j_{t-1} &\sim T(s_t, a^i_t, a^j_t, s_{t+1}) && t = 2, \ldots, T \\ \omega^i_t \mid a^i_{t-1}, a^j_{t-1}, s_t &\sim O^i(a^i_{t-1}, a^j_{t-1}, s_t, \omega^i_{t+1}) && t = 2, \ldots, T \\ \omega^j_t \mid a^i_{t-1}, a^j_{t-1}, s_t &\sim O^j(a^i_{t-1}, a^j_{t-1}, s_t, \omega^j_{t+1}) && t = 2, \ldots, T. \end{aligned} \tag{5.2}$$

In order to extend the MCMC algorithm presented in Chapter 4 to the partially observable setting, we treat the sequences $s_{1:T}, a^j_{1:T}, \omega^j_{1:T}$ as hidden variables, whose values also need to be inferred along with the PDFC's parameters. Therefore, we extend the state of the MCMC algorithm to be $(\tau, s_{1:T}, a^j_{1:T}, \omega^j_{1:T}\lambda, \alpha)^{(n)}$, where as before (n) indicates the n-th iteration of the algorithm. Accordingly, we add one step to the algorithm, that samples the value of the

Algorithm 4 LearnPDFC-PO

Input: $\omega^i_{1:T}, a^i_{1:T}, M, R, S, N_{iter}$
Output: $\tau^{(1:N_{iter})}, \alpha^{(1:N_{iter})}, \lambda^{(1:N_{iter})}$

▷ Initialize PDFC

$\tau_0^{(1)} \leftarrow 1$
$\tau_{1gh}^{(1)} \leftarrow 1 \quad \forall\, g = 1..|A^j|, h = 1..|\Omega^j|$
$\alpha^{(1)} \sim exp(\zeta_\alpha)$
$\lambda^{(1)} \sim exp(\zeta_\lambda)$

▷ Initialize hidden sequences

$(s_{1:T}, a^j_{1:T}, \omega^j_{1:T})^{(1)} \leftarrow$ sample-seq$(\tau, \lambda, \omega^i_{1:T}, a^i_{1:T})$

▷ MCMC iterations

for $n = 2..N_{iter}$ **do**
  if mod $(n, R) \neq 0$ **then**
    $\tau^{(n)} \leftarrow$ incremental-move$(\tau^{(n-1)}, \alpha^{(n-1)}, \lambda^{(n-1)}, (a^j_{1:T}, \omega^j_{1:T})^{(n)}, M)$
  else
    $\tau^{(n)} \leftarrow$ split-merge$(\tau^{(n-1)}, \alpha^{(n-1)}, \lambda^{(n-1)}, (a^j_{1:T}, \omega^j_{1:T})^{(n)}, S)$
  end if
  $(s_{1:T}, a^j_{1:T}, \omega^j_{1:T})^{(n)} \leftarrow$ sample-seq$(\tau^{(n)}, \lambda^{(n)}, \omega^i_{1:T}, a^i_{1:T})$
  $(\alpha^{(n)}, \lambda^{(n)}) \leftarrow$ sample-hyperpars$(\tau^{(n)}, \alpha^{(n-1)}, \lambda^{(n-1)}, (a^j_{1:T}, \omega^j_{1:T})^{(n)})$
end for

three hidden sequences given the current values of all other variables, as detailed in Section 5.1.1. The overall structure of the MCMC algorithm for the partially observable case is listed in Algorithm 4.

5.1.1 Sampling Hidden Sequences

An easy way to sample the sequence $s_{1:T}$, $\omega^j_{1:T}$, $a^j_{1:T}$ would be to use a Gibbs sampling schema applied to each individual variable, given the value of all the others. Although simple to implement, this method would lead to poor mixing since it does not allow more than one variable to change at the same time, and the amount of variables in these sequences can be

large. We can instead sample each entire sequence in block, by adapting similar methods used for hidden Markov models (64). It is actually possible to take this approach further, and sampling the three sequences *together* and in block, given the current values of the other variables. In order to do this, we propose a backward filtering, forward sampling procedure (BFFS), described in the following.

We begin by noticing that at each timestep the value of the variables (s_t, q_t) represents a sufficient statistics, that is, all past history of all other variables can be summarized with just these two. This is because (s_t, q_t) make all future variables conditionally independent from past variables, as evinced from the structure of the Bayesian network in Figure 14. Therefore, by applying concatenation and Bayes rule, the following holds for each $1 < t \leq T$:

$$\begin{aligned} & p(a^j_{t-1}, s_t, \omega^j_t | s_{t-1}, q_{t-1}, a^i_{t-1:T}, \omega^i_{t:T}, \tau) \\ & \quad \propto p(a^j_{t-1}|q_{t-1})\; p(s_t|s_{t-1}, a^i_{t-1}, a^j_{t-1})\; p(\omega^j_t|a^i_{t-1}, a^j_{t-1}, s_t) \\ & \quad\quad \times p(\omega^i_t|a^i_{t-1}, a^j_{t-1}, s_t)\; p(\omega^i_{t+1:T}|s_t, q_t = \tau_{q_{t-1}a^j_{t-1}\omega^j_t}, a^i_{t:T}). \end{aligned} \tag{5.3}$$

All but the last term of the RHS in the equation above are known from the conditional probabilities of the Bayesian network. The last term can be computed for each time step using a variation of the backward portion of the forward-backward algorithm for HMMs (64): it can be interpreted as a *backward probability message*, and represents the probability of future

observations given the sufficient statistics at time t, denoted as $\xi_t(s_t, q_t)$. This quantity can be computed for each $1 \leq t \leq T$ using the following recursive relation:

$$\begin{aligned}
\xi_t(s_t, q_t) &= p(\omega^i_{t+1:T} | s_t, q_t, a^i_{t:T}) \\
&= \sum_{a^j_t} p(a^j_t | q_t) \sum_{s_{t+1}} p(s_{t+1} | s_t, a^i_t, a^j_t)\ p(\omega^i_{t+1} | a^i_t, a^j_t, s_{t+1}) \\
&\qquad \times \sum_{\omega^j_{t+1}} p(\omega^j_{t+1} | a^i_t, a^j_t, s_{t+1})\ p(\omega^i_{t+2:T} | s_{t+1}, q_{t+1} = \tau_{q_t a^j_t \omega^j_{t+1}}, a^i_{t+1:T}) \\
&= \sum_{a^j_t} p(a^j_t | q_t) \sum_{s_{t+1}} p(s_{t+1} | s_t, a^i_t, a^j_t)\ p(\omega^i_{t+1} | a^i_t, a^j_t, s_{t+1}) \\
&\qquad \times \sum_{\omega^j_{t+1}} p(\omega^j_{t+1} | a^i_t, a^j_t, s_{t+1})\ \xi_{t+1}(s_{t+1}, \tau_{q_t a^j_t \omega^j_{t+1}}).
\end{aligned} \tag{5.4}$$

The computation starts at $\xi_T(\cdot, \cdot) = 1$ and proceeds backwards down to $t = 1$. Once $\xi_{1:T}$ is computed, the new values of $a_{1:T}$, $s_{1:T}$, and $\omega^j_{1:T}$ can be sampled using Equation 5.3, moving forward from $t = 1$ to T.

5.2 Experimental Results

5.2.1 Multiagent Extension of Experimental Domains

The experimental domains introduced in Section 4.3.1 are here extended to their multiagent versions, whose details are described in the following sections.

5.2.1.1 Multiagent Tiger Problem

We consider the two-agent extension of the Tiger Problem (Section 4.3.1.1) as introduced in (6). There are two agents facing two doors. As before, one door hides a tiger and the other hides gold. The reward function of the two agents is the same as before: if an agent opens the

door that leads to the gold, it will be rewarded 10, while finding the tiger yields a penalty of 100, and each listening action costs 1. Also, as before, each agent can hear the tiger's growl with 0.85 accuracy. Additionally, upon listening, the agents hear a creak from the direction of the door that the other agent opens, or just silence if the other agent is also listening. These creaks (or lack thereof) are perceived with 0.9 accuracy. This configuration leads to an observation set for the two agents of cardinality $|\Omega^i| = |\Omega^j| = 6$. The transition function is such that the position of the tiger is reset uniformly when either agent opens the door. The formal specification of this problem is provided in Table II.

5.2.1.2 Multiagent Maze Problem

We extend the Maze domain described in Section 4.3.1.2 to a setting with two agents. We assume that agent j is the agent described in the single-agent setting, while agent i is the monster that was lurking in cell 7 of the original problem. Agent i has the same set of actions and the same moving ability of agent j, that is, it moves in the intended direction with 0.8 probability, and slips off the either side with 0.1 probability. The observation function of agent j is the same as the one described for the single-agent setting. Agent i, instead, is able to observe its own position exactly, and perceives j's position with 0.9 accuracy. Note that, even though the location of the agent j is perfectly observable by agent i, its actions are not. The goal of agent i is to chase agent j and prevent it from reaching the upper-right corner. Specifically, agent i receives a reward of 1 when intercepting agent j, and each step costs 0.04.

TABLE II

SPECIFICATION OF THE MULTIAGENT TIGER PROBLEM. THE OBSERVATION FUNCTION IS FACTORED INTO *GROWLS* AND *CREAKS*.

Transition function

(a_i, a_j)	s	**TL**	**TR**
(L, L)	TL	1	0
(L, L)	TR	0	1
$(OL, *)$	$*$	0.5	0.5
$(OR, *)$	$*$	0.5	0.5
$(*, OL)$	$*$	0.5	0.5
$(*, OR)$	$*$	0.5	0.5

i's reward function

(a_i, a_j)	**TL**	**TR**
$(L, *)$	-1	-1
(OL, L)	-100	10
(OL, OL)	-100	10
(OL, OR)	-100	10
(OR, L)	10	-100
(OR, OL)	10	-100
(OR, OR)	10	-100

j's reward function

(a_i, a_j)	**TL**	**TR**
$(*, L)$	-1	-1
(L, OL)	-100	10
(OL, OL)	-100	10
(OR, OL)	-100	10
(L, OR)	10	-100
(OL, OR)	10	-100
(OR, OR)	10	-100

i's observation function

GROWLS

(a_i, a_j)	s	GL	GR
$(L, *)$	TL	0.85	0.15
$(L, *)$	TR	0.15	0.85
$(OL, *)$	$*$	0.5	0.5
$(OR, *)$	$*$	0.5	0.5

CREAKS

(a_i, a_j)	s	S	CL	CR
$(*, L)$	$*$	0.9	0.05	0.05
$(*, OL)$	$*$	0.05	0.9	0.05
$(*, OR)$	$*$	0.05	0.05	0.9
$(OL, *)$	$*$	1/3	1/3	1/3
$(OR, *)$	$*$	1/3	1/3	1/3

j's observation function

GROWLS

(a_i, a_j)	s	**GL**	**GR**
$(*, L)$	TL	0.85	0.15
$(*, L)$	TR	0.15	0.85
$(*, OL)$	$*$	0.5	0.5
$(*, OR)$	$*$	0.5	0.5

CREAKS

(a_i, a_j)	s	**S**	**CL**	**CR**
$(L, *)$	$*$	0.9	0.05	0.05
$(OL, *)$	$*$	0.05	0.9	0.05
$(OR, *)$	$*$	0.05	0.05	0.9

5.2.1.3 AUAV Reconnaissance Problem

The POMDP problem described in Section 4.3.1.3 is here extended to two agents, of which agent j is the fugitive, and agent i is an autonomous unmanned aerial vehicle (AUAV) that is tasked to locate the fugitive. This setup is an adaptation of the AUAV domain described in (30). The observation and reward functions of the fugitive agent are as before. The AUAV, who like the fugitive moves deterministically in all four directions, can observe the location of the fugitive with 0.9 accuracy. If the AUAV is successful in intercepting the fugitive, it receives a reward of 50, while each step has a cost of -5.

5.2.2 Observational Kullback-Leibler Divergence

In order to evaluate the performance of the MCMC algorithm in the partially observable case, we consider a metric based on how closely the learned PDFCs can predict the next *observation of agent* i, ω^i_{t+1}, at each time t. The weighted KL divergence of action probabilities that we considered in Section 4.3.3.1 is not a fair metric to use in the partially observable setting since j's actions $a^j_{1:T}$ are not provided as input to the learning algorithm. While a low value of wKL would also imply that the probability induced by the learned PDFC over agent i's observations resembles the probability obtained with the true controller, the opposite is in general not true. In other words, while two models of agent j might not be *behaviorally* equivalent, they could still be *observationally* equivalent from the perspective of agent i.

