Auton Agent Multi-Agent Syst (2017) 31:861–904
DOI 10.1007/s10458-016-9359-z

Interactive POMDPs with finite-state models of other agents

Alessandro Panella[1] · **Piotr Gmytrasiewicz**[1]

Published online: 25 January 2017

Abstract We consider an autonomous agent facing a stochastic, partially observable, multi-agent 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, in which the agent does not explicitly model the other agents' beliefs and preferences, and instead represents them 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 belief 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 algorithm that approximates the posterior beliefs over the other agents' PDFCs, given a sequence of (possibly imperfect) observations about their behavior. Experimental results show that the learned models converge behaviorally to the true ones. 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 that 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. Furthermore, we demonstrate how an agent can exploit the learned models to perform indirect inference over the state of the environment via the modeled agent's actions.

✉ Alessandro Panella
apanel2@uic.edu

Piotr Gmytrasiewicz
piotr@uic.edu

1 Department of Computer Science, University of Illinois at Chicago, Chicago, IL, USA

Keywords Multiagent systems · Stochastic planning · Opponent modeling

1 Introduction

An autonomous, rational agent operating in a stochastic and partially observable environment acts according to a plan that aims at maximizing its expected utility with respect to its belief, which is the agent's subjective probability distribution over the current state of the environment. The problem of finding the optimal policy can be formulated mathematically as a partially observable Markov decision process (POMDP) [33,59]. Due to uncertainty in both the environment's transitions and the agent's observations, POMDPs are notoriously hard to solve.

Yet an additional layer of complexity is introduced when there are other agents that also influence the environment with their own actions. 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 paper, 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 [32,61,62]. In particular, we propose an approach to multiagent stochastic planning that is a specialization of interactive POMDPs (I-POMDPs) [23]. 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 previous research on I-POMDPs, it has been predominant to consider *intentional* models [12,22] of other agents, which consist of 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 use *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. In this scenario, the agent's predictions of the other agents' actions can only be 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. Intentional modeling, on the other hand, relies on the fact that other agents are rational, since their actions are predicted by assuming that they themselves 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 (PDFC) 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, their beliefs are never represented explicitly.

In line with the decision-theoretic approach to planning, 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 possible PDFCs, 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 *Bayesian nonparametric* prior distribution [29]. Note that the term "nonparametric" does not indicate in this case the absence of parameters, but the 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.)

Performing Bayesian inference using nonparametric distributions is not amenable to analytical solutions, requiring the use of approximate methods. In this work, we develop a Markov chain Monte Carlo algorithm (MCMC) that learns an ensemble of PDFCs that approximates the posterior distribution given a sequence of observed behavior. Note that the inference procedure must deal with partial observability of the modeled agent's behavior, since in I-POMDPs another agent's actions may not be perfectly observable.

This also implies that the modeling agent might not be able to discover, even at the infinite limit, the true type of the other agent. This is because the learning agent is able to discern between different models only to the extent that they induce different distributions over its observations. In other words, learning can be only as good as the agent's available data. Moreover, the true *type* of the other agent may not belong to the class of PDFCs in the first place; however, we show in our experiments that even in this case the modeling agent is able to encode the regularities in the behavior of the other agent as PDFCs, and hence exploit them to increase its rewards.

We consider two scenarios of interaction. In the first, the modeling agent uses past observations to learn the other agents' models, and then exploits the inferred PDFCs by embedding them into its own decision making process, according to the I-POMDP framework. The second, more realistic scenario sees the agent interleaving learning and planning during interaction, using a finite window of past observations. This allows the agent to cope with opponents that are themselves adapting, perhaps by also learning the agent's model in return.

Experimental results show that our learning algorithm is able to infer PDFCs that converge behaviorally to the true agents' controllers as the size of the training data increases, under ideal observability conditions. Even when another agent's behavior is only partially observable, we show that the modeling agents' performance increases when exploiting the learned PDFCs. This continues to be true when the other agent's true policy is not implemented as a finite controller, but is instead computed online. Moreover, we analyze, for an example domain, the dynamics that emerge when two agents are simultaneously planning and learning about each other. In particular, we show that cooperation arises naturally when the two agents have compatible utility functions. Throughout our results, it can be observed that the size of the learned PDFCs reflects the complexity of the behavior of the modeled agent; this validates the use of a Bayesian nonparametric prior, that naturally embodies the desirable Occam's razor property of learning.

The remainder of this paper is organized as follows. Section 2 contextualizes our approach with respect to existing work, and introduces some necessary background concepts. In Sect. 3 we describe the nonparametric prior distribution over the space of PDFCs, while in Sect. 4 we present the MCMC algorithm that implements Bayesian learning, given this prior. Section 5 formally introduces interactive POMDPs with PDFC models and describes how the learned PDFCs are in practice embedded in such framework. In Sect. 6 we provide the experimental results that validate our methodology. Section 7 concludes this paper by summarizing the findings and providing hints for future research.

2 Context and background

This section introduces concepts and existing methods that are related to the work described later in this paper. Along with these descriptions, we expand on two important dichotomies that characterize different approaches to opponent modeling in multiagent settings, namely between *implicit* and *explicit* modeling, and between *intentional* and *subintentional* modeling.

2.1 Related work: model-based multiagent learning

The concept of learning in multiagent systems is complex and multi-faceted [57], 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 earliest approaches to opponent's policy modeling is *fictitious play* [5]: 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 studied extensively, with a number of theoretical results especially in relation to Nash equilibrium [20]. 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 more general learning method that applies to the same setting is *rational learning* [34]. Given a set of possible strategies Σ_j for agent j, and a set of possible histories of a game Z, the probability that i ascribes to j's strategy $\sigma_j \in \Sigma_j$, given a history $z \in Z$, is derived by applying Bayes' rule:

$$p(\sigma_j|z) = \frac{p(z|\sigma_j)\ p(\sigma_j)}{\int_{\Sigma} p(z|\sigma_j) p(\mathrm{d}\sigma_j)}. \tag{1}$$

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 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. This method has vast theoretical merits, but if the set of strategies Σ is large, infinite, or even uncountable, there is little hope that Eq. 1 can be implemented analytically.

The interactive POMDP model (I-POMDP) [23], which is described in detail in Sect. 2.2, is a form of rational learning. Importantly, it extends rational learning to sequential, stochastic, partially observable settings. In [13], the authors generalize the subjective equilibrium convergence of rational learning to cover this more general setting, by restating the grain of truth condition in terms of probability over future sequences of observations. However, they also notice that 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 [1], that considers environments that evolve stochastically (but are perfectly observable,) in which the modeling agent maintains a belief over a finite set of types of other agents.

In [6], the authors propose the use of deterministic finite automata (DFA) for opponent modeling, and define a heuristic algorithm that derives a DFA consistent with past plays of the game. This approach assumes that the environment is a (single-state) repeated game with perfect monitoring, without allowing the modeled agent to be itself adaptive.

One approach to multiagent learning that aims at explicitly modeling adapting opponents is the one in [52]. As before, the authors focus on repeated games with perfect monitoring, and establish a series of desirable properties for opponent modeling. The algorithm they provide targets a class of opponents whose policy only depends on a history of finite length. The work in [8] 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.

We can recognize commonalities between some of the related work surveyed above and the approach we propose in this paper. Like [34] and related methods, we also use Bayesian update as the basis of learning. For obvious reasons, our work is related to the analysis of convergence in I-POMDPs [13], and in particular shares the same generality about the type of environment considered. A trait in common with [6] is the use of finite-state models of other agents. The approaches in [52] and [8] consider models of opponents whose actions are based on a finite history, and could therefore be modeled by a finite state controller, such as in our approach.

2.1.1 Discussion: 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* [4,10,36]. 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 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 $g_j : \Omega_i^* \times A_i^* \rightarrow A_j$ 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 just on the current state of the world s, 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 [40], Bayes-adaptive POMDPs [55], infinite generalized policy representation [38], infinite POMDPs [17], 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.

2.2 Interactive POMDPs

The interactive partially observable Markov decision process (I-POMDP) [23] is an extension of the POMDP framework [33,59] to multiagent settings. A distinguishing feature of I-POMDPs is that each agent maintains full autonomy, and plans in isolation according to its own subjective belief and preferences. This sets I-POMDPs apart from purely cooperative approaches to multiagent stochastic planning where the agents share initial knowledge and goals, such as decentralized POMDPs (DEC-POMDPs) [3].

In the following and in the rest of this paper, we consider an environment containing two agents, namely agent i and agent j. In its general form [13], an I-POMDP for agent i is defined as a 6-tuple:

$$I\text{-}POMDP_i = (IS_i, A, \Omega_i, T_i, O_i, R_i), \tag{2}$$

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 $m_j = (O_j, z_j, f_j)$, where $O_j \in \mathcal{O}_j$ is an observation function,[1] $z_j \in Z_j$ is a history of j's actions and observations, i.e. $z_j = \omega_j^1 a_j^1 \omega_j^2 a_j^2 \ldots$, and $f_j \in F_j$ is an agent function of the form $f_j : Z_j \rightarrow \Delta(A_j)$.
- $A = A_i \times A_j$ is the set of joint actions for the two agents.
- Ω_i is the set of observations for agent i.
- $T_i : S \times A \times S \rightarrow [0, 1]$ 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 agent execute actions (a_i, a_j) when the environment is in state s.
- $O_i : A \times S \times \Omega_i \rightarrow [0, 1]$ is the stochastic observation function. $O_i(a_i, a_j, s', \omega_i) = p(\omega_i|a_i, a_j, s')$ corresponds to the probability that agent i receives observation ω_i, given that the joint action (a_i, a_j) was executed and the resulting state is s'.
- $R_i : S \times A \rightarrow \mathbb{R}$ is the reward function. $R_i(s, a_i, a_j)$ is the payoff to agent i when joint action (a_i, a_j) is executed in state s.

Similarly to single-agent POMDPs, any history can be summarized with a belief over the interactive state space, denoted as b_i, that can be updated by applying Bayes' rule as follows, given the current belief, action a_i, and observation ω_i:

$$\begin{aligned} b_i', (is_i') &= p(is_i'|a_i, \omega_i, b_i) \\ &= \beta \sum_{\substack{is_i \in IS_i \text{ s.t.} \\ (O_j', f_j') = (O_j, f_j)}} b_i(is_i) \sum_{a_j \in A_j} f_j(z_j, a_j)\ O_i(a_i, a_j, s', \omega_i)\ p(is_i'|is_i, a_i, a_j) \end{aligned} \tag{3}$$

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

where β is a normalization constant. The term $p(is'_i|is_i, a_i, a_j)$ is the *interactive transition model*, defined as:

$$p(is'_i|is_i, a_i, a_j) = T_i(s, a_i, a_j, s') \sum_{\omega_j \in \Omega_j} O_j(a_i, a_j, s', \omega_j)\ \delta(z'_j, z_j a_j \omega_j). \tag{4}$$

Here and in the rest of this paper, the symbol δ indicates the Kronecker delta function, that assumes value 1 when its two arguments are the same, and 0 otherwise. The term $z_j a_j w_j$ refers to the history obtained by concatenating the previous history z_j with a_j and w_j.

2.2.1 Intentional I-POMDPs

The definition above is mathematically sound, but of little practical use if we do not restrict the set of possible agent functions F_j. Considering every possible computable mapping from H_j to A_j is not viable. In addition to the obvious impracticality of such approach, there are actual theoretical limits that prevent the set F_j to be so general, as described in [13]. In fact, the original definition of I-POMDP introduced in [23] limits F_j by considering the set of agent functions that can themselves be implicitly encoded as I-POMDPs, thus taking an intentional stance [12]. We call this approach *intentional I-POMDP*.[2]

This is formalized by considering the models of the other agent that consist of an I-POMDP tuple and a belief state that summarizes its past history, that is:

$$m_j = (\ (IS_j, A, \Omega_j, T_j, O_j, R_j), b_j) = (\hat{m}_j, b_j), \tag{5}$$

where b_j is a probability distribution over j's interactive state space IS_j, that summarizes its past history z_j. This definition of intentional model is consistent with the game-theoretical notion of *type* in the context of Bayesian games [26], that refers to the set of agent j's private information that are involved in its decision making process. In the notation above, the element $\hat{m}_j$ is referred to as the *frame* of agent j, that is assumed to be static.