We hence derive a quantitative measure of observational similarity between two models. Recall that wKL was defined in terms of node co-location frequency and KL divergence between action distributions. In the partially observable case, computing the nodes' co-location is much

more complicated, since the composed transition probability between any two pair of nodes would have to incorporate i's belief update about the interactive space $Q \times S$. Instead, we derive an empirical measure of observational distance between two controllers. Given a PDFC c, let us define the predictive observational distribution at each time t, given agent i's current belief, as:

$$\begin{aligned} p^c_{t,obs} &= p^c(\omega^i_{t+1}|b_t(s_t,q_t),a^i_t) \\ &= \sum_{(s_t,q_t)} b_t(s_t,q_t) \sum_{a^j_t} \theta^c_{q_t}(a^j_t|q_t) \sum_{s_{t+1}} p(s_{t+1}|s_t,a^i_t,a^j_t)\; p(\omega^i_{t+1}|a^i_t,a^j_t,s_{t+1}), \end{aligned} \tag{5.5}$$

where θ^c_q is the action distribution of node q in PDFC c. This quantity is agent i's belief over what observation it will receive next. Given a test trajectory of length T_{test} generated by the true controller C_T, the **mean observational KL divergence** of a learned controller c_L from c_T is defined as the empirical mean across the test sequence of the KL divergence of predictive observation probability.

$$\bar{KL}_{obs}(c_L,c_T) = \sum_{t=1}^{T_{test}} KL(p^{c_L}_{t,obs}, p^{c_T}_{t,obs}). \tag{5.6}$$

We use this metric to evaluate the learning performance in the following section.

5.2.3 Results

For the three problems domains described above, the training sequences were generated by simulating the POMDP environment without any interaction of agent i, and with the observed agents behaving as prescribed by their optimal controllers as in 4.3.3.1, that is, assuming that

agent j is oblivious to the presence of agent i. This assumption will be relaxed in the context of online learning (Chapter 7).

Algorithm 4 was executed on input sequences of varying lengths, namely $T_{learn} = \{2^4, 2^6, 2^8, 2^{10}\}$ for the Tiger Problems, and $T_{learn} = \{2^8, 2^{10}, 2^{12}, 2^{14}\}$ for the Maze and AUAV domains. For each such values, 20 sequences $\omega^i_{1:T_{learn}}$ were simulated and a run of the MCMC sampler was executed on each of those. The parameters of the algorithm were set as in the fully observable case, that is, $N_{iter} = 5000$, $M = 50$, $R = 50$, $S = 2$, $\zeta_\alpha = \zeta_\lambda = 0.1$, and 25 PDFCs were sampled at regular intervals from the second halves of such runs. The learned PDFCs were then evaluated according to the metric introduced in the previous section, computed over a sequence of length $T_{test} = 2^{10}$.

Figure 15 reports the mean value and 95% confidence intervals of the observational KL divergence for each problem and sequence length, while Figure 16 reports the median size of the learned PDFCs. As expected, $\hat{KL}_{obs}$ generally decreases as more data is used to train the PDFCs. We observe a slight increase in the last value for the Tiger Problem, which is nevertheless still within the confidence interval of the previous value. The dashed horizontal lines represent the KL divergence that is obtained from a completely random model i.e. a PDFC with one node having uniform probability over $|A^j|$. For all three domains, the KL divergence obtained by training the PDFCs on longer sequences dips to only a small fraction of such value (9.17%, 14.23%, and 17.09% respectively for the three problems,) indicating that the learned models indeed capture important patterns in agent j's behavior, at least as far as revealed by i's observation sequences. For the Maze domain, $\hat{KL}_{obs}$ is very low even when trained on

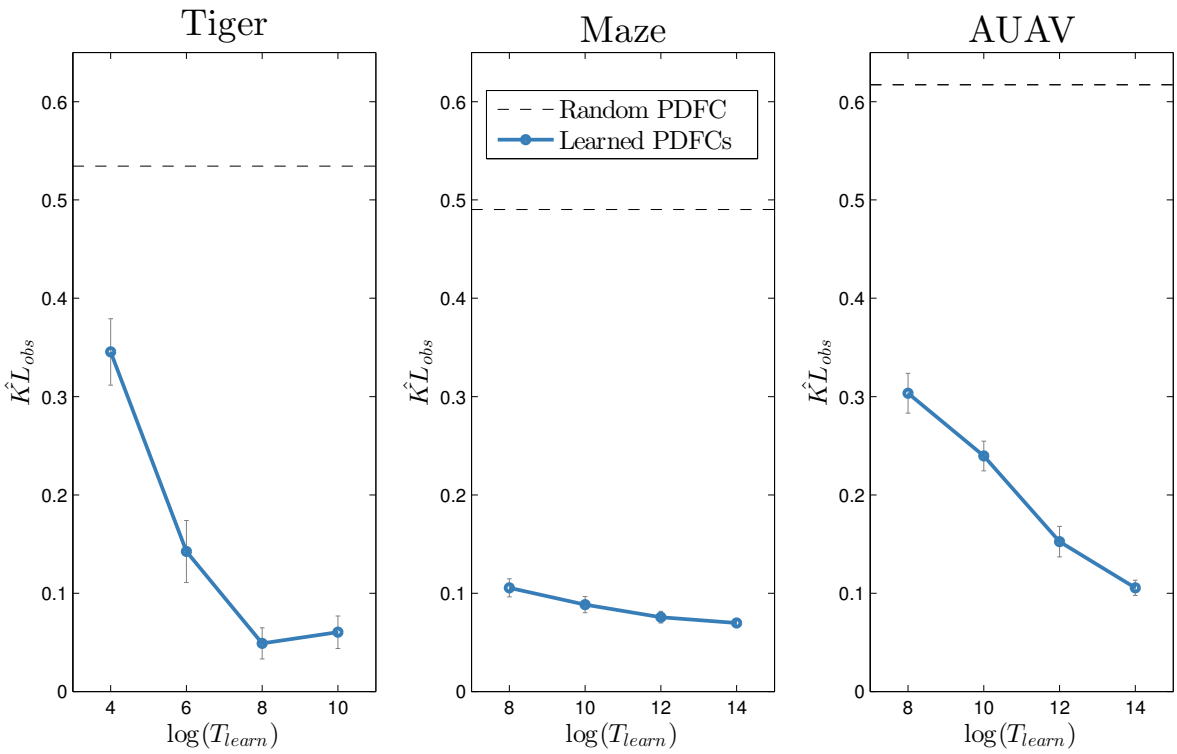

Figure 15. KL divergence of predictive observation probability of learned PDFCs from true controllers: mean (line) and 95% confidence interval (vertical bars.)

relatively short sequences of 256 steps; at the same time, the size of the learned PDFCs is small with respect to the 42 nodes of the true controller. This is in line with the results reported for the fully observable case (Figure 10), and indicates that the behavioral patterns of agent j are actually quite simple. This is a desirable feature of the proposed nonparametric Bayesian learning algorithm, that is able to adjust the size of the learned controllers to the complexity of observed behavior.

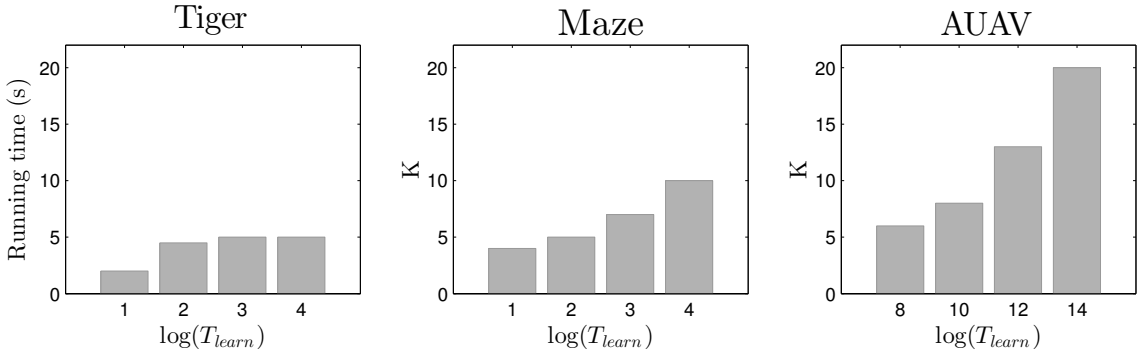

Figure 16. Median number of nodes of the learned PDFC in the partially observable case

The median size of the inferred PDFCs is shown in Figure 16, for each problem and length of training sequence. As expected, the number of nodes in the learned PDFCs grows with the amount of training data. However, the PDFCs learned in this settings are usually smaller than the ones learned with perfect observability of j's behavior. This is because, with partial observability, i's perception of j's behavior is filtered via the noisy world's dynamics, and therefore longer observations are needed to allow identification of the same behavioral patterns.

Figure 17 reports the running time of the algorithm for each domain and length of training sequence. With respect to the fully observable case, the additional sequence sampling step in Algorithm 4 makes the procedure more time consuming. Unfortunately, this sampling is an iterative procedure that cannot be vectorized or optimized algorithmically. However, other choices of implementation language can make the procedure much faster, and preliminary results have shown no noticeable loss in performance if sequences are sampled less frequently than every iteration. Moreover, the backward filtering in Equation 5.4 can be approximated via a particle

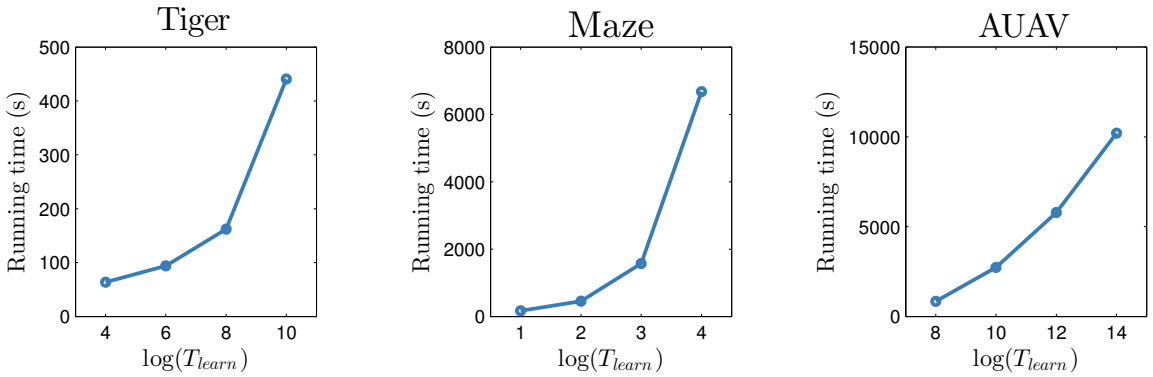

Figure 17. Running time of Algorithm 1

filter, making the execution time independent from the size of the problem. The exploration of faster sequence sampling methods is delegated to future work.

CHAPTER 6

PLANNING AGAINST LEARNED PDFCS

6.1 Best-Response Agents

While the previous chapters are focused on learning probabilistic deterministic finite-state controllers of another agent, this chapter is dedicated to the use of the learned models in the context of multiagent autonomous stochastic planning. In particular, we consider in this chapter a two-phase scenario, in which agent i first collects data about the behavior of agent j, learns a set of candidate PDFCs, and then exploits the learned models in its own decision making process. This approach works under the assumption that j's model is static, i.e. does not change throughout its interaction with the environment, and in particular remains the same from the *observation* phase into the *interaction* phase. This implies that j is oblivious of i's presence, or that j plans against a model of i that is given a priori. These assumptions are dropped in Chapter 7, where the two agents interact and simultaneously learn about each other.

Note that we use the word "against" without necessarily implying an adversarial stance between the two agents. Instead, what we mean is that agent i aims at finding its **best response** with respect to its subjective beliefs about the other agent's model, as in the case of interactive POMDPs (Section 2.3). The next section provides the details on how the learned PDFCs of agent j are embedded into agent i's decision making process.

6.1.1 Subintentional I-POMDPs

The interactive POMDP framework is general enough to accommodate different kinds of models of other agents. As described in Section 2.3, one possibility are intentional models: the other agents' beliefs and preferences are modeled explicitly, and agent i's best response with respect to its beliefs requires to recursively solve the I-POMDP models of the other agents. As noted before, this *recursive modeling* yields to a hierarchy of nested beliefs, whose depth must be limited to a finite level in order to make the approach implementable. Even so, the computational effort needed to solve intentional I-POMDPs is in general very high.

Instead of using intentional models of other agents[1], we specialize the generic I-POMDP framework by considering only subintentional models. As mentioned before, a subintentional model is any stochastic process that takes as input a sequence of observation symbols $\omega^j_{1:T}$ belonging to the set Ω^j and induces a probability distribution over the sequence of actions $a^j_{1:T}$ from the set A^j. There is no concept of agenthood in such description: the other agents' preferences and rationality criterion are not modeled. The focus is on prediction rather than explanation; as such, the choice of modeling other agents as either intentional or subintentional can be viewed in light of the general discussion about explanatory and predictive models in statistics and artificial intelligence (see for example (65)).

---

[1]Strictly speaking, the intentional I-POMDP formalization (6) considers subintentional models side by side with intentional models. However, how to obtain the set of possible subintentional models or how to update them is not explicitly discussed.