As mentioned in the introduction, agent i's intentional stance can be reciprocated by agent j: specifically, $IS_j = S \times M_j$, where $m_i \in M_i$ is defined as in Eq. 5 after swapping the subscripts i and j. Clearly, this triggers an infinite recursion of beliefs, that must be truncated at a finite arbitrary level l.

In this case, the augmented notation $IS_{i,l} = S \times M_{j,l-1}$ refers to the interactive state space for agent i at level l. The recursion ends at level 0, with $IS_{i,0} = S$.[3] Accordingly, the belief update formula in Eq. 3 specialized for intentional models becomes:

$$\begin{aligned} b'_{i,l}(is'_{i,l}) &= p(is'_{i,l}|a_i, \omega_i, b_{i,l}) \\ &= \beta \sum_{\substack{is_{i,l} \in IS_{i,l} \text{ s.t.} \\ \hat{m}'_{j,l-1} = \hat{m}_{j,l-1}}} b_{i,l}(is_{i,l}) \sum_{a_j \in A_j} p(a_j|m_{j,l-1})\ O_i(a_i, a_j, s', \omega_i)\ p(is'_{i,l}|is_{i,l}, a_i, a_j), \end{aligned} \tag{6}$$

where the term $p(a_j|m_{j,l-1})$ represents the probability of agent j executing action a_j given that it is *Bayes rational* (i.e. optimizes the sum of expected rewards) and has type $m_{j,l-1}$.

[2] Strictly speaking, the intentional I-POMDP formalization in [23] 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.

[3] In this formulation, we assume that at level 0 the behavior of the other agent is folded into the world state's transition function as noise; in general, it can be encoded in a more complex subintentional model.

The term $p(is'_{i,l}|is_{i,l}, a_i, a_j)$ is in this case defined as:[4]

$$p(is'_{i,l}|is_{i,l}, a_i, a_j) \\ = T_i(s, a_i, a_j, s') \sum_{\omega_j \in \Omega_j} O_j(a_i, a_j, s', \omega_j)\, \delta(b'_{j,l-1}, \kappa(b_{j,l-1}, a_j, \omega_j)), \tag{7}$$

where $\kappa(b_{j,l-1}, a_i, \omega_i)$ is the *belief transition function*, that computes j's new belief by applying Eq. 6 over $IS_{j,l-1}$.

In the equations above, retrieving the terms $p(a_j|m_{j,l-1})$ and $\kappa(b_{j,l-1}, a_j, \omega_j)$ triggers recursions into j's own decision making process and belief update, that in turn might involve solving i's I-POMDP from j's perspective, and so on down to level 0. Therefore, solving an intentional I-POMDP at any level of nesting is at least as complex as solving a single-agent POMDP, which is itself known to be in general PSPACE-complete [47]. On top of this, there exist a theoretical limitation: while an agent may wish to maintain a belief over every possible belief of the other agent, it can be shown [14] that it is impossible to have a probability distribution over every possible belief for nesting level larger than two.

Nevertheless, several approximate algorithms have been proposed to solve intentional I-POMDPs, among which Monte Carlo methods [14], policy iteration [60], point-based value iteration [15], and nested dynamic influence diagrams [63].

2.2.2 Discussion: intentional and subintentional I-POMDPs

In this section, we highlight some of the pros and cons related to picking an intentional or a subintentional stance in I-POMDPs. Ultimately, we believe that neither approach is absolutely preferable to the other. Given the problem at hand, one should be able to weight in the advantages of either method. Furthermore, in some cases, a combination of the two might be the best solution.

– *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 with respect to the prior knowledge is available to the agent, practical implementations always assume knowledge about the frame of the other agent, 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. On the other hand, subintentional models do not require such assumption.
– *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. 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 for scenarios in which the reward model of the other agent is known.
– *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

[4] Here and in the remainder of this paper, δ denotes the Kronecker delta function, that is equal to 1 if its arguments are equal, 0 otherwise.

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 modeled agent's own model of 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 [9]. We leave the exploration of such possibility for future work.

2.3 Probabilistic deterministic finite-state controllers

Finite-state controllers (FSC) represent in some cases the optimal solution to a POMDP [33]; in fact, some algorithms search directly for a solution in the space of finite state controllers [25,41,51]. Hence, our choice of modeling an observed agent's behavior as a finite state controller is perhaps not surprising. We consider a special case of FSCs called probabilistic deterministic finite-state controllers (PDFCs). A PDFC is a 6-tuple:

$$c = (\Omega, A, Q, \tau, \theta, q^1), \tag{8}$$

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.
- $\tau : Q \times A \times \Omega \to Q$ 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 $\tau_{qa\omega}$. In this paper, we use the term "transition" and "edge" interchangeably to refer to a single element $\tau_{qa\omega}$.
- $\theta : Q \times A \to [0, 1]$ 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}. The notation θ_q denotes the probability vector $(\theta_{q1}, \theta_{q2}, \ldots, \theta_{q|A|})$.
- $q^1 \in Q$ is the starting node.

It is straightforward to see how a PDFC can implement an agent function: conditional on the history[5] $z^{1:T} = (a^{1:T}, \omega^{1:T})$, the current PDFC node is defined recursively for $t = 1, 2, \ldots$, starting from q^1, as $q^t = \tau(q^{t-1}, a^{t-1}, \omega^t)$. From q^t, the agent generates an action using the probability distribution θ_{q^t}. 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 $\omega^{1:T}$, a PDFC induces a probability distribution over the action sequence $a^{1:T}$ defined as follows:

$$p(a^{1:T}|\omega^{1:T}) = \theta(q^1, a^1) \prod_{t=2:T} \sum_{q \in Q} \delta(q, \tau(q^{t-1}, a^{t-1}, \omega^t))\, \theta(q, a^t). \tag{9}$$

PDFCs are related to transducers in the context of grammar learning [28]. In [45], the authors propose a heuristic algorithm for learning deterministic transducers based on state-merging moves. More recent work [2] attempts at learning stochastic transducers using spectral methods.

5 Here and in the rest of this paper, the notation $x^{1:t}$ indicates the sequence $(x^1, x^2, \ldots, x^t)$. Sometimes, a condensed notation is used for two or more sequences, i.e. $(x, y)^{1:t} \triangleq (x^{1:t}, y^{1:t})$

2.4 Partially observable Monte Carlo planning for POMDPs

We briefly describe in this subsection the POMCP algorithm [58] for planning in (single-agent) POMDPs, since it is the method that we adapt to solve I-POMDPs with subintentional models (Sect. 5).

Traditional online POMDP planners work by executing a "full-width" forward search of the 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, an issue that is sometimes defined as the *curse of history* [49]. In contrast, *partially observable Monte Carlo planning* (POMCP) is an online stochastic 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 exploration towards 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 [35] 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 agents 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.

3 A prior distribution for PDFCs

This section formally introduces the Bayesian learning problem we want to solve, and describes what features are desirable for a prior probability distribution over PDFCs. The specific prior probability that we adopt is subsequently described, along with some of its features and possible alternatives.

3.1 The need of a suitable prior

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, given an observed trajectory (or history) $z_i^{1:T}$. We want to compute the posterior distribution:

$$p(c_j|z_i^{1:T}) \propto p(z_i^{1:T}|c_j)\; p(c_j). \tag{10}$$

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 the definition of PDFC that each node k is associated to a continuous parameter $\theta_k \in \Delta(A_j)$. Therefore, the set of PDFCs (of unbounded size) is infinite-dimensional. Bayesian nonparametric (BNP) techniques have increasingly been used in recent years to design priors over infinite-dimensional spaces, such as the case of Dirichlet process mixture models (DPMMs) [19] and variations thereof [29]. In the remainder of this section, we describe the BNP prior distribution over PDFCs that we adopt in our work.

3.2 Prior distribution over the transition function

In order to have a less cluttered notation, we will refer to j's PDFC states, observations, and actions using numerical indexes. First, let us introduce the quantities $K = |Q_j|$, the number of nodes of j's PDFC; $G = |A_j|$, the cardinality of j's action set; $H = |\Omega_j|$, the cardinality of j's observation set. Now, let $k = 1, 2, \ldots, K$ represent a specific value that the variable q_j can assume; similarly, let $g = 1, 2, \ldots, G$ represent the value of a_j, and $h = 1, 2, \ldots, H$ the value of ω_j. Using this notation, τ_{kgh} refers to the destination node $\tau(q_j = k, a_j = g, \omega_j = h)$ reached when action g is executed in node k, and observation h is perceived. In general, the number of nodes K is unknown and unbounded; therefore, k is also unbounded.

For each value k of the starting node, action g, and observation h, the destination node τ_{kgh} is drawn from an infinite discrete probability vector $\pi = (\pi_1, \pi_2, \ldots)$. Intuitively, the component π_k can be interpreted as the tendency of node k to attract incoming transitions.

The vector π is in turn drawn from a distribution called *stick-breaking process*, with parameter α, defined as follows:

$$\pi_k = \mu_k \prod_{h=1}^{k-1} (1 - \mu_h), \qquad \text{where } \mu_k \sim \text{Beta}(1, \alpha). \tag{11}$$

This distribution is traditionally denoted as GEM(α).[6] The components $\pi_1, \pi_2, \ldots$ can be thought as being generated by the following intuitive process. Imagine to have a stick of initial length 1, and brake it at a point selected according to Beta$(1, \alpha)$. The length of the left chunk is π_1. Now, we break the right chunk by using the same procedure, obtaining π_2, and so on infinitely. Summarizing, we have:

$$\begin{aligned} \pi \mid \alpha &\sim \text{GEM}(\alpha) \\ q^1 \mid \pi &\sim \text{Discrete}(\pi) \\ \tau_{kgh} \mid \pi &\sim \text{Discrete}(\pi) \qquad k = 1..\infty; \quad g = 1..G; \quad h = 1..H \end{aligned} \tag{12}$$

Figure 1 represents this hierarchical prior as a Bayesian network.

An equivalent construction of this hierarchical distribution is obtained by assigning the PDFC transitions sequentially, drawing from the following distribution:

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

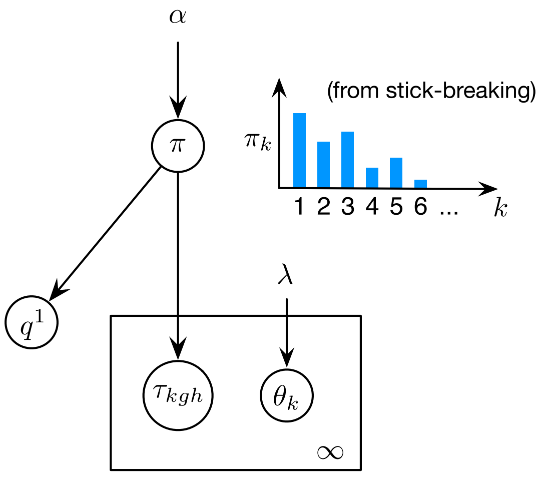


Fig. 1 Graphical model of the nonparametric prior over PDFCs. In evidence is the fact that π is an infinite discrete probability vector

$$\begin{aligned} &p(\tau_{kgh} = k' | \alpha, \tau_{<kgh}) \propto v_{k'} && \text{if node } k' \text{ has been previously chosen} \\ &p(\tau_{kgh} = \bar{k}' | \alpha, \tau_{<kgh}) \propto \alpha && \text{for "new" node } \bar{k}', \end{aligned} \tag{13}$$

where $\tau_{<kgh}$ are the values of the edges that have been sampled prior to τ_{kgh}, and $v_{k'}$ is a count statistics representing the number of edges in $\tau_{<kgh}$ that "already" point to node k'. Note that the vector π does not appear explicitly in this characterization, and is in fact integrated out.