Without getting into the merits of this general debate, we highlight in the following some pros and cons of the two approaches in the context of I-POMDPs:

- **Complexity.** Using subintentional models is arguably less computationally intensive than recursively solving other agent's intentional models.
- **Prior assumptions.** Despite the intentional I-POMDP formulation is in theory very flexible on what prior knowledge is available to the agent, practical implementations always assume knowledge about the structure of the other agent (its *frame*,) and in particular its reward function, or limit the set of possible frames to a small finite set. The subintentional I-POMDP methodology that we propose, on the other hand, assumes no knowledge about the other agent's payoff structure, and about what transition function the other agent considers; however, we assume in this work that its observation function is known. Moreover, intentional I-POMDPs assume that the other agent is rational. There is no such assumption when using subintentional models.
- **Use of prior knowledge.** If the true frame of the agent is indeed known, the subintentional approach is wasteful in that it does not exploit it. Nonetheless, the subintentional approach described in this work was explicitly designed to cope with situations where little prior knowledge is given, and does not necessarily represent best approach to all possible scenarios.
- **Transferable knowledge.** If the frame of the other agent is not fixed, intentional modeling may provide insight into its reward function by means of belief update. Since an agent preferences may carry over from one domain to another, this constitutes transferable

knowledge that can be used in situations different than the one in which the knowledge was obtained. On the other hand, subintentional models are more intricately tied to a particular domain, and are only valid insofar the environment (or more precisely, the modeled agent's belief about the environment) remains unchanged. Switching to a new domain requires the other agent's models to be re-learned from scratch. However, it might be possible to speculate about an agent's preferences starting from its inferred subintentional model, using inverse reinforcement learning methods such as the one in (66). We leave the exploration of such possibility for future work.

6.1.2 Subintentional I-POMDPs with PDFC Models

A subintentional interactive POMDP with PDFC models of other agents is an interactive POMDP (Equation 2.9) in which the agent function of j is implemented by a PDFC. Recall that a model of the other agent is a tuple[1] $m_j = (h_j, f_j, O_j)$, where h_j is a history for agent j, f_j is an agent function, and O_j is an observation function. We specialize the notion of model from the general I-POMDP case as $m_j = (q_j, c_j, O_j)$, where c_j is a PDFC that implements the agent function, and q_j is a node of the PDFC, that summarizes the past history. Since we assume that j's observation model is known, an interactive state is a tuple $is = (s, c_j, q_j)$. Ideally, c_j belongs to the set of all possible PDFCs. In practice, the set of PDFCs C_j that we consider is the finite ensemble resulting from Algorithm 4. This is not a severe limitation, however: the set C_j was indeed inferred from the set of all possible controllers by learning on past data, and

[1]Since there is no longer the need to keep track of complex time indexes, we switch to the more usual notation that indicates the agent with subscripts rather than superscripts.

hence represents an approximation of the distribution over the uncountably infinite set of all possible PDFCs.

The formula of the I-POMDP belief update in Equation 2.10 can be rewritten as:

$$p(is'|a_i, \omega_i, b) = \beta \sum_{is\,:\,c_j = c'_j} b(is) \sum_{a_j} \theta^{c_j}(q_j, a_j)\; O_i(a_i, a_j, s', \omega_i)\; p(is'|is, a_i, a_j), \tag{6.1}$$

where β is a normalization constant, and $p(is'|is, a_i, a_j)$ is the interactive transition function defined as:

$$p(is'|is, a_i, a_j) = T(s, a_i, a_j, s') \sum_{\omega_j} O_j(a_i, a_j, s', \omega_j)\; \delta_K(q'_j, \tau^{c_j}(q_j, a_j, \omega_j)). \tag{6.2}$$

Since there is no recursion into j's intentional models, a subintentional I-POMDP can be flattened into a POMDP. Therefore, standard POMDP algorithms can be adapted in order to compute a solution. However, the size of the interactive state space can be very large, even for simple problems, since we are considering a potentially large number of models of agent j.

For this reason, we use in this work the Monte Carlo tree search method for POMDPs (POMCP) introduced in (20) and described in Chapter 2 to solve the subintentional I-POMDPs. The running time of the algorithm is virtually independent from the size of the problem, since it makes use of a generator implementation of the I-POMDP rather than a flat or factorized specification.

6.2 Experimental Results

We demonstrate how the use of j's learned PDFCs in agent i's own planning improves the agent's performance. In this setting, j is oblivious to i's actions, and always operates according to its true controller. We consider the reward collected by agent i with respect to the amount of observations used for learning j's models. In particular, we used the same PDFCs that were generated during the evaluation of the learning algorithm in Section 5.2.

The rewards obtained by exploiting the learned PDFCs are compared against the following baselines:

- **Uniform (U):** agent i models j's actions uniformly at random, which is equivalent to a PDFC with single node and action distribution $p(a_j) = |A|^{-1}$.
- **Proportional (P):** i predicts j's actions with probability given by their observed frequency.
- **True (T):** i knows j's true controller.

The performance of the resulting I-POMDPs was computed by averaging the total reward collected during 1000 runs of the POMCP algorithm, with discount factor 0.9 and using 2^{10} simulations for exploring the search tree at each step; all other POMCP parameters were set as in (20).

Figure 18 reports agent i's mean total reward for the three problems and the median size of j's PDFCs. Numbers along the x-axes indicates the base-2 logarithm of the observation size, while letters identify baseline models. Note that as before, since the Tiger problem is much

smaller, we consider shorter training sequences. For all problems, the performance obtained when using the learned models of j is always higher than when using uniform or proportional models, and approaches the upper bound (known j's true model) with longer training sequences. This is because, with more information available, agent i is able to learn more accurate models that better predict j's actions.

We underline how, even with shorter observation sequences, agent i is able to learn models that provide a large performance gain over the random or proportional models. In particular for the Maze problem, using only 256 observations for learning, agent i's performance grows to about 90% of the difference between using the true model and the random model. Similar, albeit less extreme jumps are also observable for the other problems. This is a demonstration that, even though learning the exact model of another agent is unattainable, especially with realistic observability assumptions, we can still largely improve our performance by recognizing behavioral patterns that are statistically significant and encode them in a compact model.

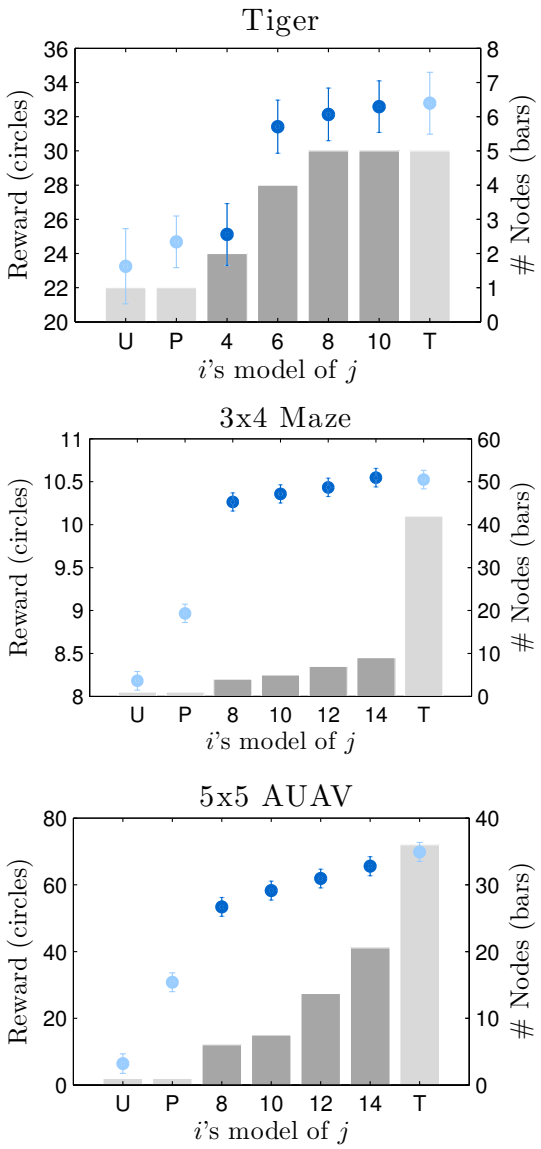

Figure 18. Reward by agent i with different models of agent j (lines) and number of nodes of learned PDFCs (bars).

CHAPTER 7

INTERLEAVED PLANNING AND LEARNING

The methodology described in the previous chapter, in which agent i first learns about agent j's policy, and then exploits this knowledge to provide its own best response during a subsequent interaction phase, is of limited application in many realistic scenarios. The underlying assumption that agent j's policy remains stationary, so that the models learned during the training phase can be exploited during the interaction phase, is very strong, and rules out the possibility that j's policy might also be adaptive. In particular, agent j may itself be modeling agent i, and therefore its policy may change throughout the interaction.

It is evident that endowing the agent with the capability of interleaving learning and planning is crucial for maximizing its rewards when facing unknown, possibly adaptive opponents. The methodology we propose combines the MCMC learning algorithm presented in Chapter 5 with the POMCP algorithm described in (20) and summarized in 2.1.2, as described in the following section.

7.1 Integrating learning and planning

The POMCP algorithm maintains a particle representation of the agent's belief about the state of the environment. In the context of a subintentional interactive POMDP, this means that each particle is a triple (s, c_j, q_j) whose elements represent respectively the physical state of the world, a PDFC for agent j, and a node within such PDFC. Note that s is not necessarily

atomic, and may be itself composed by a set of variables. The component c_j of the state belongs to a set of candidate PDFCs C_j, that are either given to the agent before the interaction or have been previously learned. In order to allow the agent to adapt to changing opponents, agent i needs to be able to modify the set C_j of possible models. In the methodology we propose, this is done by periodically re-training the PDFCs based on newly available information.

One possibility is for the agent to update the set of PDFCs at fixed time intervals. We hence introduce the parameter ΔT_{update}, that represents the number of timesteps that separates two updates. Each update is performed by invoking the MCMC algorithm described in Chapter 5. However, instead of initializing the MCMC state with a single-node PDFC, one of the current elements of C_j is used instead to "hot-start" the MCMC sampler. As a heuristic, we choose this PDFC that results from the majority vote among the current particles in agent i's belief state. The choice of hot-starting the sampler means that the models learned beforehand are not completely discarded.

The data that is used for re-training the models at each update is i's most recent history, going back W steps, i.e., at time t, $h^i_{t-W+1:t} = (a^i_{t-W:t-1}, \omega^i_{t-W+1:t})$. Note that the observation window length W and ΔT_{update} do not necessarily have the same value. The choice of these parameters might depend on several factors, mainly based on the time available to the agent to re-train models during execution. Ideally, agent i would like to update j's models frequently, so to keep up to speed even with an opponent that changes at a quick pace, but may not be able to do so because of runtime constraints.

7.1.1 Resampling j's PDFCs in the POMCP algorithm

As described above, a new set of PDFCs is obtained at each timestep t multiple of ΔT_{update}, by executing the MCMC algorithm with the observations gathered by i in the last W timesteps. We assume that the agent's interaction with the environment is on hold during the execution of the learning algorithm. In order for the learned PDFCs to be synchronized when the interaction resumes, it is important to know what node of the PDFC is occupied by agent j at time t, that is, at the *end* of the training sequence. Note that this is usually different from the initial node τ_0 of a PDFC, defined as the node agent j occupies at the *beginning* of the training sequence. In our setting, we assume that the sequence resampling procedure (Section 5.1.1) stores the sampled value of q_t^j, so that it is readily available when interaction resumes. Accordingly, we represent the learned PDFCs at time t as a set C_j^t of pairs (c_j, q_j), where c_j is a PDFC and q_j is the node occupied at time t.

Once a new set C_j^t has been obtained, the set of unweighted particles B^t that represents agent i's belief in the POMCP algorithm needs to be refreshed, in order to take into account the new models. As mentioned above, the particles in B^t form the empirical joint distribution $\hat{b}^t(s, c_j, q_j)$ over the set of world states and models of j, i.e.

$$\hat{b}^t = \frac{1}{|B^t|} \sum_{(s,c_j,q_j) \in B^t} \delta_D(s, c_j, q_j). \tag{7.1}$$

The set of particles is refreshed by drawing, for each particle, its new components (c_j, q_j) uniformly at random from C_j^t, leaving the state s unchanged. In this way, the marginalized belief over the state of the world encoded in the existing particles is preserved.

After resampling the belief, one last operation needs to take place before resuming execution. Recall that in the POMCP algorithm the current belief is the root of a belief sub-tree (the search tree starting from the current belief) that was previously expanded and stored. Normally, the subtree is retained in POMCP so that the new Monte Carlo search from the current node does not start from scratch, and utilizes the portion of the tree that had already been expanded under the current root. When j's PDFCs are resampled, however, the belief subtree must be pruned to avoid incompatibility, since it had been generated assuming a different set C_j, meaning that its branches extends into a state space that is different from the current, updated one.

7.2 Experimental Results

In this section, we report the results of the experiments conducted in order to assess the performance of online learning agents in various versions of the multiagent Tiger Problem.

7.2.1 Planning and Learning Against a Stationary Opponent

The multiagent Tiger Problem described in Section 5.2.1.1 induces a low degree of interaction between the two agents: for agent i, knowing agent j's policy is only useful insofar it reveals whether j has opened a door, so that i can infer that the state of the world might have been disrupted by resetting the tiger with uniform probability, and therefore recently heard growls might be uninformative. Even without any model of j, agent i would know with high

probability whether a door has been opened due to the high reliability of creaks. This is the reason why the reward increase for the Tiger Problem in Figure 18 is relatively smaller with respect to the confidence interval, compared to the other two problems. If the creaks were perfect, in fact, knowing j's model would be completely inconsequential.

For this reason, we modify the multiagent Tiger Problem to make the agents' rewards directly dependent on both agents' actions, so to increase the importance of accurate action predictions. In this new setup, the two agents receive a reward of 50, instead of 10, when they open the same door and find the gold. Moreover, we assume that the agents observe each others' last action perfectly at each timestep, that is, the creaks are perceived with 100% accuracy for any action. Note that, even though actions are observable, the agents do not perceive each others' observations. The complete specification for this version of the multiagent Tiger Problem is provided in Table III. This version of the tiger problem clearly promotes *cooperative behavior*: the two agents will want to synchronize and open the same door at the same time. In order to do so, they need to predict each others' actions accurately.

In this section, we explore the scenario in which only one agent, i, repeatedly updates j's model, while j does not. We consider three different types of agent j, described in the following:

1. Agent j acts according to the optimal solution to the single-agent Tiger Problem, i.e. the 5-nodes deterministic controller in Figure 7, that we denote here as c_{tiger}. This policy corresponds to j being oblivious to i's presence, or equivalently, assuming that agent i always listens.

TABLE III

SPECIFICATION OF THE COOPERATIVE MULTI-AGENT TIGER PROBLEM WITH OBSERVABLE ACTIONS. THE OBSERVATION FUNCTION IS FACTORED INTO *GROWLS* AND *CREAKS*.

Transition function

(a_i, a_j)	s	**TL**	**TR**
(L, L)	TL	1	0
(L, L)	TR	0	1
$(OL, *)$	$*$	0.5	0.5
$(OR, *)$	$*$	0.5	0.5
$(*, OL)$	$*$	0.5	0.5
$(*, OR)$	$*$	0.5	0.5

i's reward function

(a_i, a_j)	**TL**	**TR**
$(L, *)$	-1	-1
(OL, L)	-100	10
(OL, OL)	-100	50
(OL, OR)	-100	10
(OR, L)	10	-100
(OR, OL)	10	-100
(OR, OR)	50	-100

j's reward function

(a_i, a_j)	**TL**	**TR**
$(*, L)$	-1	-1
(L, OL)	-100	10
(OL, OL)	-100	50
(OR, OL)	-100	10
(L, OR)	10	-100
(OL, OR)	10	-100
(OR, OR)	50	-100

i's observation function

GROWLS

(a_i, a_j)	s	**GL**	**GR**
$(L, *)$	TL	0.85	0.15
$(L, *)$	TR	0.15	0.85
$(OL, *)$	$*$	0.5	0.5
$(OR, *)$	$*$	0.5	0.5

CREAKS

(a_i, a_j)	s	**S**	**CL**	**CR**
$(*, L)$	$*$	1	0	0
$(*, OL)$	$*$	0	1	0
$(*, OR)$	$*$	0	0	1

j's observation function

GROWLS

(a_i, a_j)	s	**GL**	**GR**
$(*, L)$	TL	0.85	0.15
$(*, L)$	TR	0.15	0.85
$(*, OL)$	$*$	0.5	0.5
$(*, OR)$	$*$	0.5	0.5

CREAKS

(a_i, a_j)	s	**S**	**CL**	**CR**
$(L, *)$	$*$	1	0	0
$(OL, *)$	$*$	0	1	0
$(OR, *)$	$*$	0	0	1

2. Agent j plans its actions online using the POMCP algorithm, solving a subintentional I-POMDP that assumes a uniform random model of i. We denote this as POMCP(c_{rand}).

3. Agent j plans online with POMCP, assuming that i's model is the 5-nodes solution to the single-agent Tiger Problem. This is denoted as POMCP(c_{tiger}).

In the experiments, the parameters of the POMCP algorithm were set identically for the two agents. In particular, 2^{12} simulations were used to select the agents' action at each timestep. Agent i initially assumes a completely random model of j, corresponding to a single-node PDFC with uniform action distribution. During execution, i updates j's models every $\Delta T_{update} = 500$ timesteps, based on its observations and actions in the last $W = 500$ timesteps. The parameters of the PDFC learning algorithm were set as follows: $N_{iter} = 2000$, $M = 50$, $R = 50$, $S = 2$, $\zeta_\alpha = \zeta_\lambda = 0.1$. After each update, the set of PDFCs C_t^j is obtained by subsampling the second half of the MCMC realization every 100 iterations, resulting in 20 PDFCs. Each scenario was simulated 50 times for 2500 timesteps.

Figure 19 reports the results obtained during such executions: the top row shows the rewards obtained by both agents in the three scenarios, while the bottom plots represent the average size of j's PDFCs learned by agent i. The rewards are averaged over segments of 500 timesteps, so to align with the model updates performed by agent i.

In all four cases, we can see that agent i's performance improves as a result of learning. This result is relevant because it shows that modeling j as a PDFC is robust with respect to the actual process that generates j's actions. While it is obvious that PDFCs are suitable to model behavior that is indeed generated by a finite controller, the results presented here show

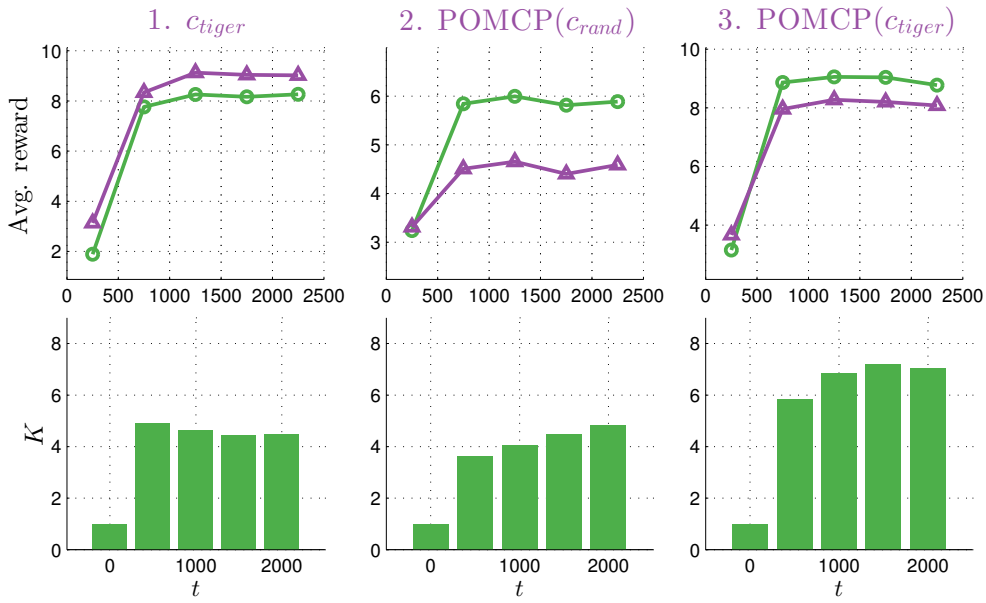

Figure 19. Agent i learns j's model during interaction, with j being either a PDFC or an online planning agent that assumes some stationary model of i. Top row reports average rewards for the agents, bottom row reports the size of j's model as learned by i.

that PDFCs are a good choice even when the behavior is generated by a process (here, online POMDP planning) that lies outside the space of finite state controllers. Note that, for now, we are considering opponents that do not adapt themselves.

The results also show that the increment in i's reward varies depending on j's type. In the following, we analyze each case separately.

1. $\boxed{j\text{'s policy: } c_{tiger}}$

   When j is implemented by the controller that solves the single-agent Tiger Problem, we can see that both agents' rewards increase by a large margin as soon as i learns about j. The fact that j's rewards also improve is due to the fact that we are dealing with a

cooperative environment, and therefore an increase in i's reward implies that j's payoff also goes up. In particular, agent i tries to synchronize door openings with j, and can do so successfully when it has learned an accurate model of j's behavior. The fact that j's average rewards are actually *higher* than i's is somewhat counter-intuitive: how come that agent i, who is here doing the "heavy lifting" for improving cooperation, receives a lower reward? The reason lies in the fact that there is no competition at all between i and j in this scenario, and i is only concerned about its own payoff; it so happens that maximizing i's expected rewards leads to j actually receiving an even higher payoff. To validate this result, let us consider the hypothetical case in which i knows that j's policy is c_{tiger}, and does not perform any learning. The flattened I-POMDP is small enough to be solved by value iteration[1], resulting in a 10-node finite controller. We can then compute the expected average utility as described in Appendix B, obtaining the values 8.33 and 9.26 respectively for i and j, which are the values that the agents' rewards converge to in our experiment. This means that i learns j's model and is able to provide a best response that is as good as if it *knew* j's true model

As a possible follow-up question, we may ask: does this mean that i would be better off playing according to a simple 5-node controller, as j is doing? The answer is, obviously, no. This is because if both agents were to play according to such policy, the expected average reward would be just 4.59. Indeed, one could conceive a meta game-like setting

[1] we use the "witness algorithm" (2).

in which the two agents have to choose their strategy (e.g. a PDFC, POMCP, POMCP with learning, etc.) and then implement their policy. Although this seems like a fascinating direction, this meta-problem takes us to game-theoretic solution concepts, which constitute the approach to multiagent systems from which we are trying to depart in the first place, by proposing a methodology based on decision theory.

2. j's policy: POMCP(c_{rand})

   The plot in the middle of Figure 19 shows the rewards of the two agents when j is choosing its actions online with the POMCP algorithm, as the best response to a completely random model of i. Both rewards increase as a result of i learning, but j's utility remains substantially lower than i's. This is due to the fact that j is planning against a model of i that is very different from i's actual decision process. In particular, modeling i as random gives j more confidence when it comes to opening a door, since j believes that at each timestep i will open the same door with probability $1/3$, effectively raising the payoff of finding the gold to 26.67. Agent i learns a model of this behavior and tries to synchronize its own actions. Despite being agent i's better option when facing this type of agent, this leads to a lower utility than the previous case, in which syncing with j's policy led to less risk and higher payoff. Note that at the beginning the average reward is the same for both agents, since they both have a random model of the other.

3. j's policy: POMCP(c_{tiger})

   The last plot on the right reports the agents' rewards when j plans assuming that i behaves according to the 5-node solution to the single-agent scenario. The utilities that

the agents achieve are higher than in the previous case, and seem to mirror the utilities observed in the first scenario that we analyzed above, with i's rewards now being slightly higher than j's. This observation is indeed very interesting. Arguably, j is now adopting the same kind of behavior that i was using in that first scenario, since its model of i is now the same controller that i was planning against in that case (although i had to learn such controller.) Therefore, we know there must be a policy for i that provides at least as much reward as j collected in the first scenario, which is exactly what we see in the plot.

The charts on the bottom of Figure 19 display the mean size of the inferred PDFCs. We can see that i learns significantly larger controllers in the third scenario. This is line with the complexity of agent j's true policy, especially in light of the analysis above, that describes how in the first two settings agent j indeed follows a rather simple policy. In the third scenario, j's decision making becomes more involved, which causes i to learn more complex models. This showcases the ability of our Bayesian nonparametric learning approach, that infers PDFCs whose complexity scales with the complexity of observed data.

7.2.2 Self Play: Coordination and Discoordination

In this section, we consider the case in which both agents periodically update each others' model. Following game theoretic terminology, we call this scenario *self-play*, indicating that the same learning algorithm is employed by both agents. Our goal is to explore the dynamics that ensue when the agents are learning about each other at the same time. For this experiments, we use the same baseline domain as the previous section, but consider alternative reward functions for the agents. The three reward functions we consider defines three types of agent:

- **Altruistic.** This payoff function is the same as the previous section: the agent receives a reward of 50 instead of 10 when it opens the door hiding the gold and the other agent opens the same door.
- **Indifferent.** When finding the gold, this agent does not care whether the other agents opens the same door or not, and receives a reward of 10 regardless. This reward function is the same as for the original two-agent Tiger Problem.
- **Selfish.** The agent receives a reward of 50 instead of 10 when finding the gold, and the other agent either opens the other door or listens.

Table IV formally specifies these reward functions, from the point of view of agent i.

TABLE IV

ALTERNATIVE REWARD MODELS FOR THE MULTIAGENT TIGER GAME.