To exemplify this construction, also known as the "Chinese Restaurant Process" (CRP), consider an agent j with $G = 2$ actions and $H = 2$ observations. First, we draw the initial node q^1. Since there is no node currently instantiated, the probability in Eq. 13 reduces to $p(q^1 = 1|\alpha) = \alpha/\alpha = 1$. Note that "1" is just the numerical label assigned to the node. Now, for this first node, we need to sample the $HG = 4$ outgoing transitions. Proceeding in order, we first sample the transition for $g = 1$ and $h = 1$. The probability for τ_{111} to be node 1 itself is $1/(1 + \alpha)$, whereas the probability of going to a newly instantiated node (that we label as 2) is $\alpha/(1 + \alpha)$. Let us assume that, in this particular instance, it so happens that we draw $\tau_{111} = 1$. Now, the transition τ_{112} will have probability $2/(2 + \alpha)$ of being assigned to node 1 itself, and probability $\alpha/(2 + \alpha)$ of going to a new node. Let us assume that in this case $\tau_{112} = 2$; the next transition to be sampled, τ_{121}, will be drawn with the following probabilities:

$$\begin{aligned} p(\tau_{121} = 1|\alpha, \tau_{111}, \tau_{112}) &= \frac{2}{3+\alpha} \\ p(\tau_{121} = 2|\alpha, \tau_{111}, \tau_{112}) &= \frac{1}{3+\alpha} \\ p(\tau_{121} = 3|\alpha, \tau_{111}, \tau_{112}) &= \frac{\alpha}{3+\alpha} \text{ (new node.)} \end{aligned} \tag{14}$$

This process continues until all transitions have been assigned and no new nodes are generated.

The prior distribution over the PDFC transitions so defined induces a "rich get richer" property to the nodes of the PDFC: transitions are more likely to point to nodes that already receive many incoming transitions. This self-reinforcing 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 stick-breaking prior reflects the fact that some nodes represent more commonly visited "mental states" for the agent, such as reset states.

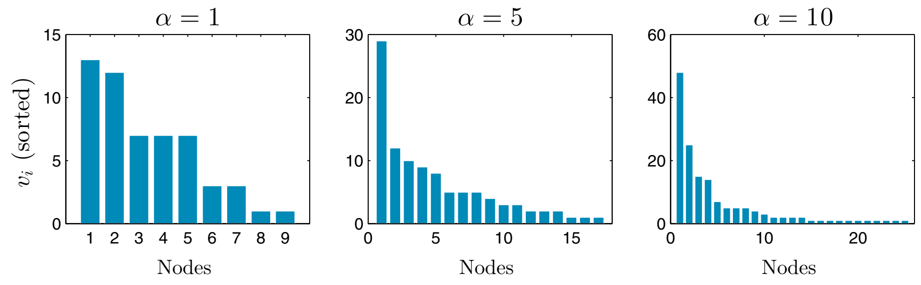


Fig. 2 Number of incoming transition per node, sorted, with different values of α

The histograms in Fig. 2 depict the number of incoming transitions v_i (sorted in descending order) for the nodes of a PDFC, sampled according to the prior distribution above, for different values of α, and $|A_j| = G = 3$, $|\Omega_j| = H = 2$. We can see that, for larger values of α, more nodes are generated, which is expected since α is proportional to the probability of instantiating a new node at each step of the CRP as described above. The next subsection quantifies exactly the probability over the number of instantiated nodes as a function of α.

3.2.1 Induced distribution over the number of nodes

Strictly speaking, Eq. 12 defines a distribution over PDFCs with infinite nodes, rather than 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. It is useful to determine analytically the prior probability $p(K|\alpha)$ over the effective number of nodes K induced by our prior distribution. "Appendix 1" shows that it is given by:

$$p(K|\alpha) = \frac{\alpha^K (KY)!}{\alpha^{(KY+1)}} \sum_{l=K-1}^{(K-1)Y} \frac{\bar{\phi}(K,l)}{l!}, \tag{15}$$

where $\alpha^{(KY+1)}$ indicates the rising factorial, $Y = GH$, and $\bar{\phi}(K,l)$ is given by the recurrence relation

$$\bar{\phi}(K, l+1) = y\, \bar{\phi}(K,l) + \begin{cases} \bar{\phi}(K-1,l) & \text{if } l < (K-2)Y \\ 0 & \text{o.w.} \end{cases}, \tag{16}$$

that is initialized as $\bar{y}(K, K) = 1$. Figure 3 shows the distribution of K for various values of α, assuming as before $G = 3$ and $H = 2$. Again, we observe that the expected number of nodes increases with α, since a larger α corresponds to a higher probability of creating new nodes in the construction of Eq. 13.

3.3 Distribution over the emission function

For each node $k = 1, 2, \ldots, \infty$, the corresponding emission distribution θ_k over A_j is given a symmetric Dirichlet prior with total parameter λ, that is,

$$\theta_k \mid \lambda \sim \text{Dir}\left(\tfrac{\lambda}{G}, \ldots, \tfrac{\lambda}{G}\right) \quad k = 1..\infty \tag{17}$$

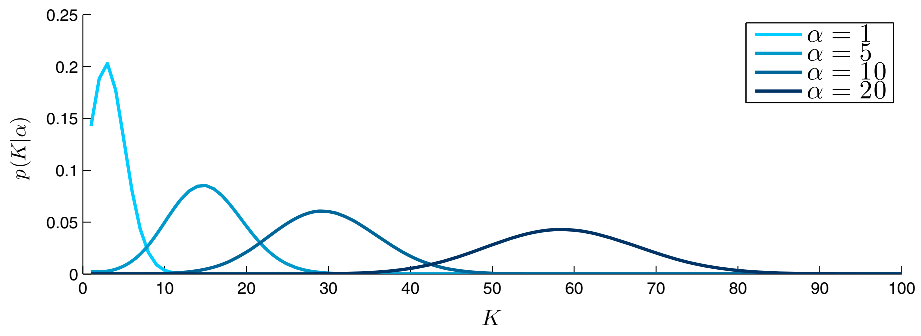


Fig. 3 Probability of PDFC size, for different values of α

The parameter λ encodes our prior belief on the entropy of the emission distributions of the PDFC's nodes: a large value of λ reflects a bias towards more stochastic actions, while lower values favor emission distributions skewed toward a single action. Setting $\lambda = G$ yields a non-informative (flat) prior distribution.

3.4 Alternative priors and related work

The hierarchical distribution described in this section is not the only prior one could design over the space of PDFCs. Other similar priors can be considered, such as ones based on the Pitman-Yor process, which is a slight generalization of the one presented here, and other forms of two-parameters stick-breaking priors [46]. Moreover, some classes of parametric prior distributions could be adapted to PDFCs, such as the hierarchical distribution described in [24]. Although these latter traditionally require more convoluted inference based on reversible jump Markov chain Monte Carlo methods, recent work [42] has shown promising results in deriving simpler inferential methods. We leave the exploration of such prior distributions for future work.

In a related work [48], 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 has similarities with our work. Moreover, stick-breaking priors have been used over the space of policies for decentralized POMDPs [37]. While the approach may be similar to ours in the way that priors are designed, the cited work presents a "planning as inference" methodology for decentralized POMDPs, targeting a very different problem than the one we consider in this paper.

4 Bayesian learning of PDFCs

This section provides the details of the algorithm used to implement the Bayesian learning process described in Sect. 3.1. First, the Bayesian learning problem is instantiated in our specific context of learning a PDFC in an I-POMDP environment, with the aid of a Bayesian network representation of the learning setup. Then, the MCMC learning algorithm is introduced, and each of the important steps is described in detail.

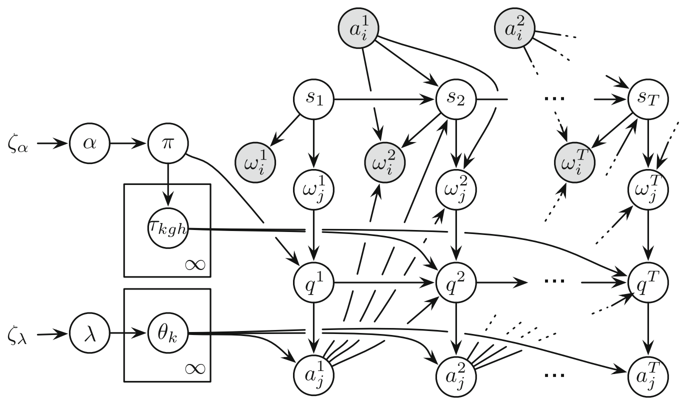


Fig. 4 Graphical model representation of PDFC learning. Observed variables are shaded in *grey*

4.1 Learning setup

As an I-POMDP, agent i is in general unable to perfectly observe the trajectory of observations and actions $(\omega_j^{1:T}, a_j^{1:T})$ of agent j, and instead just receives its own sequence of observations from the environment $\omega_i^{1:T}$. Additionally, agent i may not just be a passive observer, therefore its own actions $a_i^{1:T}$ need to be considered when learning j's model. We hence need to infer the posterior distribution over j's PDFC parameters:[7]

$$p(\tau, \theta | \omega_i^{1:T}, a_i^{1:T}) = \frac{p(\omega_i^{1:T} | \tau, \theta, a_i^{1:T})\ p(\tau, \theta)}{p(\omega_i^{1:T} | a_i^{1:T})}. \tag{18}$$

As we mentioned in the introduction, this distribution may not, in the infinite limit, converge to the actual true model of the other agent, since agent i perceives the world only through his own observations. Moreover, the true *type* of agent j may not belong to the class of PDFCs to begin with; even so, modeling j as a finite controller is a reasonable approach, as it allows to encode regularities in j's observed behavior. However, we cannot claim that PDFCs can in general model *every* behavior of agent j.[8]

The dynamic Bayesian network in Fig. 4 depicts the learning scenario graphically. The PDFC parameters (τ, θ) have prior distributions defined by Eqs. 12 and 17. The rest of the conditional distributions of the model are given by the dynamics (transition and emission function) of the PDFC being learned and the I-POMDP's transition and observation functions.

Moreover, we place an exponential hyperprior over the 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:

$$\begin{aligned} \alpha \mid \zeta_\alpha &\sim \exp(\zeta_\alpha) \\ \lambda \mid \zeta_\lambda &\sim \exp(\zeta_\lambda). \end{aligned} \tag{19}$$

[7] In order not to clutter notation, we consider the initial node q^1 as being part of τ.

[8] For instance, j's behavior may be time dependent, or be encoded as a pushdown transducer.

4.2 MCMC sampler for PDFC inference

Computing the posterior distribution defined in Eq. 18 is not analytically tractable. This is usually the case for complex models in Bayesian inference. In this work, we adopt a Markov-chain Monte Carlo (MCMC) algorithm to approximate the posterior distribution, inspired by previous research on inference for Dirichlet process mixture models (DPMM). Using a Monte Carlo method means that we will obtain an set of samples that approximates the true posterior distribution. 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, and the hidden variables $(s, \omega_j, a_j)^{1:T}$. However, the weight vector π can be integrated out analytically via the CRP construction, as described in Sect. 3.2 (Eq. 13). Moreover, since we placed a conjugate Dirichlet prior over the nodes' multinomial emission parameters, θ can also be treated 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 the sequence $q_{1:T}$ explicitly in the state: it can be derived at will when needed. The resulting *collapsed* state of the Markov chain is therefore a tuple $(\tau, \omega_j^{1:T}, a_j^{1:T}, \lambda, \alpha)$.