"Altruistic" agent

(a_i, a_j)	**TL**	**TR**
$(L, *)$	-1	-1
(OL, L)	-100	10
(OL, OL)	-100	50
(OL, OR)	-100	10
(OR, L)	10	-100
(OR, OL)	10	-100
(OR, OR)	50	-100

"Indifferent" agent

(a_i, a_j)	TL	TR
$(L, *)$	-1	-1
(OL, L)	-100	10
(OL, OL)	-100	10
(OL, OR)	-100	10
(OR, L)	10	-100
(OR, OL)	10	-100
(OR, OR)	10	-100

"Selfish" agent

(a_i, a_j)	TL	TR
$(L, *)$	-1	-1
(OL, L)	-100	50
(OL, OL)	-100	10
(OL, OR)	-100	50
(OR, L)	50	-100
(OR, OL)	50	-100
(OR, OR)	10	-100

A set of experiments was conducted by considering the reward functions above to be assigned to two agents i and j; since the observation and transition functions of this I-POMDP are symmetric for the two agents, there are a total of 6 possible combinations. We remark here that the agents do not know, nor model, the other agent's reward function. Both agents plan their actions online with the POMCP algorithm and update each others' models every 500 timesteps, using an observation window of length 500. All parameters of the POMCP planner and the MCMC algorithm were set as in the previous section, except that the agents interact for 5000 timesteps. Figure 20 reports the average rewards obtained by the two agents for each experiment, each averaged over 50 executions, while Figure 21 contains the average size of the controllers learned by both agents. In the following, we analyze individually the results for each combination of agent types.

7.2.2.1 Analysis of the Agent's Behavior

- i: ALTRUISTIC, j: ALTRUISTIC

  When both agents want to achieve coordination, their average reward increases quickly (two rounds of model updates) and then stabilizes between 10 and 11. As expected, the rewards for the two agents are the same, except for some noise, since they have the same reward function and the environment is perfectly symmetrical. The equilibrium behavior that is achieved consists of both agents opening a door every other action; specifically, they open the door opposite the source of the tiger's growl heard in the previous timestep. We denote this behavior as *aggressive*, because, differently from the single-agent Tiger Problem, the agents do not wait for two agreeing growls before opening

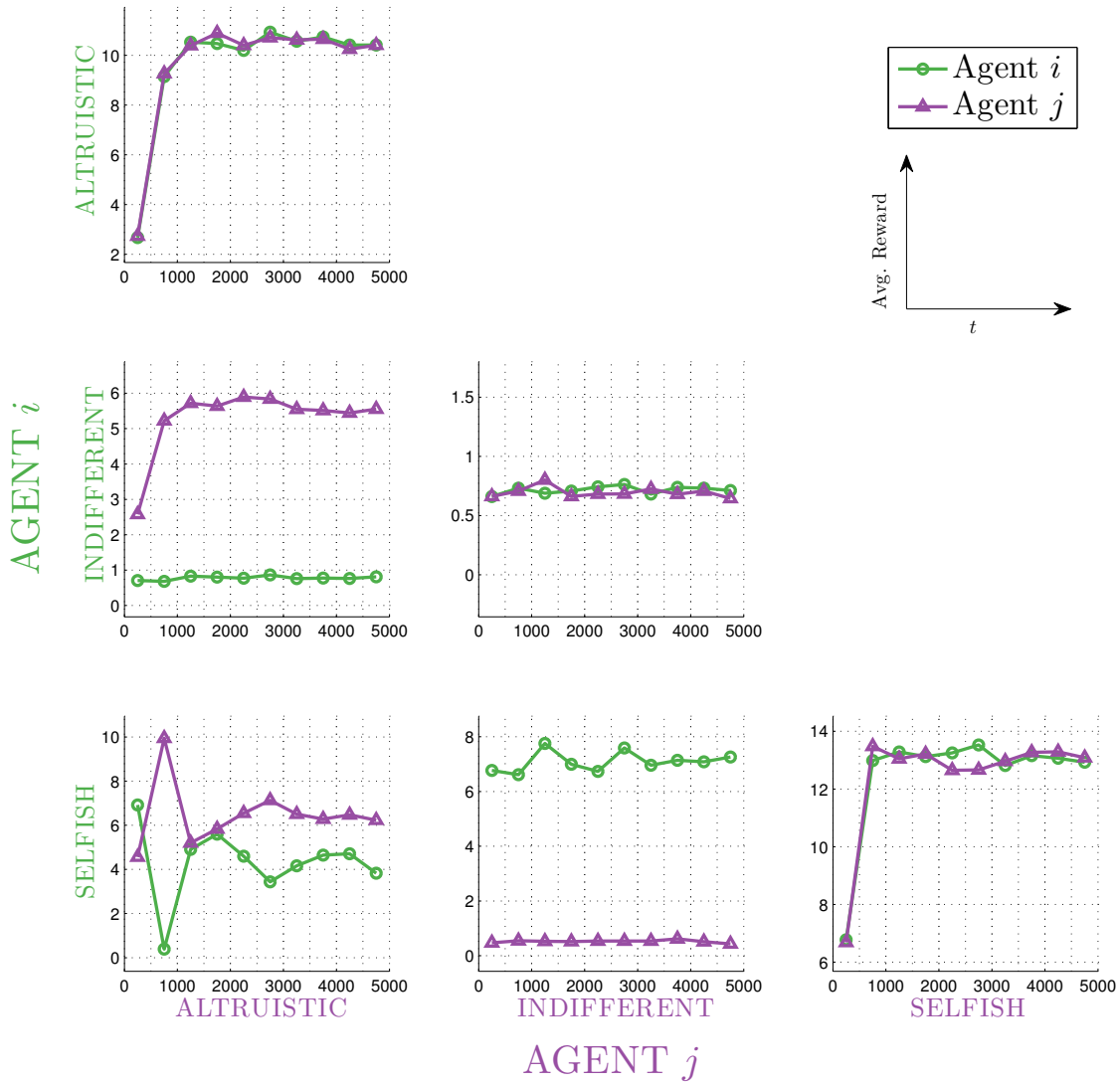

Figure 20. Average rewards for the two agents when both are learning, with respect to different combinations of payoff functions.

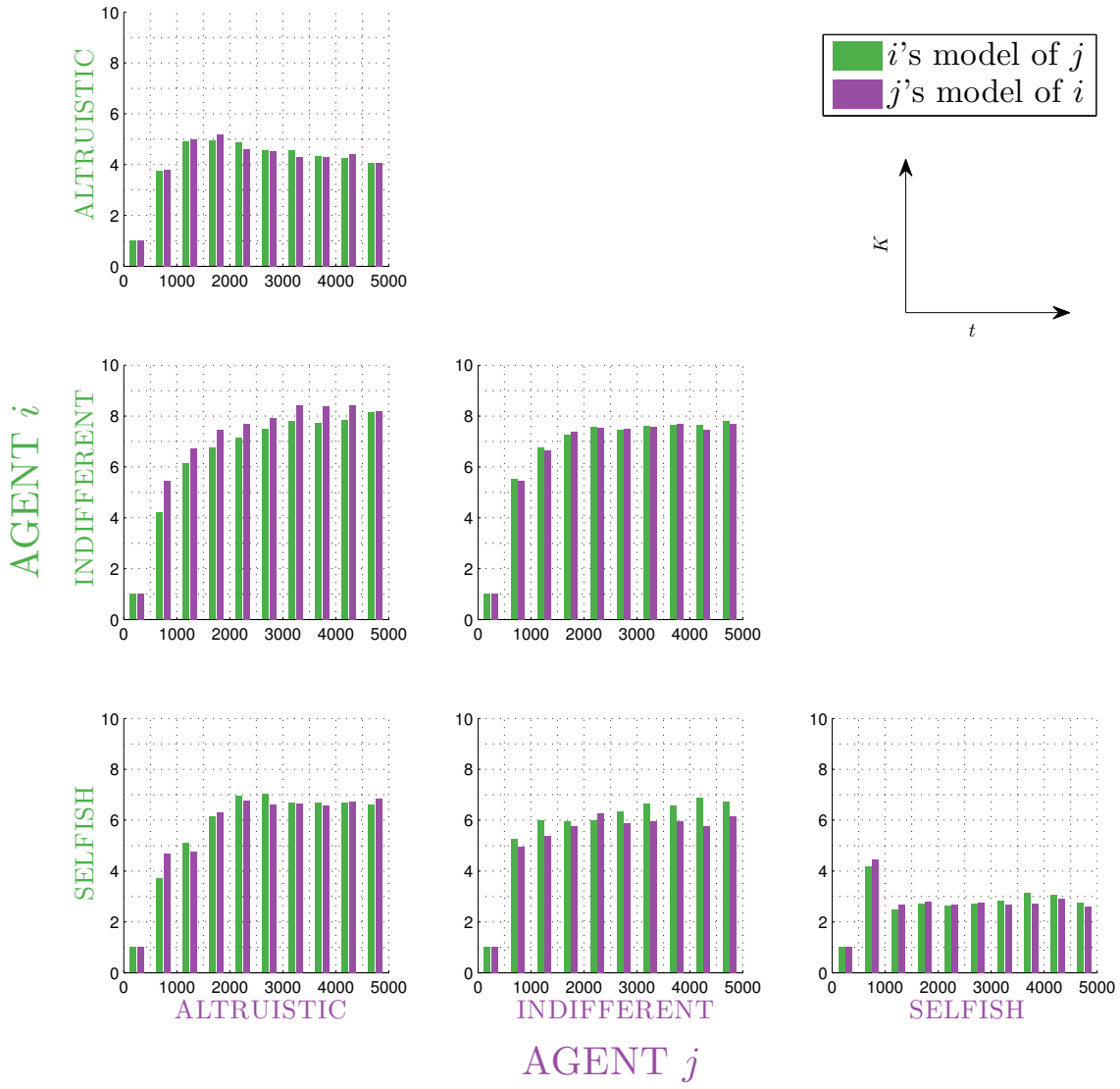

Figure 21. Number of nodes of learned PDFCs for the two agents when both are learning, with respect to different payoff functions.

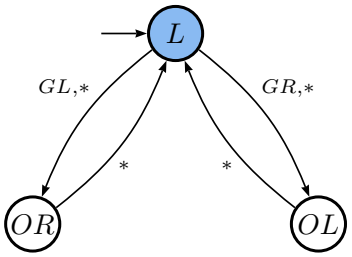

Figure 22. Finite state controller for the Tiger Problem with three nodes; the agent opens a door every other action, after only one listen.

a door. Moreover, their actions are *synchronized*: they both listen and then both open a door, and so on, resulting in a highly cooperative behavior. The agents can be aggressive because, having learned each others' model, they are confident enough that the other agent will likely open the same door, which increases the expected utility of opening the door. This pair of strategies is arguably Pareto-optimal, that is, no strategy pair other than both agents being aggressive and synchronized yields to higher payoffs for either agent. In fact, opening a door at each timestep would be detrimental, because the agents would have no idea about the position of the tiger, yielding to lower expected payoff. If instead they would choose to wait longer, they would "waste" time, since in the long run it pays off to take a higher (but calculated) risk and start over instead of waiting for another timestep. Of course, this argument would not necessarily hold were the agents to plan for a short finite horizon.

The fact that this pair of strategies indeed constitutes an equilibrium is confirmed by observing that an aggressive strategy can be encoded in the 3-node deterministic controller

in Figure 22, where the starting node is picked by assuming that the agent is initially uninformed about the position of the tiger (i.e. has a $(0.5, 0.5)$ belief.) Assume now that one agent, i, models the other agent j with such controller. The resulting flattened I-POMDP is sufficiently simple, and i's best response policy can be computed exactly via value iteration; we obtain that i's best response, when i is also initially uninformed about the location of the tiger, *is the same 3-node controller*. Therefore, this strategy pair constitutes a mutual best response, that is, the two strategies are in equilibrium. Moreover, the average expected utility of the two agents can be derived analytically, and is equal to 10.7, which is exactly the average payoff that we observe in the reward plot. Note that in our experiment the two agents do not know each others' initial belief about the tiger's location (actually, the two agents never model each others' beliefs explicitly;) however, the creaks are sufficient for them to synchronize their behavior and open at the same time. This is a remarkable result and shows that, in this problem, Pareto-optimal cooperation emerges naturally from our methodology, without the agents having any knowledge of each others' payoff.

In Figure 21 we can see that the corresponding mean size of the learned PDFCs converges to around four nodes. The fact that the PDFCs are on average slightly larger than the 3-node minimal controller that describe the agents' behavior can be attributed to noisy observation, and the fact that, when opening, the agents do not perceive an informative growl, therefore their history of observations is less informative than a sequence of the same length obtained by an agent that is always listening. Nonetheless, we can see clearly

that the size of the inferred PDFC does not increase as time progresses, which is indicative of the fact that the complexity of the learned controllers aligns with the complexity of the observed behavior, thanks to the use of a Bayesian nonparametric prior.

- i: INDIFFERENT, j: ALTRUISTIC

  In this scenario, j has an altruistic reward function that makes it strive for coordination, once adequate models of i's behavior have been learned, its rewards increase, as shown in the corresponding plot in Figure 20. As for i, recall that an indifferent agent gets the same reward when finding gold, regardless of what the other agent does in that timestep. As we mentioned before, modeling another agent is inconsequential from the point of view of an indifferent agent in this environment since the agents are perfectly informed about each others' past action, and can therefore determine whether the tiger's position might have been reset. This is why, at a first look, there is no noticeable change in i's average reward (green line,) although there is in fact a slight increase for $t > 1000$. This is due to the fact that as i becomes more successful in coordinating its openings with j's, the position of the tiger is less likely to be disrupted by i while j is listening, making it easier for j to acquire consistent observations.