Given that the state comprises multiple variables, it seems natural to implement the MCMC algorithm as Gibbs sampling, which is a during which one variable at a time is sampled, while the others are kept fixed, until convergence. The most critical element to sample is the transition function τ. We can devise a Gibbs sampling jumping probability that, at each iteration, re-samples the destination of a single edge τ_{kgh}, given the value of all other state variables. Since only one edge can be modified at each iteration, we call this jumping distribution an *incremental move*, described in detail in Sect. 4.2.1. Despite being relatively easy to perform, allowing only one edge 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. This problem is not new in the context of Gibbs sampling for mixture models [7,30]. 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* [31], 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 the MCMC algorithm. A detailed description of this step is provided in Sect. 4.2.2.

Moreover, the algorithm needs to resample the hidden sequences $(s, \omega_j, a_j)^{1:T}$. As we describe in Sect. 4.2.3, this is done *in block* by using a backward filtering-forward sampling (BFFS) procedure. Additionally, the algorithm resamples the hyperparameters α and λ at each iteration. This is done via a Metropolis-Hastings (MH) step, as detailed in Sect. 4.2.4.

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*.

Algorithm 1 LearnPDFC

```
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
1: $q^{1\,(1)} \leftarrow 1$
2: $\tau_{1gh}^{(1)} \leftarrow 1 \quad \forall\, g = 1..G, h = 1..H$
3: $\alpha^{(1)} \sim exp(\zeta_\alpha)$
4: $\lambda^{(1)} \sim exp(\zeta_\lambda)$
                                                         ▷ Initialize hidden sequences
5: $(s^{1:T}, a_j^{1:T}, \omega_j^{1:T})^{(1)} \leftarrow$ sample-seq$(\tau, \lambda, \omega_i^{1:T}, a_i^{1:T})$
                                                         ▷ MCMC iterations
6: for $n = 2..N_{iter}$ do
7:     if mod $(n, R) \neq 0$ then
8:         $\tau^{(n)} \leftarrow$ incremental-move$(\tau^{(n-1)}, \alpha^{(n-1)}, \lambda^{(n-1)}, (a_j^{1:T}, \omega_j^{1:T})^{(n)}, M)$
9:     else
10:        $\tau^{(n)} \leftarrow$ split-merge$(\tau^{(n-1)}, \alpha^{(n-1)}, \lambda^{(n-1)}, (a_j^{1:T}, \omega_j^{1:T})^{(n)}, S)$
11:    end if
12:    $(s^{1:T}, a_j^{1:T}, \omega_j^{1:T})^{(n)} \leftarrow$ sample-seq$(\tau^{(n)}, \lambda^{(n)}, \omega_i^{1:T}, a_i^{1:T})$
13:    $(\alpha^{(n)}, \lambda^{(n)}) \leftarrow$ sample-hyperpars$(\tau^{(n)}, \alpha^{(n-1)}, \lambda^{(n-1)}, (a_j^{1:T}, \omega_j^{1:T})^{(n)})$
14: end for
```

4.2.1 Incremental moves (Algorithm 1, line 8)

In order to perform an incremental Gibbs move, we first choose a single edge source uniformly at random out of the $KGH + 1$ transitions of the PDFC in the current state, where K is the current number of PDFC nodes. Note that the initial node q^1 is here considered as a special edge, hence the '+1'. Let the chosen edge be indexed as kgh, indicating the source node, the action, and the observation it corresponds to, respectively; 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 existing nodes or a new one, that is added to the PDFC. By using Bayes' rule and applying the conditional independence encoded in the model's structure (see Fig. 4), we obtain:

$$p(\tau_{kgh}|\tau_{-kgh}, \omega_j^{1:T}, a_j^{1:T}, \alpha, \lambda) \;\propto\; p(\tau_{kgh}|\alpha, \tau_{-kgh})\; p(a_j^{1:T}|\tau, \omega_j^{1:T}, \lambda), \qquad (20)$$

where τ_{-kgh} denotes all current values of τ except the one being sampled, i.e. $\tau = \tau_{kgh} \cup \tau_{-kgh}$.

The first term of the RHS side of Eq. 20 is the conditional prior distribution. Since the edges of a PDFCs represent an exchangeable sequence given the stick-breaking prior, we can consider that each edge is sampled after all the others and apply the CRP conditional distribution in Eq. 13. 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 by computing the expectation of the quantity in Eq. 9 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, the likelihood can be easily computed in closed form by using the properties of the Dirichlet-multinomial model [21]. 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,

$d_{kg} = \sum_{t=1}^{T} \delta(q^t, k)\delta(a_j^t, g)$. The likelihood term in Eq. 20 is then given by:

$$p(a_j^{1:T} | \omega_j^{1:T}, \tau, \lambda) = \prod_{k=1}^{K} \left[\frac{\Gamma(\lambda)}{\Gamma(\sum_{g=1}^{G} d_{kg} + \lambda)} \prod_{g=1}^{G} \frac{\Gamma(d_{kg} + \lambda/G)}{\Gamma(\lambda/G)} \right]. \tag{21}$$

Computing the likelihood of assigning τ_{kgh} to a *new* node is more complicated 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 [48]. In our case, however, such method leads to slow mixing rates. A better solution is to adapt the *auxiliary variables* algorithm described in [43]. 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 Eq. 20, distributing α uniformly among the M candidates, so that the total prior probability of generating a new node is still proportional to α. We refer the reader to [43] for additional details on this procedure.

4.2.2 Splitting and merging nodes (Algorithm 1, line 9)

This step starts by sampling two edges uniformly at random. If they 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 Metropolis-Hastings 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 propose high-likelihood splits, the algorithm described in [31] 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 statistical details of this procedure are quite involved, and go beyond the scope of this paper; the interested reader can find a description of the basic method in [31].

4.2.3 Sampling hidden sequences (Algorithm 1, line 12)

An easy way to sample the sequences $(s, \omega_j, a_j)^{1:T}$ 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, considering that 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 [53]. It is actually possible to take this approach further, and sampling the three sequences *together* and in block, given the current values of τ and the evidence $(a_i, \omega_i)^{1:T}$. In order to do this, we use a backward filtering, forward sampling procedure (BFFS), described in the following.

We begin by noticing that, given τ, the value of the variables (s^t, q^t) at each timestep makes all future variables conditionally independent from past ones; this is due to the Markovian nature of both I-POMDP and PDFC and can be evinced from the structure of the Bayesian network in Fig. 4. Therefore, by applying concatenation and Bayes rule, the following holds for each $1 < t \leq T$:

$$\begin{aligned} &p(a_j^{t-1}, s^t, \omega_j^t | s^{t-1}, q^{t-1}, a_i^{t-1:T}, \omega_i^{t:T}, \tau) \\ &\quad\propto p(a_j^{t-1}|q^{t-1})\ p(s^t|s^{t-1}, a_i^{t-1}, a_j^{t-1})\ p(\omega_j^t|a_i^{t-1}, a_j^{t-1}, s^t) \\ &\quad\times p(\omega_i^t|a_i^{t-1}, a_j^{t-1}, s^t)\ p(\omega_i^{t+1:T}|s^t, q^t = \tau_{q^{t-1}a_j^{t-1}\omega_j^t}, a_i^{t:T}). \end{aligned} \tag{22}$$

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 [53]: 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:

$$\begin{aligned} \xi^t(s^t, q^t) =& p(\omega_i^{t+1:T}|s^t, q^t, a_i^{t:T}) \\ =& \sum_{a_j^t} p(a_j^t|q^t) \sum_{s^{t+1}} p(s^{t+1}|s^t, a_i^t, a_j^t)\ p(\omega_i^{t+1}|a_i^t, a_j^t, s^{t+1}) \\ &\times \sum_{\omega_j^{t+1}} p(\omega_j^{t+1}|a_i^t, a_j^t, s^{t+1})\ \xi^{t+1}(s^{t+1}, \tau_{q^t a_j^t \omega_j^{t+1}}). \end{aligned} \tag{23}$$

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 Eq. 22, moving forward from $t = 1$ to T.

4.2.4 *Resampling hyperparameters (Algorithm 1, line 13)*

The stick-breaking parameter α and the Dirichlet parameter λ are distributed exponentially with parameters ζ_α and ζ_λ, respectively. We adopt a Metropolis-Hastings (MH) procedure in order to resample their values at each iteration of Algorithm 1. The details are described in the following.

The parameter α is conditionally independent from all the other variables, given the current number of nodes K of the PDFC. Since sampling from $p(\alpha|K)$ directly is not feasible, we use the MH method. First, 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 $a(\alpha^*|\alpha)$ [27]. By substituting the appropriate densities into the formula and simplifying, we obtain:

$$a(\alpha^*, \alpha) = \min\left[1, \frac{\alpha^*}{\alpha}\frac{e^{-\zeta_\alpha \alpha^*}}{e^{-\zeta_\alpha \alpha}}\frac{p(K|\alpha^*)}{p(K|\alpha)}\right]. \tag{24}$$

where the likelihood terms $p(K|\alpha)$ and $p(K|\alpha^*)$ can be computed from Eq. 15.

The Dirichlet parameter λ is conditionally independent from all the other variables, given the counts d_{kg} of the actions executed in each node. To sample λ, we use the same method as above, obtaining the MH acceptance ratio:

$$a(\lambda^*, \lambda) = \min\left[1, \frac{\lambda^*}{\lambda}\frac{e^{-\zeta_\lambda \lambda^*}}{e^{-\zeta_\lambda \lambda}}\frac{p(d|\lambda^*)}{p(d|\lambda)}\right], \tag{25}$$

where the likelihoods $p(d|\lambda)$ and $p(d|\lambda^*)$ are given by Eq. 21.

5 Planning against the learned models

While the previous sections focus on learning probabilistic deterministic finite-state controllers of another agent, this section is dedicated to the use of the learned models in interactive POMDPs. We first formally introduce the framework of subintentional I-POMDPs with PDFC models, and subsequently describe two methods of use of such framework.

5.1 Subintentional I-POMDP with PDFC models

Recall that in the general case, the interactive state space IS_i of an I-POMDP is defined as the cross product of the set of world states S and j's models M_j, i.e. $IS_i = S \times M_j$, and that a model of the other agent is a tuple $m_j = (h_j, f_j, O_j)$, where h_j is a history for agent j, f_j is an agent function, and O_j is an observation function.

A *subintentional I-POMDP with PDFC models* is defined by specializing the set of j's models M_j from the general I-POMDP case to only contain models that can be expressed as PDFCs. Note that this approach takes a subintentional stance to opponent modeling, as opposed to the intentional (or *type-based*) approach that is traditional in I-POMDP literature. Specifically, there is no explicit modeling of the other agent's beliefs and preferences; j is simply modeled as a statistical process that takes observations as inputs and generates actions as output.

Formally, an element of M_j is a tuple $m_j = (q_j, c_j, O_j)$, where c_j is a PDFC that implements the agent function, and q_j is a node of the PDFC, that summarizes j's past history. Since we assume that j's observation model is known, an interactive state is a tuple $is_i = (s, c_j, q_j)$. 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 1. 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 observations, and 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 Eq. 3 can be specialized to our case and rewritten as:

$$p(is_i'|a_i, \omega_i, b_i) = \beta \sum_{is_i \,:\, c_j = c_j'} b_i(is_i) \sum_{a_j} \theta^{c_j}(q_j, a_j)\ O_i(a_i, a_j, s', \omega_i)\ p(is_i'|is_i, a_i, a_j), \tag{26}$$

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

$$p(is'_i|is_i, a_i, a_j) = T(s, a_i, a_j, s') \sum_{\omega_j} O_j(a_i, a_j, s', \omega_j)\ \delta(q'_j, \tau^{c_j}(q_j, a_j, \omega_j)). \quad (27)$$

5.2 Solving subintentional I-POMDPs with Monte Carlo methods

From Eq. 26, we observe that there is no recursion into j's intentional models. In fact, we can formalize a subintentional I-POMDP as a single-agent POMDP where the transition function is defined as in Eq. 27. In other words, a subintentional I-POMDP can be flattened into a POMDP, by extending the state space to include the other agent's models. It follows that standard POMDP algorithms can be adapted in order to compute a solution.

Note however that 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. Therefore, exact POMDP solving techniques are surely doomed to fail in this case due to the problem's dimensionality. A faster, approximate solution method is necessary.

In this paper, we propose an adaptation of the POMCP algorithm, introduced in [58] and summarized in Sect. 2.4, to solve subintentional I-POMDPs. An attractive feature of this algorithm is that its running time of does not directly depend on the size of the problem, since it makes use of a generator implementation of the I-POMDP rather than a flat or factorized specification. In other words, it avoids enumerating each single interactive state, or even a factorized, propositional description of $S \times M_j$. We note, however, that larger problems will likely need more simulations to explore the forward search tree, and therefore may still require a higher computation time to achieve best performance.

We consider two modalities for applying I-POMDPs to interactive scenarios, described in the following sections.

5.3 Two-phase approach

The first method we consider is a two-phase approach, in which agent i first collects data about the behavior of agent j, learns a set of candidate PDFCs C_j, and then exploits the learned models in its own decision making process by embedding C_j into the I-POMDP's interactive state space.

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 to i's presence, or that j plans against a model of i that is given a priori.

While an approach that separates learning from planning may sometimes be unrealistic, it is applicable to some scenarios in which interaction data about agent j is available. For example, in [11] the authors describe a two-phase approach to learning and planning for interactive dynamic influence diagrams [16], with application to learning opponent models in videogames from replay data. Moreover, a two-phase approach allows us to examine the properties of learning and planning in isolation.

5.4 Interleaved learning and planning

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 PDFC inference method described in Sect. 4 with the POMCP algorithm for POMDPs.

As described in Sect. 2.4, the POMCP algorithm maintains a particle representation of the agent's belief about the state of the environment. For a subintentional I-POMDP, this means that each particle is a triple (s, c_j, q_j). As we saw above, the element c_j 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. This is done by periodically re-train 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, which are performed by invoking Algorithm 1. However, instead of initializing the learning 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 the PDFC that results from the majority vote among the current particles in agent i's belief state, that is, the current maximum a-posteriori. The choice of hot-starting the sampler means that the models learned in the past are not completely discarded.

The data that is used for re-training the models at each update is is i's most recent history, going back W steps, i.e., at time t, $z_i^{t-W+1:t} = (a_i^{t-W:t-1}, \omega_i^{t-W+1:t})$. Note that 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 interaction. 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.

5.4.1 Resampling j's PDFCs in the POMCP algorithm

As described above, a new set of PDFCs is obtained every ΔT_{update} timesteps, by executing the MCMC algorithm with the observations gathered by agent 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 q^1 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 (Sect. 4.2.3) stores the sampled value of q_j^t, so that it is readily available when interaction resumes. Moreover, we store the last value s^t of the world state sequence that gets sampled by the algorithm (see Eq. 22). This is necessary because s^t and c_j^t are not independent variables: different models of j induce different probabilities over S, and vice versa.

Accordingly, we represent –slightly abusing notation– the learned PDFCs at time t as a set C_j^t of triples (s^t, c_j, q_j^t), where c_j is a PDFC, q_j^t is the node occupied at time t, and s^t is the sampled state of the world at time t.

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

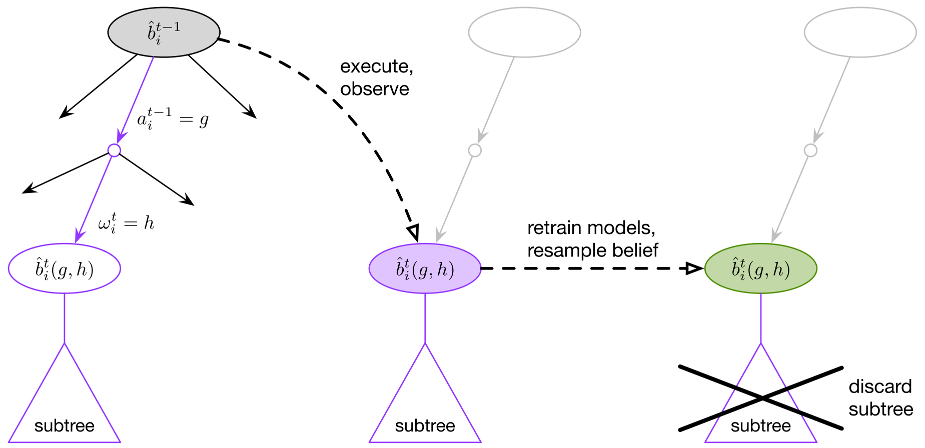


Fig. 5 One step of the execution of the POMCP algorithm for subintentional I-POMDPs, including retraining and resampling of j's PDFCs

$$\hat{b}_i^t = \frac{1}{|B_i^t|} \sum_{(s^t, c_j, q_j^t) \in B_i^t} \delta_D(s^t, c_j, q_j^t), \tag{28}$$

where δ_D is the Dirac delta function.

The set of particles is refreshed by re-sampling each particle uniformly at random from C_j^t. Note that C_j^t is the result of the MCMC learning algorithm, and therefore represents an empirical approximation of the posterior belief over agent j's PDFCs and the world state, i.e.:

$$p(s^t, c_j, q_j^t | z_i^{1:t}) \approx \frac{1}{|C_j^t|} \sum_{(s^t, c_j, q_j^t) \in B^t} \delta_D(s^t, c_j, q_j^t), \tag{29}$$

where again δ_D is the Dirac delta. Note that Equations 28 and 29 have the same form. Therefore, sampling uniformly at random from C_j^t corresponds to sampling from the posterior distribution over agent j's PDFCs and the world states, and produces a belief consistent with agent i's history of observations.

After resampling the belief state, 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 subtree (the search tree starting from the current belief) that was previously expanded and stored during the execution of the algorithm's forward search.

Normally, this 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.

However, after the set of j's PDFC is periodically retrained, the subtree that was previously expanded contains nodes that are no longer compatible with the newly sampled belief state. This is because the domain $|C_j^t|$ of possible models of j has changed. In other words, the previously explored belief subtree extends into a state space that is different from the current, updated one. When this happen, therefore, the belief subtree must be discarded to avoid incompatibility, and the forward search for the optimal action must restart from scratch.

Figure 5 depicts one step of execution of the POMCP algorithm adapted to our case, including retraining and resampling of j's PDFCs. At time $t-1$, agent i performs a randomized forward search according to the POMCP procedure, eventually choosing action

$a_i^{t-1} = g$ and receiving observation $\omega_i^t = h$ from the new world state. Agent i's updated belief state at time t corresponds to the node here denoted as $\hat{b}_i^t(g, h)$. At this point, i uses its recent observation history (of length W) to re-train the set of j's models, and refreshes its belief state by sampling from the output of the MCMC algorithm according as described above. At this point, the existing subtree of node $\hat{b}_i^t(g, h)$ is no longer compatible with the resampled belief, and must therefore be discarded.

5.5 Note: generalization to more than two agents

It is possible to generalize learning and planning to more than one opponent. Consider that each of the other agents has its own actions A_j and observations Ω_j. We can define sets $A_{-i} = \times_{j \neq i} A_j$ and $\Omega_{-i} = \times_{j \neq i} \Omega_j$. The combined observation function of the other agents would then be $O_{-i}(a_i, a_{-i}, \omega_{-i}, s) = p(\omega_{-i}|a_i, a_{-i}, s) = \prod_{i \neq j} O_j(a_i, a_{-i}, \omega{-i}, s)$, which works under the reasonable assumption that the individual agents' observations are conditionally independent given state and actions. Given this, and the other elements of the I-POMDP, the modeling agent i could learn a single *combined* PDFC of the other agents, by applying the same method described in this paper.

This approach is simple, but it can be impractical if there are a lot of agents with large actions and observation sets. Under the same assumption above, which implies that the PDFC nodes Q_j of each of the other agents are also conditionally independent, we can have a learning setup that involves multiple PDFCs. Obviously, the learning problem is in this case more complex, and the issue of non-discernible configurations of variables with respect to the modeling agents' own stream of observation is potentially exacerbated.

While both of the above approaches are theoretically sound and are relatively simple generalizations of the methodology presented in this paper, we have not yet explored their practical limitations experimentally.

6 Experimental results

Our experimental evaluation has the following main objectives:

1. Demonstrate that, under ideal observability conditions, Algorithm 1 learns PDFCs that approximate the true controllers generating the modeled agent's behavior.
2. Quantify the performance increase provided by the integration of the learned PDFCs in the modeling agent's I-POMDP, both in the two phase and interleaved learning and planning approach.
3. Analyze the dynamics emerging in case both agents are interacting and simultaneously learning each others' model via interleaved learning and planning.

The following section introduces the three experimental domains that are used in our experiments, while the subsequent sections cover each of the three objectives above separately.

6.1 Experimental domains

6.1.1 Multiagent tiger problem

This domain is the two-agent extension of the well-known Tiger Problem [33] introduced in [23]. There are two rooms, one hiding a ferocious tiger and the other hiding some gold.

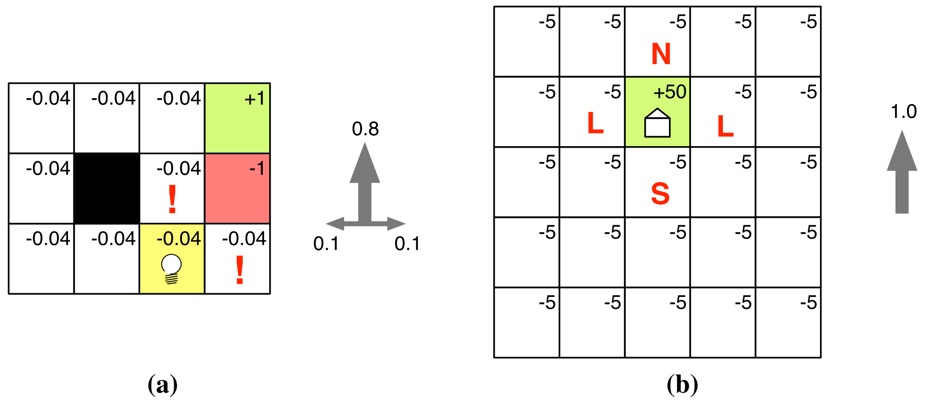


Fig. 6 **a** Maze and **b** AUAV domains from agent j's perspective

Each agent can either listen (L), open the left door (OL), or open the right door (OR). Upon listening, an agent hears a growl coming from the direction where the tiger is with 0.85 probability, and from the other direction with 0.15 probability (GR, GR). Moreover, each agent hears a creak coming from the direction of the door opened by the other agent, or no creak (silence) if the other agent does not open a door (CL, CR, S). The observation set of both agents is hence composed of 6 possible observations. Upon opening a door, each observation is perceived with 1/6 probability (uninformative). When both agents listen, the state of the tiger persists onto the next timestep, while when either agent opens a door the position of the tiger is reset with uniform probability. For both agents, listening costs 1, opening the door hiding the tiger costs 100, and opening the door with the gold has a reward of 10. The formal specification of this problem is provided in Table 1, that can be found in "Appendix 2".