  We observe from Figure 21 that the size of the learned PDFCs in this scenario increases over time, since the policy of the two agents is more involved than in the previous case of two altruistic agents. Note that agent j's model of i also grows in complexity, although as mentioned above i's model is inconsequential for j's planning; however, j does not

perform itself the meta-reasoning that lead us to such conclusion, and still tries to learn accurate models of i.

- i: INDIFFERENT, j: INDIFFERENT

  Two indifferent agents have no incentive to coordinate their actions. The only way in which their actions affect each other is by resetting the position of the tiger, and since they know exactly when this event might occur from hearing perfect creaks, their models of each other are inconsequential. This explains why there is no significant change in their rewards as time progresses.

- i: SELFISH, j: ALTRUISTIC

  In this scenario, the two agents have conflicting goals: one agent wants to synchronize door openings, while the other agent has the opposite goal of opening a door when the other agent does *not* open the same door. We refer to this situation as *discoordination*. Note that both agents are still trying to avoid the tiger and acquire the gold, so there is a delicate balance of interests at play. The corresponding reward plot in Figure 20 reveals an interesting dynamic, described in the following. First, let us observe that, by initially assuming a completely random model of the other agent, the selfish agent (i) starts off acting *aggressively*, that is, adopting the behavior encoded in the controller of Figure 22. This is because i believes at each timestep that j will *not* open the same door with probability $\frac{2}{3}$ (corresponding to agent j opening the other door or listening.) In contrast, the altruistic agent j is initially more cautious, since it attributes a probability of only $\frac{1}{3}$ to i opening the same door. Agent i's aggressive behavior is very easily recognized by j

as soon as learning takes place, at which point j starts synchronizing its openings with i's. At the same time, i learns that j is acting cautiously, leading it to continue with its aggressive behavior, which j has no problem taking advantage of, increasing its rewards and sinking i's.

At the next iteration of learning, agent i becomes aware of j's aggressive synchronizing behavior, and starts opening doors only immediately after hearing a creak from j, knowing that i will be then listening, intuitively beating agent j at its own game. This has the result of increasing j's reward and lowering i's, and at the same time slows down the pace at which the agents open doors, so neither reward becomes particularly high or low.

As time progresses, the combined behavior of the two agents starts to vary from run to run and becomes more difficult to analyze. This is due to the stochastic nature of the environment, the agent's planning, and their learning. However, by looking at traces from individual runs, one could observe that the mutual modeling dynamics described above re-emerge at irregular intervals through the agents' interaction.

- i: SELFISH, j: INDIFFERENT

  As described above, the selfish agent, i, starts behaving aggressively since the start. Even after learning agent j's PDFCs, the aggressive strategy remains optimal for the agent, so there is no noticeable upward or downward trend in the agents' rewards in the corresponding plot in Figure 20.

- i: SELFISH, j: SELFISH

  Given the fact that they both want to "avoid each other", two selfish agents have cooper-

ative reward functions. Arguably, their best combined strategy is to behave aggressively in an *anti-synchronization* pattern, so that each agent opens a door while the other is listening. This pair of strategies can be encoded with two 3-node controllers of the type showed in Figure 22, with different starting nodes, that is, one agent starts with a listen action, while the other starts by opening either door. This pair of policies can be shown to be a best response to each other by exactly solving the resulting flattened I-POMDP exactly from the point of view of either agent. It is easy in this case to compute the expected utility if the two agents were indeed to use such policies, by considering that when opening the door opposite from the direction of the perceived growl, an agent will receive a reward of 50 with probability 0.85 (gold, not shared, since the other agent is listening), and of -100 with probability 0.15 (tiger). Since the agent listens half the time, with a cost of -1, the expected average payoff is $\frac{1}{2}(-1 + 50 * 0.85 - 100 * 0.15) = 13.15$. The corresponding plot in Figure 20 shows that this is indeed the average reward obtained by the agents, that are able to recognize each others' aggressive policy and start anti-synchronizing their actions as soon as they learn about each others' models.

The size K of the learned PDFCs has an interesting trend. From the corresponding chart in Figure 21, we see that after learning for the first time, the PDFCs have on average about 4 nodes, for both agents. These PDFC capture the aggressive behavior that the two agents observe about each other in the first 500 timesteps. After learning happens the second time at $t = 1000$, however, the size of the learned PDFCs drops to slightly above two nodes. This is interesting, and is explained by the fact that when successfully

playing an aggressive anti-synchronization policy (i.e. door openings are interleaved,) one agent, say i, cannot speculate on what observation the other agent may have received while i itself is opening a door, and therefore there is no observed pattern that reveals that the other agent is opening the opposite door than the one it receives the growl from. All is there to infer is the alternation of listening and (any) opening, which can indeed be encoded in a PDFC with 2 nodes. Once again, this shows that the Bayesian nonparametric prior allows the agents to learn PDFCs whose size mirrors the complexity of *observed* behavior.

7.2.3 Social Dynamics: "Follow the Leader"

In this section, we consider a further variation of the multiagent Tiger Problem. As before, the agents receive creaks at each timestep, that perfectly reveals the last action of the other agent. Moreover, we consider the case in which both agents are "indifferent" with respect to sharing or not sharing the gold with the other agents, that is, the reward functions are the same as in the original multiagent Tiger Problem (6). In this case, however, the agents are endowed with asymmetric growl observation functions: agent i receives a growl from the door hiding the tiger with high accuracy 0.96, while agent j has a lower accuracy of only 0.7. Moreover, when either agent opens a door, the tiger persists in its current location with 0.96 probability, instead of being relocated uniformly at random. The formal description of this I-POMDP is provided in Table V.

Given its high hearing accuracy, agent i only needs to hear one growl in order to be sufficiently confident about the position of the tiger to open the opposite door. Because it is likely

TABLE V

SPECIFICATION OF THE COOPERATIVE MULTI-AGENT TIGER PROBLEM IN THE "FOLLOW THE LEADER" SCENARIO. THE POSITION OF THE TIGER IS PERSISTENT UPON OPENINGS WITH PROBABILITY 0.96 AND THE AGENTS HAVE ASYMMETRIC GROWL HEARING ABILITIES.

Transition function

(a_i, a_j)	s	TL	TR
(L, L)	TL	1	0
(L, L)	TR	0	1
$(OL, *)$	TL	0.96	0.04
$(OL, *)$	TR	0.04	0.96
$(OR, *)$	TL	0.96	0.04
$(OR, *)$	TR	0.04	0.96
$(*, OL)$	TL	0.96	0.04
$(*, OL)$	TR	0.04	0.96
$(*, OR)$	TL	0.96	0.04
$(*, OL)$	TR	0.04	0.96

i's reward function

(a_i, a_j)	**TL**	**TR**
$(L, *)$	-1	-1
(OL, L)	-100	10
(OL, OL)	-100	10
(OL, OR)	-100	10
(OR, L)	10	-100
(OR, OL)	10	-100
(OR, OR)	10	-100

j's reward function

(a_i, a_j)	**TL**	**TR**
$(*, L)$	-1	-1
(L, OL)	-100	10
(OL, OL)	-100	10
(OR, OL)	-100	10
(L, OR)	10	-100
(OL, OR)	10	-100
(OR, OR)	10	-100

i's observation function

GROWLS

(a_i, a_j)	s	**GL**	**GR**
$(L, *)$	TL	0.96	0.04
$(L, *)$	TR	0.04	0.96
$(OL, *)$	$*$	0.5	0.5
$(OR, *)$	$*$	0.5	0.5

CREAKS

(a_i, a_j)	s	**S**	**CL**	**CR**
$(*, L)$	$*$	1	0	0
$(*, OL)$	$*$	0	1	0
$(*, OR)$	$*$	0	0	1

j's observation function

GROWLS

(a_i, a_j)	s	**GL**	**GR**
$(*, L)$	TL	0.7	0.3
$(*, L)$	TR	0.3	0.7
$(*, OL)$	$*$	0.5	0.5
$(*, OR)$	$*$	0.5	0.5

CREAKS

(a_i, a_j)	s	**S**	**CL**	**CR**
$(L, *)$	$*$	1	0	0
$(OL, *)$	$*$	0	1	0
$(OR, *)$	$*$	0	0	1

that the tiger persists at its current location, agent i will open the same door again immediately, and only after it will need to perform another listen, before possibly opening a door again. On the other hand, agent i becomes confident enough about the tiger's position only after the number of growls coming from one door is 5 more than the growls coming from the other door, assuming that it starts from a uniform belief. After opening, j needs to listen again immediately before being able to open a door again. Moreover, agent i might open a door while j is trying to listen, effectively further deteriorating j's already feeble hearing ability.

Based on these considerations, the initial interaction that takes place between these two agents is very asymmetric, with i opening doors very frequently and collecting high rewards, and j acting more timidly and gathering very little. The initial average rewards of the two agents in Figure 23 show that this is the case.

Recall from the analysis above that when the agents' hearing capabilities are the same and both are of the "indifferent" type, there is no performance improvement when the agents model each other, since knowing what actions the other agent is about to do is inconsequential, given that the immediate rewards depend directly only on each agent's own actions, and the other agent's previous action is revealed in the next timestep.

For the scenario we consider in this section, however, we see in the plot that this is not the case: there is a significant improvement in j's rewards after learning about i's model. This is due to j's ability to perform inference about the location of the tiger from observing i's actions. Intuitively, j's reasoning is that, if agent i (who j knows has good hearing) opened a door, it means that the gold was probably there, and it likely still is, given that it persists with

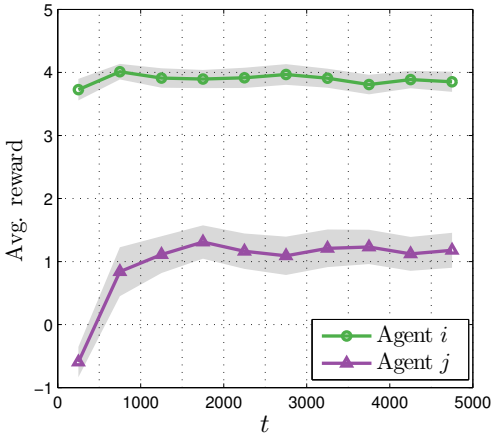

Figure 23. Average rewards (left) and number of inferred nodes (right) for the "follow the leader" scenario. The gray area represents the 95% confidence interval.

high probability in its location. This social dynamics has previously been analyzed from an I-POMDP perspective using intentional models (67), and given the name "Follow the Leader". In the intentional case, however, it is assumed that j knows i's reward function: it is then clear that j is able to perform inference on the state via i actions since it knows what i's preferences are. In other words, j knows that i does not *want* to open the door that hides the tiger, and therefore it is likely that the door it opens leads to the gold.

In our case, instead, j does not know what agent i likes or does not like, and in fact never attempts at modeling i's preferences. Still, j is able to increase its performance as a result of learning about i's policy in the form of a PDFC. This is somewhat surprising: j's observation function does not change as a result of learning, yet j is able to better locate the tiger by "piggy-back riding" on i's behavior, even though j does not know about i's preferences. This

is possible because, using its own (very uncertain!) past observations, j is able to recognize a pattern between the growls, that are stochastically related to the position of the tiger, and the creaks, that depend on i's actions. This establishes a relation between i's actions and the state of the world, that is implicitly encoded in the learned PDFCs. Agent j then exploits this pattern to effectively augment its feeble hearing.

7.2.4 Summary of Results

In this section, we summarize the main results obtained from the experiments described in this chapter.

- Modeling another agent j as a PDFC is robust with respect to the process underlying j's decision making, for the case we tested, i.e. when agent is a PDFC, an online planner with static model of i, or models i back (self play). This shows that the hypothesis space of PDFCs and the proposed methodology are powerful enough to capture regularities even in behaviors that are not actually generated by finite state controllers. [Section 7.2.1]
- Two learning agents in self play achieve coordination in the multiagent Tiger Problem when their reward functions are compatible. This is remarkable because the agents do not know anything about each others' payoff. Coordination emerges naturally from reciprocal modeling. [Section 7.2.2]
- In the multiagent Tiger Problem, we show an example of two learning agents with competitive payoffs (discoordination) that take turns at exploiting each others' model. [Section 7.2.2]

- The size of the inferred PDFCs reflect the complexity of behavior exhibited by the modeled agents. This validates the use of Bayesian nonparametrics. [Section 7.2.1, Section 7.2.2]
- A myopic (low hearing accuracy) agent is able to learn models of a more informed agent and exploit them to increase its own payoff, by performing inference based on the other agents' actions ("Follow the Leader"). This effectively augments the observation capabilities of the modeling agent. Again, this is remarkable because the agents do not know each others' reward functions: the myopic agent is able to "piggy back ride" on the more informed agents' actions without knowing *why* the latter opens a door or listens. [Section 7.2.3]

CHAPTER 8

CONCLUSION AND FUTURE DIRECTIONS

In this thesis, we presented a novel approach to planning in interactive POMDPs where other agents are modeled as probabilistic deterministic finite controllers (PDFCs). In order to do so, we constructed a suitable Bayesian nonparametric prior on the space of PDFCs that allows the size of the controllers to scale with the complexity of the observed behavior. We designed an ad-hoc MCMC learning algorithm that allows to learn accurate PDFCs representing agents' policies from a sequence of observations, that is in general only a noisy realization of the modeled agent's behavior. We showed how to embed the learned controllers in the modeling agent's own decision making process, adapting the general I-POMDP framework to consider subintentional models. Moreover, we provided a methodology that allows an agent to interleave learning about other agents and planning during execution, by combining our MCMC learning algorithm with an online POMDP solver.

Our reported experimental results show that this type of opponent modeling allows the planning agent to increase its performance. The results also show that when two agents are simultaneously learning about each other, coordination can emerge naturally from our approach.

One strength of our methodology is that it can be applied to situations where the modeling agent has little prior knowledge about the opponents' type. In particular, we assume that the agent is completely uninformed about their reward function. To the best of our knowledge, this is the first approach to I-POMDPs that operates under such assumptions. Therefore, we

consider our methodology very promising, and advocate that subintentional modeling should be considered as a strong alternative to intentional modeling in interactive POMDPs and multiagent planning problems in general.

In the following, we provide possible directions for future research.

8.1 Truly Online PDFC Inference and Scaling Up

In this work, we show how the MCMC algorithm can be used periodically during interaction to update the models of other agents. However, a fully embedded learning method would update the posterior distribution over all PDFCs at each timestep during execution, and not consider only the PDFCs that resulted from the last run of the MCMC algorithm.

We believe that a particle-based approach to learning PDFCs could fit well within the POMCP planning algorithm. However, online sampling of nonparametric Bayesian models is still not supported by well-developed methodologies. Nonetheless, approaches such as particle learning (68) seem promising for being applied to our problem.

The proposed approach has been shown to work for problems of low and medium complexity. We showed that our learning algorithm behaves well with controllers and environments each with a few dozens states. While this may be sufficient for some applications, some real-world problem have a higher complexity, and would make our approach too slow in time-sensitive situations.

We believe that particle-based methods can provide as much speedup in the learning phase as they do in online POMDP planning. Future efforts should then be focused on sequential particle-based methods for PDFC learning, to be fully integrated within the POMCP algorithm.

8.2 Inferring Other Agents' Observation Function

Recall that a *model* of another agent in the interactive state space of an I-POMDP is composed by an agent function *and* an observation function, that describes how the world feeds that agent information in the considered model. In this thesis, we assume that the observation function of another agent is known. However, this may sometimes not be true in realistic situations. Relaxing this assumption while constraining the other agent's possible observation functions to belong to a finite set would not represent a challenge for our approach: we would simply have to learn a different set of PDFCs for each observation function. A less trivial problem would arise if the modeling agent would be completely noninformed about the others' observation functions.

To tackle this, a possibility would be to treat the observation parameters of the opponents as yet another variable to be estimated by our Bayesian inference procedure. Since the observation function of an agent is a set of multinomial distributions, we consider it part of the state of the MCMC algorithm and place a Dirichlet prior over it. A preliminary investigation in this direction can be found in our previous work (69).

Dropping the assumption of knowledge of other agents' observation functions (and in a strong sense, of having *any* prior knowledge about it) raises some questions about what is indeed learnable. In particular, it does not seem possible, in theory, to learn both another agent's PDFC *and* its observation functions as separate entities. This is due to the fact that the learning agent would not know which of the two elements is responsible for the modeled agent to act a certain way. Therefore, we believe that a methodology that avoids treating the

two separately is more appropriate. For example, a modeling agent i can attempt at learning a combined model that directly relates its own actions and observations to the actions of another agent, that is, learning a representation of $g : (\Omega_i \times A_i)^* \rightarrow A_j$. We believe that this is an interesting direction for future work, that would make our approach applicable under even less stringent assumptions.

8.3 Inferring Other Agents' Utilities

As mentioned several times in this thesis, our approach makes no assumption of knowledge about the other agents' preferences. This makes the approach general, but as we point out in Chapter 6, the learned PDFCs are hardly applicable outside the domain where they were obtained. On the other hand, learning explicitly about an agent's preferences would instead generate *transferable* knowledge that the agent can exploit in different situations.

This is similar to a problem known as *inverse reinforcement learning*, that has been extensively studied in the context of (fully observable) MDPs (70; 71; 72). Although there are less results for inverse reinforcement learning in partially observable domains, we believe that recent approaches, such as (66), can be applied to our case, making it possible to back-engineer information about an agent's utilities from the learned PDFC policies.

APPENDICES

Appendix A

COMPUTING THE NODE CO-FREQUENCY OF TWO PDFCS

Let $m = (S, A, \Omega, T, O, R)$ be a POMDP specification, and let $c_T = (\Omega, A, Q_T, \tau_T, \theta_T, \tau_{0_T})$ and $c_L = (\Omega, A, Q_L, \tau_L, \theta_L, \tau_{0_L})$ be two PDFCs. We refer to to c_T and c_L as the *true* and the *learned* PDFC, respectively, indicating that the world responds solely on the actions of c_T. Let us introduce the following definition.

Definition 3. *The* T-composition *of c_T and c_L is a PDFC $c_{TL} = (\Omega, A, Q_{TL}, \tau_{TL}, \theta_{TL}, \tau_{0_{TL}})$ such that:*

- *$Q_{TL} = Q_T \times Q_L$, that is, each state is defined as a pair (q_T, q_L) where $q_T \in Q_T$ and $q_L \in Q_L$.*
- *$\tau_{TL} : Q_{TL} \times A \times \Omega \to Q_{TL}$ is defined as follows: $\forall\ q_L \in Q_L, q_T \in Q_T, a \in A, \omega \in \Omega$,*

$$\tau_{TL}((q_T, q_L), a, \omega) = (q'_T, q'_L) \quad \iff \quad \tau_T(q_T, \omega, a) = q'_T \ \wedge\ \tau_L(q_L, \omega, a) = q'_L. \qquad \text{(A.1)}$$

- *$\theta_{TL} : Q_{TL} \to \Delta(A)$ is defined as follows: $\forall\ q_L \in Q_L, q_T \in Q_T$, $\theta_{TL}((q_T, q_L), \cdot) = \theta_T(q_T, \cdot)$, that is, the action is generated according to the true PDFC's distribution.*
- *$\tau_{0_{TL}} = (\tau_{0_T}, \tau_{0_L})$.*

We next derive the transition matrix D_{mc} of a Markov chain whose states are given by the cross product of the states S in POMDP m and the nodes Q in PDFC c. The transition matrix

Appendix A (Continued)

is computed by combining the dynamics of the POMDP environment and the components of the PDFC. We have that the probability of transitioning from a state (q, s) to a state (s', q') is:

$$p(s', q'|s, q) = D_{mc}((s, q), (s'q')) = \sum_{a \in A} \theta_c(q, a) \sum_{s' \in S} T(s, a, s') \sum_{\omega \in \Omega} O(a, s', \omega)\ \delta_K(q', \tau_c(q, a, \omega)). \tag{A.2}$$

If D_{mc} represents an irreducible Markov chain, we compute its stationary distribution π. We can then consider the marginalized distribution $\pi_Q(q) = \sum_{s \in S} \pi(s, q)$ and observe that the node co-frequency $\eta_{q_T q_L} = \pi_{Q_{TL}}((q_T, q_L))$ is indeed the marginalized distribution of the Markov chain built in this way, by using POMDP m and the T-composition c_{TL} of the two PDFCs c_T and c_L.

Note that, even if the chain is irreducible but not aperiodic, the stationary distribution still represents the proportion of time spent in a given node in the limit. However, if the resulting Markov chain is not irreducible, we first have to compute its communication classes, and derive the stationary distribution of each. We can then combine the resulting distributions by considering the probability of the chain to reach each communication class.

Appendix B

COMPUTING THE EXPECTED I-POMDP REWARDS OF TWO PDFC AGENTS

Assume that we are given the description of an I-POMDP and two compliant PDFCs c_i and c_j, that prescribe the behavior of agent i and j respectively. By "compliance", we refer to the fact that the observation and action sets of the I-POMDP and the PDFCs correspond for both agents. We show how to compute the agents' expected long-term average rewards, defined for agent i as:

$$R_i^{avg} = \lim_{n\to\infty} \sum_{t=1}^{n} \big[\bar{R}_i^t | c_i, c_j\big], \tag{B.1}$$

where $\bar{R}^t|c_i, c_j$ represents the random variable denoting the reward obtained at time t when the agents are operating according to PDFCs c_i and c_j. The definition for agent j is straightforward.

First, let us define the expected immediate reward in interactive state $is = (s, q_i, q_j)$ for agent i as:

$$\bar{R}_i(s, q_i, q_j) = \sum_{a_i \in A_i} p(a_i|q_i) \sum_{a_j \in A_j} p(a_j|q_j) R_i(s, a_i, a_j), \tag{B.2}$$

and similarly for agent j.

In order to compute the average long-term reward, we derive the combined dynamics of the world and both agent's PDFCs as a matrix D, whose entries are indexed by (is, is') and represent

Appendix B (Continued)

the probability of transitioning from the interactive state $is = (s, q_i, q_j)$ to $is' = (s', q'_i, q'_j)$. We have:

$$D(is, is') = \sum_{a_i \in A_i} p(a_i|q_i) \sum_{a_j \in A_j} p(a_j|q_j) \sum_{s' \in S} p(s'|s, a_i, a_j) \\ \times \sum_{\omega_i \in \Omega_i} p(\omega_i|a_i, a_j, s') \sum_{\omega_j \in \Omega_j} p(\omega_j|a_i, a_j, s') \; \delta_K(q'_i, \tau^i_{q_i a_i \omega_i}) \; \delta_K(q'_j, \tau^j_{q_j a_j \omega_j}). \quad \text{(B.3)}$$

To compute the average reward for either agent, we obtain the stationary distribution of the Markov chain induced by D, denoted as π. As discussed in Appendix A, we need to be careful in case the chain so defined is not irreducible.

The average reward is then computed as the expectation of rewards with respect to π, that is simply the scalar product:

$$R_i^{avg} = \pi \bar{R}_i. \quad \text{(B.4)}$$

CITED LITERATURE

1. Russell, S. and Norvig, P.: Artificial Intelligence: A Modern Approach (3rd Edition). Prentice Hall, 3rd edition edition, December 2009.

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

3. Kadane, J. B. and Larkey, P. D.: Subjective probability and the theory of games. Management Science, 28(2):113–120, February 1982.

4. Yoshida, W., Dolan, R. J., and Friston, K. J.: Game theory of mind. PLoS Comput Biol, 4(12):e1000254+, December 2008.

5. Wright, J. R. and Leyton-Brown, K.: Behavioral game theoretic models: a Bayesian framework for parameter analysis. In International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012, Valencia, Spain, June 4-8, 2012 (3 Volumes), pages 921–930, 2012.

6. Gmytrasiewicz, P. J. and Doshi, P.: A framework for sequential planning in multi-agent settings. Journal of Artificial Intelligence Research, 24(1):49–79, July 2005.

7. Dennett, D. C.: Intentional systems. Journal of Philosophy, 68(February):87–106, 1971.

8. Gmytrasiewicz, P. J.: On reasoning about other agents. In Intelligent Agents II, Agent Theories, Architectures, and Languages, IJCAI '95, Workshop (ATAL), Montreal, Canada, August 19-20, 1995, Proceedings, pages 143–155, 1995.

9. eds. N. L. Hjort, C. Holmes, P. Müller, and S. G. Walker Bayesian Nonparametrics. Cambridge University Press, April 2010.

10. Sondik, E. J.: The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. Operations Research, 26(2):282–304, 1978.

11. Papadimitriou, C. and Tsitsiklis, J. N.: The complexity of Markov decision processes. Math. Oper. Res., 12(3):441–450, August 1987.

12. Spaan, M. T. J. and Vlassis, N.: Perseus: Randomized point-based value iteration for POMDPs. Journal of Artificial Intelligence Research, 24:195220, 2005.

13. Pineau, J., Gordon, G., and Thrun, S.: Point-based value iteration: an anytime algorithm for POMDPs. In Proceedings of the 18th international joint conference on Artificial intelligence, IJCAI'03, page 10251030, San Francisco, CA, USA, 2003. Morgan Kaufmann Publishers Inc.

14. Thrun, S.: Monte carlo POMDPs. In Advances in Neural Information Processing Systems (NIPS 1999), eds. S. A. Solla, T. K. Leen, and K.-R. Müller, pages 1064–1070. MIT Press, 2000.

15. Boutilier, C. and Poole, D.: Computing optimal policies for partially observable decision processes using compact representations. In Proceedings of the National Conference on Artificial Intelligence, pages 1168–1175, 1996.

16. Poupart, P.: Exploiting Structure to Efficiently Solve Large Scale Partially Observable Markov Decision Processes. Doctoral dissertation, University of Toronto, Toronto, Ont., Canada, Canada, 2005. AAINR02727.

17. Meuleau, N., Peshkin, L., Kim, K.-e., and Kaelbling, L. P.: Learning finite-state controllers for partially observable environments. In Proceedings of the 15th International Conference on Uncertainty In Artificial Intelligence, pages 427–436, 1999.