6.1.2 3 × 4 *maze*

In the single-agent version of this problem adapted from [56], an agent, j, tries to reach the top-right (+1) corner of the 3x4 grid in Fig. 6a, starting from a random, unknown location, trying to avoid getting caught (-1) by the monster lurking in the center-right. The agent can move up, down, left, or right, with 0.8 success rate, and 0.1 probability of slipping to either side, and each move costs 0.04. In general, the agent does not know its own position with certainty, but receives some signals: its location is revealed in the cell marked with a light bulb, and the monster's smell is perceived in the locations with the exclamation mark with accuracy 0.8. When reaching the goal or meeting the monster, agent j is re-spawn randomly for a new round. In the two-agent version of the problem, agent i is the monster that j believes stationary, that is tasked with catching j. It receives a reward of 1 when co-located with j and has the same transition model. It perceives i's position with 0.9 accuracy at each timestep.

6.1.3 AUAV reconnaissance

This problem, adapted from [63], models a fugitive agent, j, trying to reach a safe house, with reward 1, located on a 5 × 5 grid as in Fig. 6b. The agent can move deterministically in four directions, with cost 0.04. The agent can sense its position with respect to the safe house, namely whether it is north of it (N), south of it (S), or at the same level (L). These observations are received with accuracy 0.8, and only when the agent is adjacent to it. When the agent reaches the safe house, it receives and "end" signal and its position is reset to a uniformly random location for a new round. In the two-agent version, agent i is an autonomous unmanned aerial vehicle (AUAV), to whose presence agent j is oblivious. The AUAV moves deterministically and perceives the fugitive's position with 0.9 accuracy. It receives a reward of 1 when co-located with j, causing j's position to be reset at random, and a penalty of 0.04 otherwise.

6.2 Learning PDFCs from observable trajectories

We evaluate how similar the learned PDFCs are to j's true controller, when i is able to observe perfectly j's action and observation sequences (making line 12 in Algorithm 1 superfluous.) Although this may be in some cases an acceptable assumption (see for example the work in [11],) it is often unrealistic; however, it allows to evaluate the performance of the PDFC learning algorithm under ideal conditions. We assume that j's behavior is prescribed by the optimal policies computed assuming that j is oblivious to the presence of agent i. For the Tiger Problem, this produces an optimal finite controller with 5 nodes (depicted in Fig. 11a); for the Maze problem, j's controller has 42 nodes; for the UAUV problem, 36 nodes.

To derive a similarity measure between PDFCs, suppose that a hypothetical agent operates according to the true controller c_T, and another does so according to the learned controller c_L. 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. We then define the *weighted Kullback-Leibler distance* between the two controllers as:

$$wKL(c_L, c_T) = \sum_{(q_L, q_T) \in Q_L \times Q_T} \eta_{q_L q_T} \; KL(\theta_{q_L}, \theta_{q_T}), \tag{30}$$

which is a measure of similarity between the distributions over actions sequences induced by the two controllers [28], hence reflecting a *behavioral* similarity between PDFCs. Note that this measure does not compare the actual PDFCs: in fact, there may exist different models that induce the same distribution over the action sequence.

For each of the considered problems and for different lengths of observed history (T_{learn}), we performed 10 learning trials. The following parameters were used for the MCMC sampler: $M = 50$, $R = 50$, $S = 2$. In each trial, the MCMC sampler was run for 5000 iterations, and the second halves of the generated sample chains were subsampled every 100 iterations, resulting in ensembles of 25 PDFCs per trial. We computed the overall mean wKL across trials, which is reported in Fig. 7 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. 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, Fig. 8 reports the fraction of time spent in each node of the true controller, sorted in decreasing order. 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

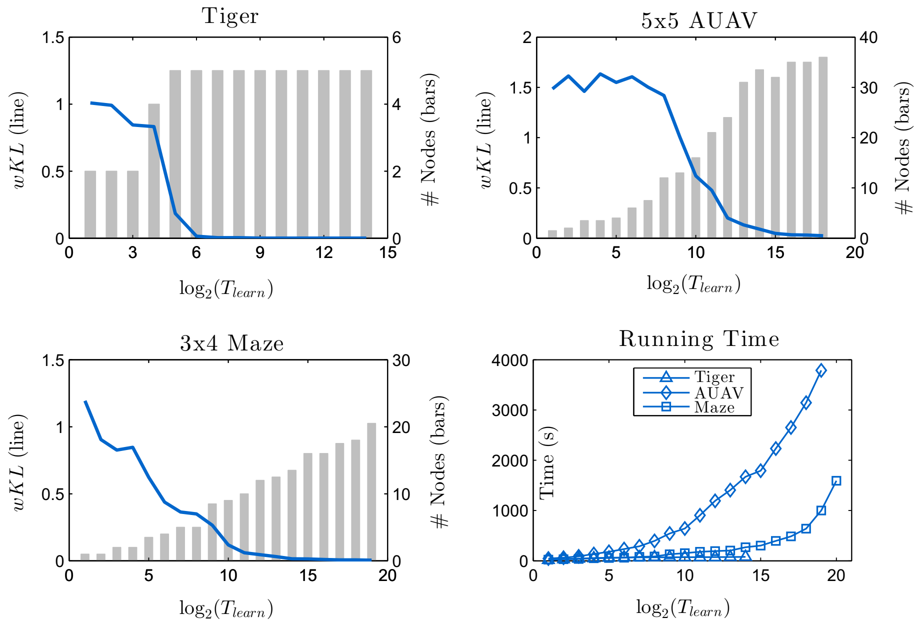


Fig. 7 Weighted KL distance between the learned and the true controllers (*line*) and number of nodes of learned PDFC (*bars*). The fourth panel reports the timing results

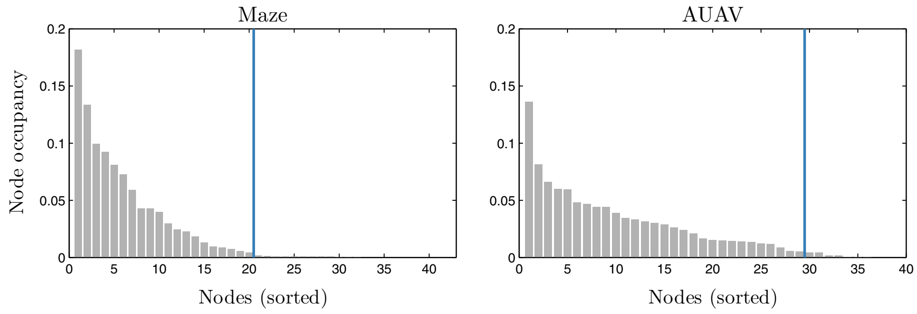


Fig. 8 Node occupancy for Maze and AUAV problems. The *vertical line* indicates the 99% percentile

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 or learning, and will be explored more in depth in future work.

The bottom-right panel of Fig. 7 reports the running time of the MCMC algorithm,[9] 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 optimizations in the computation of the quantities d used in Eq. 21, which contains the only dependency of running time on T_{learn} (since line 12 is not executed here.)

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

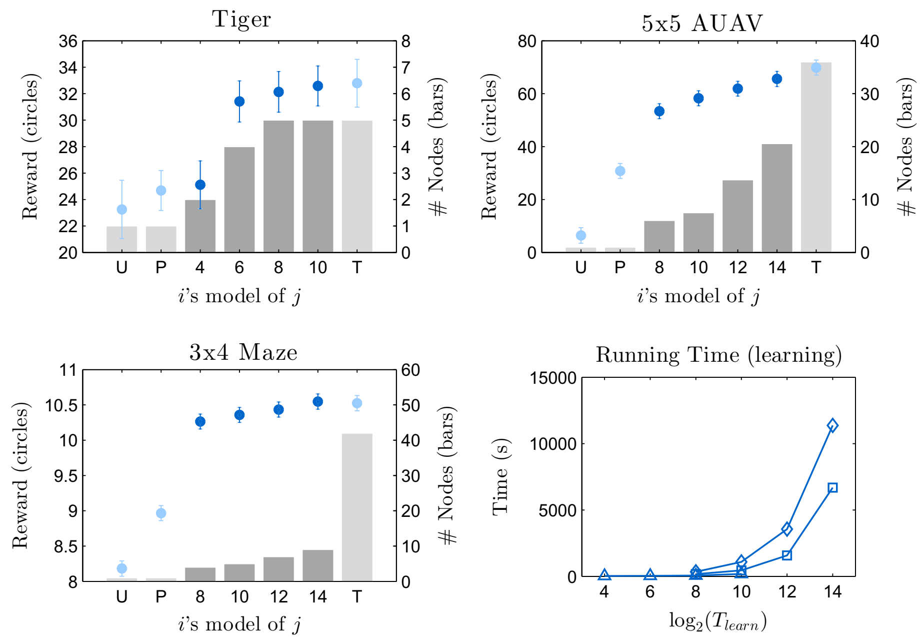


Fig. 9 Reward by agent i with different models of agent j (*lines*) and number of nodes of learned PDFCs (*bars*). The fourth panel reports learning time results

6.3 Learning offline and planning

In this section, we drop the assumption of perfect observability of agent j's behavior. Even so, we show how PDFC learning allows agent i to improve its performance. In our 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, and compare it against the following baselines: (**U**) i models j's actions uniformly at random, $p(a_j) = |A_j|^{-1}$; (**P**) i predicts j's action proportionally to their long-term frequency; and (**T**) i knows j's true PDFC. For each problem and observation size, we performed 20 learning and planning trials; for each trial, the MCMC algorithm was run for 5000 iterations and 25 sample PDFCs were retained as before, which constitute C_j in our I-POMDP model. The performance of the resulting I-POMDPs was then 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 [58].

Figure 9 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. Notice that, 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 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. This is also reflected in the size of the inferred controllers, which as expected grows with the amount of data. However, the PDFCs learned in this settings are usually smaller than the ones learned with perfect observability of j's behavior (previous section.) This is because now i's perception of j's behavior is filtered by noisy world's dynamics, and therefore longer observations are needed to allow identification of the same behavioral patterns.

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 in general unattainable, especially with realistic observability assumptions, i can still largely improve its performance by recognizing behavioral patterns that are statistically significant and encode them in a compact model.

The fourth panel of Fig. 9 reports the learning times. With respect to the previous section, we observe that sampling sequences 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.

6.4 Learning and planning online

6.4.1 Interacting with a stationary opponent

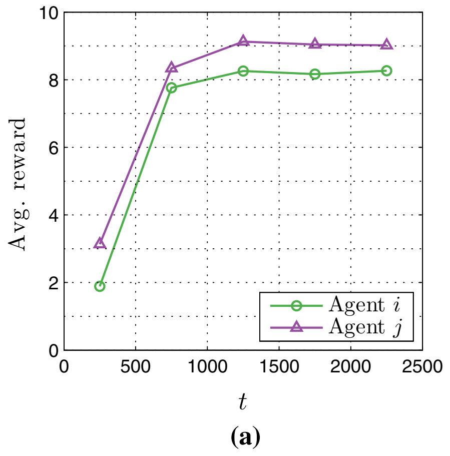
The experiments described in this section consider a learning agent, i, facing a stationary opponent j, with the goal of determining how the agents' rewards change during interaction as a result of i learning. With respect to the results in the previous section, it is important to note that here i collects observations while itself interacting, and therefore has to consider the effect of its own actions during learning.

We slightly modify the two-agent Tiger Problem from the previous section by making the rewards directly dependent on both agent actions: in this version, the agents get a reward of 50, instead of 10, when they find the gold *and* the other agent also open the same door. Intuitively, the agents are happier sharing the gold than finding it by themselves. This modification increases the importance of accurately predicting the other agent's action. 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 domain is provided in Table 2 in "Appendix 2".

For this domain, two types of agent j are considered (Fig. 10):

1. Agent j is implemented by the finite controller in Fig. 11a, which is the optimal policy for the single-agent Tiger Problem, corresponding to j assuming that agent i always listens, or being oblivious to i's presence. We refer to this policy as c_{tiger}.
2. Agent j plans online with POMCP, assuming that i's model is c_{tiger}. This is denoted as POMCP(c_{tiger}).

Agent i initially considers a uniform random PDFC of j, and subsequently updates j's models every $\Delta T_{update} = 500$ timesteps, using the last $W = 500$ observations, and the whole interaction has a duration of 2500 timesteps. All the parameters of the POMCP and MCMC algorithms were set as in the previous section, except the number of iterations of the MCMC



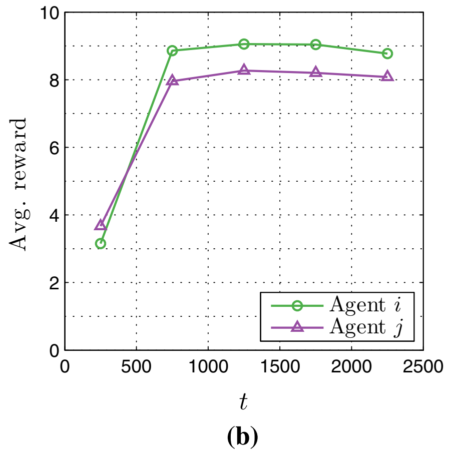
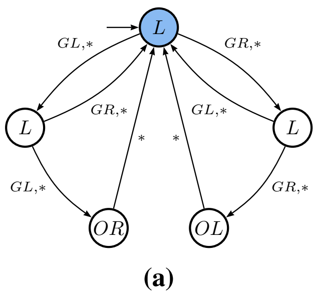


Fig. 10 Rewards for the Tiger domain: i learns j's model during interaction, with j being either a PDFC or an online planning agent that assumes a fixed model of i. **a** $f_j = c_{tiger}$. **b** $f_j = POMCP(c_{tiger})$



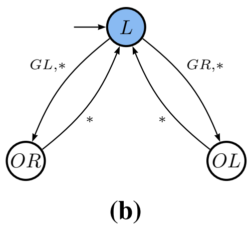


Fig. 11 Controllers for the tiger problem: **a** solution to the single-agent domain (c_{tiger}), and **b** aggressive synchronization strategy

sampler, which is here set to 2000. After each update, the set of PDFCs C_t^j is obtained by subsampling the second half of the MCMC realization every 100 iterations (20 PDFCs.)

Figure 10 reports the mean average rewards over 50 runs for each of agent j's types, aggregated in segments of 500 timesteps, so to align with ΔT_{update}. We can see that, in both cases, the rewards of the two agents rapidly grow after i learns about j's policy, due to i's increasing ability to predict j's actions. This is a noteworthy result: PDFCs appear to be suitable to describe j's behavior even when the true policy is not implemented by a finite controller.

When j is of type 1, we observe that its utility is actually higher that i's, despite i being the one that enables coordination. If we were to assume that i knew with certainty that j's policy was c_{tiger} beforehand, the resulting flattened I-POMDP would be simple enough to be solved exactly; i's best response would be in this case a 10-node FSC (not depicted for brevity.) The agents' expected average rewards, computed analytically, would be 8.33 and 9.26 respectively for i and j, which are the values that we observe in the plot, although in our case i has to *learn* j's policy. When j is of type 2, the agents' rewards observed in Fig. 10b mirror the ones in Fig. 10a. This is quickly explained by the fact that j is here providing a best response to c_{tiger}, and therefore its behavior should be similar to the best response

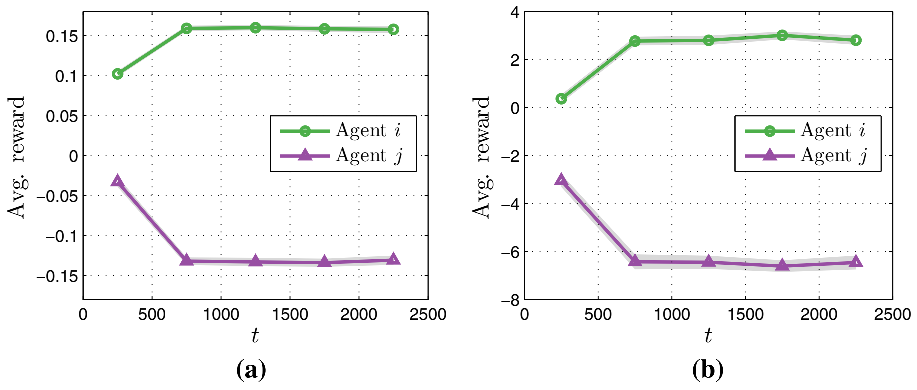


Fig. 12 Average rewards for the two agents as i learns, in the **a** Maze and **b** AUAV domains

that i provides in the previous case. Now, if i is able to learn a PDFC of j that accurately encodes this behavior, we already know that it should obtain an average reward that is at least as high as the one obtained by j in the previous case. The fact that this is indeed the case demonstrates the quality of i's learning and planning. In both situations, we see how i, despite being initially completely uninformed about j's policy, quickly learns an accurate predictive model, and then behaves approximately optimally with respect to this subjective belief.

In both the Maze and AUAV domain, we consider the case in which j uses the POMCP algorithm to make decisions, assuming a completely random model of agent i. The experimental setup is the same as for the Tiger domain. The rewards depicted in Fig. 12 show that in both domains agent i drastically increases its rewards as a result of learning, thus causing j's rewards to decrease. Note that i's observation set has cardinality 144 and 625, respectively for the Maze and AUAV domain, making the backward filtering operation in Eq. 23 very time consuming. For this reason, an ad-hoc particle filter implementation [18] of this operation is used in Algorithm 1 for these two larger domains.

Learning about j's policy allows i to better predict j's future trajectory. By planning its own actions according to such predictions, agent i is able to move toward locations where j will be in the future, instead of just chasing erratically after j. Two representative examples of i's behavior in the AUAV domain before and after having learned j's PDFC are depicted in Fig. 13a, b, respectively. In the left figure, we see that i moves in the general direction of j, but does not succeed in being co-located. In the right figure, instead of chasing blindly after j, i smartly moves towards the left of the grid, where it intercepts j and collects its reward.

6.4.2 *Self-play in the multiagent tiger domain*

In this section, we focus on the multiagent Tiger domain and analyze the dynamics that emerge when the two agents are simultaneously acting and learning about each other, that is, when our approach is used in *self-play*.

We consider two types of agents, whose only difference is in the reward function. The first type, that we call "altruistic", receives the same payoff as in the previous section. The second type, denoted as "selfish", receives a reward of 50 when finding the gold *if the other agent does not open the same door* (i.e. it opens the other door or listens), and 10 otherwise. All

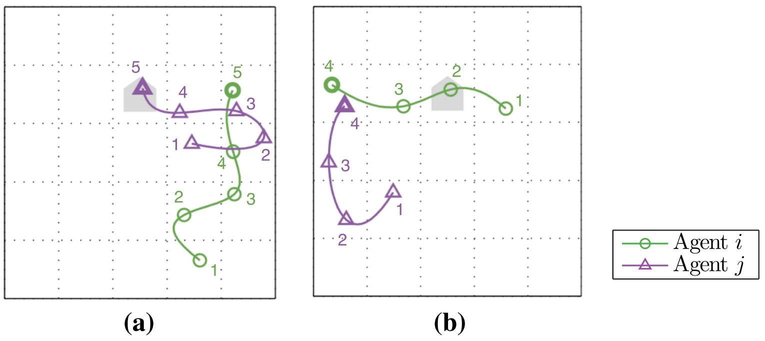


Fig. 13 Sample trajectories in the AUAV domain, before and after i learns j's model. **a** Before learning. **b** After learning

other I-POMDP parameters are the same as before. The reward functions described above are formalized in Table 3 in "Appendix 2".

We consider the three combinations of agents being of the two types above, namely, both agents being altruistic or selfish, and one being selfish and the other altruistic. The two agents' payoffs are clearly compatible when both are altruistic; we expect that they will want to synchronize their actions in order to open the same door at the same time. When both are selfish, we expect that the agents will try to "avoid each other", whilst still trying to accumulate optimal rewards. In the third case, the agents have conflicting goals. One will try to synchronize, while the other will try do the opposite.

The agents do not know each others' type. More strongly, they do not even know that the other might be of one of the two types above. Indeed, the other agent's preferences are never modeled explicitly. Coordination, or lack thereof, will exclusively be the effect of the two agents' learning, and not of prior knowledge.

Figure 14 shows the results of our experiments. The plots in the top row report the agents' average rewards, while the bottom charts depict the mean size of the learned PDFCs. The simulations were run with the same parameters as before, only this time with a duration of 5000 timesteps.

When both agents are altruistic, their average rewards increase quickly and then stabilize slightly above 10. The behavior of the two agents at this regime consists of iteratively listening for one timestep, then immediately opening the door opposite the direction of the growl. This behavior is *synchronized*: both agents listen, then open, and so on. This policy, here obtained as a result of online planning, can be encoded as a finite controller with 3 nodes, shown in Fig. 11b. The I-POMDP in which the other agent's model is represented by such policy can be solved analytically, and it can be shown that the best response is in fact *the same 3-node controller*. This means that this policy is a best response to itself, hence an equilibrium. Moreover, the expected utility of such strategy pair is 10.7, which is the value the rewards converge to experimentally. The fact that the agents achieve an equilibrium without any prior knowledge of each others' payoffs or policy is remarkable and shows that, in this setting, cooperation emerges naturally from our methodology. In the bottom of Fig. 14a we can see that the average size of the inferred PDFCs initially grows and then decreases to about 4 nodes. While there might be some redundant nodes, it is evident that the size of the learned models does not grow indefinitely, instead adjusting to the complexity of observed behavior, due to the use of a Bayesian nonparametric prior.

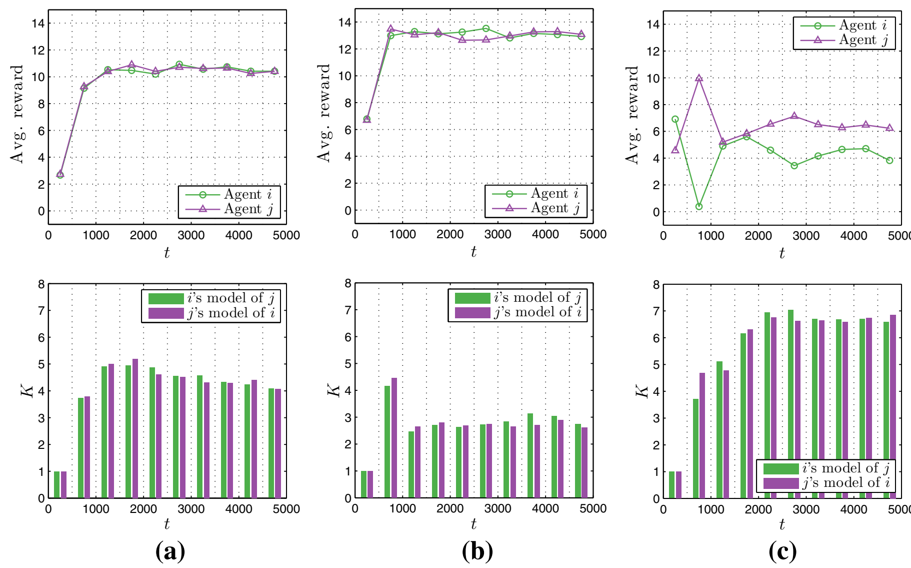


Fig. 14 Average rewards and number of nodes in the learned PDFCs in the Tiger domain, for different combinations of agent types. **a** i altruistic, j altruistic. **b** i selfish, j selfish. **c** i selfish, j altruistic

Since two selfish agents want to avoid each other, their best combined strategy is arguably to open doors every other timestep, as before, only now in an *anti-synchronization* pattern. This is exactly the behavior generating the rewards reported in Fig. 14b, where the agents quickly learn about each others' policy. This pair of strategies can be encoded with two 3-node controllers of the type showed in Fig. 11b, with different starting nodes, that is, one agent starts listening, while the other starts by opening either door, yielding an expected average utility of 13.15. Consider that with this strategy pair, an agent that opens a door cannot speculate about what growl the other agent receives. Therefore, the only significant pattern in the *observed* behavior of the opponent is the alternation of listening and (any) opening, which can be encoded in a PDFC with two nodes. This is why the size of the inferred controller drops to slightly above two in this case.