18. Poupart, P. and Boutilier, C.: Bounded finite state controllers. In Advances in Neural Information Processing Systems 16, 2003.

19. Ross, S., Pineau, J., Paquet, S., and Chaib-draa, B.: Online planning algorithms for pomdps. J. Artif. Int. Res., 32(1):663–704, July 2008.

20. Silver, D. and Veness, J.: Monte-Carlo planning in large POMDPs. In Advances in Neural Information Processing Systems 23, eds. J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, pages 2164–2172. Curran Associates, Inc., 2010.

21. Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S.: A survey of monte carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in Games, 4:1–43, 03/2012 2012.

22. Kocsis, L. and Szepesvári, C.: Bandit based Monte-Carlo planning. In Proceedings of the 17th European Conference on Machine Learning, ECML'06, pages 282–293, Berlin, Heidelberg, 2006. Springer-Verlag.

23. de la Higuera, C.: Grammatical Inference: Learning Automata and Grammars. New York, NY, USA, Cambridge University Press, 2010.

24. Oncina, J., García, P., and Vidal, E.: Learning subsequential transducers for pattern recognition interpretation tasks. IEEE Trans. Pattern Anal. Mach. Intell., 15(5):448–458, May 1993.

25. Balle, B., Quattoni, A., and Carreras, X.: A spectral learning algorithm for finite state transducers. In Machine Learning and Knowledge Discovery in Databases, eds. D. Gunopulos, T. Hofmann, D. Malerba, and M. Vazirgiannis, volume 6911 of Lecture Notes in Computer Science, pages 156–171. Springer Berlin Heidelberg, 2011.

26. Doshi, P. and Gmytrasiewicz, P. J.: On the difficulty of achieving equilibrium in interactive POMDPs. In Proceedings of the 21st National Conference on Artificial Intelligence - Volume 2, AAAI'06, pages 1131–1136. AAAI Press, 2006.

27. Doshi, P. and Gmytrasiewicz, P. J.: Monte Carlo sampling methods for approximating interactive POMDPs. Journal of Artificial Intelligence Research, 34(1):297–337, March 2009.

28. Sonu, E. and Doshi, P.: Generalized and bounded policy iteration for interactive POMDPs. In International Symposium on Artificial Intelligence and Mathematics (ISAIM), 2012.

29. Doshi, P. and Perez, D.: Generalized point based value iteration for interactive POMDPs. In AAAI, eds. D. Fox and C. P. Gomes, pages 63–68. AAAI Press, 2008.

30. Zeng, Y. and Doshi, P.: Exploiting model equivalences for solving interactive dynamic influence diagrams. J. Artif. Int. Res., 43(1):211–255, January 2012.

31. Brooks, S. P.: Markov Chain Monte Carlo Method and Its Application. Journal of the Royal Statistical Society. Series D (The Statistician), 47(1):69–100, 1998.

32. Hastings, W. K.: Monte Carlo sampling methods using Markov chains and their applications. Biometrika, 57(1):97–109, 1970.

33. Chib, S. and Greenberg, E.: Understanding the Metropolis-Hastings Algorithm. The American Statistician, 49(4):327–335, November 1995.

34. Geman, S. and Geman, D.: Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. IEEE Trans. Pattern Anal. Mach. Intell., 6(6):721–741, November 1984.

35. Ferguson, T. S.: A Bayesian analysis of some nonparametric problems. Ann. Statist., 1(2):209–230, 03 1973.

36. Sethuraman, J.: A constructive definition of Dirichlet priors. Statistica Sinica, 4:639–650, 1994.

37. Aldous, D. J.: Exchangeability and related topics. In cole d't de Probabilits de Saint-Flour XIII 1983, ed. P. Hennequin, volume 1117 of Lecture Notes in Mathematics, pages 1–198. Springer Berlin Heidelberg, 1985.

38. Escobar, M. D. and West, M.: Bayesian density estimation and inference using mixtures. Journal of the American Statistical Association, 90:577–588, 1994.

39. Neal, R. M.: Markov chain sampling methods for Dirichlet process mixture models. Journal of Computational and Graphical Statistics, 9(2):pp. 249–265, 2000.

40. Antoniak, C. E.: Mixtures of Dirichlet processes with applications to Bayesian nonparametric problems. The Annals of Statistics, 2(6):1152–1174, November 1974.

41. Shoham, Y. and Leyton-Brown, K.: Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations. New York, NY, USA, Cambridge University Press, 2008.

42. Brown, G. W.: Iterative solutions of games by fictitious play. In Activity Analysis of Production and Allocation, pages 374–376. Wiley, 1951.

43. Fudenberg, D. and Levine, D. K.: The theory of learning in games. MIT press series on economic learning and social evolution. Cambridge (Mass.), London, the MIT press, 1998.

44. Kalai, E. and Lehrer, E.: Rational learning leads to Nash equilibrium. ECONOMETRICA, 61(5):1019–1045, 1993.

45. Albrecht, S. V., Crandall, J. W., and Ramamoorthy, S.: Belief and truth in hypothesised behaviours. CoRR, abs/1507.07688, 2015.

46. Carmel, D. and Markovitch, S.: Learning models of intelligent agents. In Proceedings of the 13th National Conference on Artificial intelligence, pages 62–67, 1996.

47. Carmel, D. and Markovitch, S.: Model-based learning of interaction strategies in multi-agent systems. J. Exp. Theor. Artif. Intell., 10(3):309–332, 1998.

48. Powers, R. and Shoham, Y.: Learning against opponents with bounded memory. In Proceedings of the 19th International Joint Conference on Artificial Intelligence, IJCAI'05, pages 817–822, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc.

49. Chakraborty, D. and Stone, P.: Online multiagent learning against memory bounded adversaries. In Machine Learning and Knowledge Discovery in Databases, European Conference, ECML/PKDD 2008, Antwerp, Belgium, September 15-19, 2008, Proceedings, Part I, pages 211–226, 2008.

50. Littman, M. L.: Markov games as a framework for multi-agent reinforcement learning. In Proc. 11th International Conference on Machine Learning, pages 157–163. Morgan Kaufmann, 1994.

51. Bowling, M. and Veloso, M.: Multiagent learning using a variable learning rate. Artificial Intelligence, 136:215–250, 2002.

52. Conitzer, V. and Sandholm, T.: Awesome: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. Machine Learning, 67(1-2):23–43, 2007.

53. Mccallum, A. K.: Reinforcement Learning with Selective Perception and Hidden State. Doctoral dissertation, The University of Rochester, 1996.

54. Ross, S., Draa, B. C., and Pineau, J.: Bayes-adaptive POMDPs. In Proc.˜of the Conference on Neural Information Processing Systems, 2007.

55. Liu, M., Liao, X., and Carin, L.: The infinite regionalized policy representation. In Proceedings of the 28th International Conference on Machine Learning, eds. L. Getoor and T. Scheffer, pages 769–776, 2011.

56. Doshi-Velez, F., Pfau, D., Wood, F., and Roy, N.: Bayesian nonparametric methods for partially-observable reinforcement learning. IEEE Transactions on Pattern Analysis and Machine Intelligence, 99(PrePrints):1, 2013.

57. Pfau, D., Bartlett, N., and Wood, F.: Probabilistic deterministic infinite automata. In Advances in Neural Information Processing Systems, pages 1930–1938, 2010.

58. Liu, M., Amato, C., Liao, X., Carin, L., and How, J. P.: Stick-breaking policy learning in Dec-POMDPs. In Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015, pages 2011–2018, 2015.

59. Celeux, G., Hurn, M., and Robert, C. P.: Computational and inferential difficulties with mixture posterior distributions. Journal of the American Statistical Association, 95(451):957–970, 2000.

60. Jain, S. and Neal, R. M.: A split-merge markov chain Monte Carlo procedure for the Dirichlet process mixture model. Journal of Computational and Graphical Statistics, 13(1):pp. 158–182, 2004.

61. Jain, S. and Neal, R. M.: Splitting and merging components of a nonconjugate Dirichlet process mixture model. Bayesian Anal., 2(3):445–472, 09 2007.

62. Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. B.: Bayesian Data Analysis, Second Edition. Chapman and Hall/CRC, 2 edition edition, July 2003.

63. Lovejoy, W. S.: Computationally feasible bounds for partially observed markov decision processes. Operations Research, 39(1):162–175, 1991.

64. Rabiner, L. R.: A tutorial on hidden Markov models and selected applications in speech recognition. In Proceedings of the IEEE, pages 257–286, 1989.

65. Shmueli, G.: To explain or to predict? Statist. Sci., 25(3):289–310, 08 2010.

66. Choi, J. and Kim, K.-E.: Inverse reinforcement learning in partially observable environments. Journal of Machine Learning Research, 12:691–730, March 2011.

67. Polich, K. and Gmytrasiewicz, P.: Interactive dynamic influence diagrams. In Proceedings of the 6th International Joint Conference on Autonomous Agents

and Multiagent Systems, AAMAS '07, pages 34:1–34:3, New York, NY, USA, 2007. ACM.

68. Lopes, H., Carvalho, C. M., Johannes, M. S., and Polson, N. G.: Particle learning for sequential Bayesian computationn. In Bayesian Statistics 9, eds. J. M. Bernardo, M. J. Bayarri, J. O. Berger, A. P. Dawid, D. Heckerman, S. A. F. M., and M. West, pages 317–360. Oxford University Press, 2011.

69. Panella, A. and Gmytrasiewicz, P.: Learning policies of agents in partially observable domains using Bayesian nonparametric methods. In AAMAS Workshop on Multiagent Sequential Decision Making Under Uncertainty, 2014.

70. Ng, A. Y. and Russell, S.: Algorithms for inverse reinforcement learning. In Proceedings of the 17th International Conference on Machine Learning, pages 663–670. Morgan Kaufmann, 2000.

71. Ramachandran, D. and Amir, E.: Bayesian inverse reinforcement learning. Proceedings of the 20th International Joint Conference on Artical Intelligence, 51:2586–2591, 2007.

72. Ziebart, B. D., Maas, A., Bagnell, J. A., and Dey, A. K.: Maximum entropy inverse reinforcement learning. In Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3, AAAI'08, pages 1433–1438. AAAI Press, 2008.

VITA

NAME	Alessandro Panella
EDUCATION	B.A., Dipartimento di Elettronica e Informazione, Politecnico di Milano, 2006
	M.S., Department of Computer Science, University of Illinois at Chicago, 2008
	M.S., Dipartimento di Elettronica e Informazione, Politecnico di Milano, 2009
	Ph.D., Department of Computer Science, University of Illinois at Chicago, 2016
WORK EXPERIENCE	Research Fellow, Computation Institute at the University of Chicago (June 2014 - June 2015)
	Teaching Assistant, Department of Computer Science, University of Illinois at Chicago (September 2013 - May 2014)
	Fellow, Data Science for Social Good, University of Chicago (June 2013 - August 2013)
	Research Assistant, Department of Computer Science, University of Illinois at Chicago (January 2009 - May 2013)
PUBLICATIONS	**A. Panella**, P. Gmytrasiewicz. "Bayesian Learning of Other Agents Finite Controllers for Interactive POMDPs", AAAI International Conference on Artifical Intelligence (AAAI-16), Phoenix, Arizona, Feb. 2016 (to appear.)
	A. Panella, P. Gmytrasiewicz. "Nonparametric Bayesian Learning of Other Agents' Policies in Multiagent POMDPs", *International Conference on Autonomous Agents and Multiagent Systems* (Extended Abstract), Istanbul, Turkey, May 2015.

Charlie Catlett, Tanu Malik, Brett Goldstein, Jonathan Giuffrida, Yetong Shao, **Alessandro Panella**, Derek Eder, Eric van Zanten, Robert Mitchum, Severin Thaler, Ian T. Foster. "Plenario: An Open Data Discovery and Exploration Platform for Urban Science." *IEEE Data Eng. Bull.* 37(4): 27-42 (2014)

F. Cafaro, **A. Panella**, L. Lyons, J. Roberts, J. Radinsky. "I See You There! Developing Identity-Preserving Embodied Interaction for Museum Exhibits." *CHI Conference on Human Factors in Computing Systems (CHI 2013).* Paris, France, May 2013.

A. Panella, P. Gmytrasiewicz. "A Partition-Based First-Order Probabilistic Logic to Represent Interactive Beliefs." *5th International Conference on Scalable Uncertainty Management (SUM 2011*), pp. 233-246. Dayton, OH, Oct. 2011.

A. Panella, M.D. Santambrogio, F. Redaelli, F. Cancar, D. Sciuto. "A Design Workflow for Dynamically Reconfigurable Multi-FPGA Systems." *18th IEEE/IFIP International Conference on Very Large Scale Integration (VLSI-SoC 2010),* pp. 414-419. Madrid, Spain, Sept. 2010.

M. Murgida, **A. Panella**, V. Rana, M.D. Santambrogio, and D. Sciuto. "Fast IP-Core Generation in a Partial Dynamic Reconfiguration Workflow." *14th IFIP International Conference on Very Large Scale Integration (VLSI-SOC 2006*), pp. 74-79. Nice, France, Oct. 2006.