When the two agents have conflicting goals, their behavior is not easily predicted, and their policies do not seem to converge to an equilibrium. The spikes in the initial part of the reward plot in Fig. 14c are caused by the agents taking advantage of each others' model in an alternate pattern: first, the altruistic agent j takes advantage of i's initial aggressive policy by synchronizing door openings; after the next learning cycle, j recognizes this behavior and is able to respond by delaying opening doors. 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, and of the planning and learning algorithms. However, looking at individual runs, one could observe that the reciprocal exploitation dynamics described above re-emerge at irregular intervals. The agents exhibit a more complex behavior than in the previous two cases, which is reflected in the size of the learned PDFCs.

Figure 15 reports the rewards for all three cases when reducing the size of the re-train sequence and intervals from 500 to $W = \Delta T_{update} = 200$. We can see that for this problem the reward dynamics remain the same even when using shorter and more frequent updates. While this is true in this specific domain, we cannot claim that in general the results are

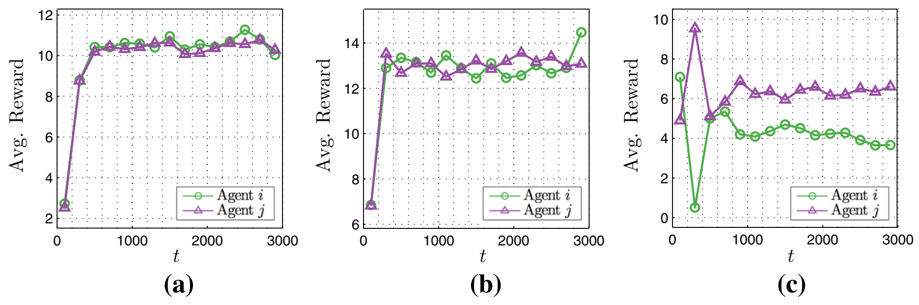


Fig. 15 Average rewards in the Tiger domain for different combinations of agent types, with $\Delta T_{update} = W = 200$. **a** i altruistic, j altruistic. **b** i selfish, j selfish. **c** i selfish, j altruistic

qualitatively unaffected by varying the size of observation window and update frequency. We defer the investigation of this important aspect to future research.

6.4.3 Social dynamics: "Follow the Leader"

In this section, we consider a further variation of the multiagent Tiger Problem, in which both agents are "indifferent" with respect to sharing or not sharing the gold with the other agents. In this case, however, the agents are endowed with asymmetric growl observation functions: agent i perceives growls with accuracy 0.96, while j has a hearing 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 complete specification for this domain is provided in Table 4 in "Appendix 2".

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 and open the opposite door. 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. As a result, the interaction between these two agents is initially very asymmetric, with i opening doors frequently and collecting high rewards, and j acting more timidly and gathering very little.

However, we see in Fig. 16 that 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 high probability in its location. This dynamics has previously been analyzed from an I-POMDP perspective using intentional models [50], and given the name "Follow the Leader." In the intentional case, however, it is assumed that j knows i's reward function, and therefore knows that i *wants to avoid the tiger*, making it straightforward to deduce that the door it opens likely leads to the gold.

In our case, instead, j does not know agent i's preferences, and in fact never attempts at modeling them. Yet, 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

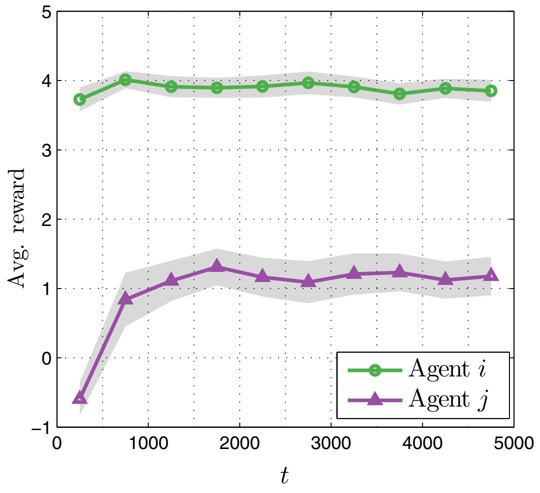


Fig. 16 Average rewards for the "follow the leader" scenario. The *gray* area represents the 95% confidence interval

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 Conclusion and future work

In this paper, we presented a novel approach to 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 PDFCs in the modeling agent's own decision making process, specializing the general I-POMDP framework to 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 experimental results validate the proposed methodology. For three domains of varying complexity, we show that the MCMC algorithm learns PDFCs that converge behaviorally to the true controllers generating the data, under perfect observability of the modeled agents' behavior trajectory. Even removing this assumption, the agent is able to learn models that when exploited in an I-POMDP framework allow its rewards to increase, by capturing relevant patterns in observed agent's behavior. We demonstrate how this continues to be true even when learning and planning are interleaved at execution time. Moreover, we analyze the dynamics emerging when two agents simultaneously learn about each other in one interaction domain, showing that cooperation, if beneficial, emerges naturally from our approach. Lastly, we provide an example of how learning another agent's PDFC allows to augment one's observation capabilities, by performing indirect inference via the modeled agents' behavior.

There are several directions for future work. While we provide a method for interleaving learning and planning periodically, a fully online learning method would update the other agent's models at every timestep, instead of just updating the belief over an ensemble of PDFCs as in the current approach. Even though online nonparametric Bayesian inference is currently still a niche in its early development, we believe that particle-based learners [39] are a promising avenue to achieve this goal and would integrate fully with particle-based planning algorithms such as POMCP. We believe that particle-based methods can provide as much speedup in the learning phase as they do in online POMDP planning, thus allowing our methodology to be applied to more complex problems.

Throughout this work, we assume that the observation function of other agents is known. However, if this is not true, and especially if their observation function is completely unknown, it should also be the subject of learning. At the current time, it is unclear whether an agent would be able to learn separately another agents' observation function *and* its policy, since the impact of these two elements may not be discernible from the point of view of the observer. Learning a combined model that directly maps the observer's actions and the state of the world to the other agents' actions might be a better approach in this situation.

As mentioned before, our approach makes no assumption of knowledge about the other agents' preferences. This makes the approach general, but 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 generate *transferable* knowledge that the agent can exploit in different situations. This problem is similar to *inverse reinforcement learning*, that has been extensively studied in the context of (fully observable) MDPs [44,54,64]. Although there are less results for inverse reinforcement learning in partially observable domains, we believe that recent approaches, such as [9], can be applied to our case, making it possible to back-engineer information about an agent's utilities from the learned PDFC policies.

Appendix 1: Derivation of the induced prior probability on 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 Eq. 12 as a function of the concentration parameter α, the number of actions G and observations H. 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 Eq. 13, 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 $Y = GH$ 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 $1 \leq y \leq Y$ are from the first node, transitions $(Y+1) \leq y \leq 2Y$ 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:

$$p(K=1|\alpha) = \frac{\alpha}{\alpha}\frac{1}{(1+\alpha)}\frac{2}{(2+\alpha)}\cdots\frac{Y}{(Y+\alpha)} = \frac{\alpha Y!}{\alpha^{(Y+1)}}, \tag{31}$$

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

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 Y outgoing transitions, therefore:

$$p(K = 2|\alpha) = \frac{\alpha}{\alpha^{(2Y)}} \frac{(2Y)!}{Y!} \left(\alpha \cdot 2 \cdot \ldots \cdot Y \,+\, 1 \cdot \alpha \cdot \ldots \cdot Y \,+\, \ldots \,+\, 1 \cdot 2 \cdot \ldots \cdot \alpha\right). \tag{32}$$

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 $y \le Y$, that points to the second node (and contributes "one α") and one transition indexed $y < y' \le 2Y$ that goes to the third node. This transition may come 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 $w^K = (w_1^K, w_2^K, \ldots, w_{(K-1)Y}^K)$ of length $(K - 1)Y$, 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 the definition of "legal" configuration.

Definition 2 A configuration w^K is legal if, for all $0 < k < K$, we have that $L_{k-1} < L_k < Y(K - 1)$. 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. $z(w^K) = \prod_{i=1}^{Y(K-1)} i \cdot w_i^K$. The combinatorial quantity that we need for computing the probability of having K nodes, denoted as $\phi(K)$, is the sum of such quantities for all legal configurations, i.e.

$$\phi(K) = \sum_{w^K \in W^K} z(w^K). \tag{33}$$

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

$$p(K|\alpha) = \frac{\alpha^K}{\alpha^{(KY+1)}} \frac{(KY)!}{((K - 1)Y)!} \phi(K), \tag{34}$$

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;

- $\alpha^{(KY+1)} = \alpha(\alpha+1)(\alpha+2)\dots(\alpha+KY)$ is the rising factorial, resulting from the product of the denominators of the CRP conditional distributions;
- $\frac{(KY)!}{((K-1)Y)!} = \big((K-1)Y \cdot ((K-1)Y+1) \cdot \dots \cdot KY\big)$ 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;
- $\phi(K)$ is the sum of products of legal configurations, described above.

Efficient computation

A brute-force computation of the ϕ terms in Eq. 15, according to the formula in Eq. 33, would have exponential complexity. In the following, we instead describe a way to compute $\phi(K)$ more efficiently. Let us introduce the quantity $\phi(K,l)$, 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 $Y(K-1)$, we have that $\phi(K) = \sum_{l=K-1}^{Y(K-1)} \phi(K,l)$. In order to make its manipulation easier, we decompose $\phi(K,l)$ into the sum of products of the configurations truncated at index l included, denoted as $\bar{\phi}(K,l)$, and the remaining product of the configuration (which does not contain any zero), i.e.:

$$\phi(K,l) = \bar{\phi}(K,l)\ (l+1)(l+2)\dots((K-1)Y). \tag{35}$$

It follows that:

$$\phi(K) = \sum_{l=K-1}^{Y(K-1)} \bar{q}(K,l)\frac{((K-1)Y)!}{l!}. \tag{36}$$

We can now derive a recursive relation for $\bar{\phi}(K,l)$ from $\bar{\phi}(K,l-1)$. When "moving" the position of the last α from $(l-1)$ to l, we have to multiply the previous $\bar{\phi}$ by $(l-1)$, since in the corresponding configuration the element w^K_{l-1} 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 $(l-1) < (K-2)Y$. Putting all this together, we have:

$$\bar{\phi}(K,l) = (l-1)\ \bar{\phi}(K,l-1) + \begin{cases} \bar{\phi}(K-1,l-1) & \text{if } (l-1) < (K-2)Y \\ 0 & \text{otherwise.} \end{cases} \tag{37}$$

Computing the values of ϕ in this way has a complexity of $O(K^2)$, much lower than $O(2^K)$ that results from direct computation of Eq. 33. Moreover, these values can be pre-computed and stored, since they are not dependent on α, and used when needed.

Appendix 2: Tiger problem specifications

Table 1 Specification of the "standard" multiagent Tiger problem

Transition function				*i*' s reward function			*j*' s reward function		
(a_i, a_j)	s	**TL**	**TR**	(a_i, a_j)	**TL**	**TR**	(a_i, a_j)	**TL**	**TR**
(L, L)	TL	1	0	$(L, *)$	−1	−1	$(*, L)$	−1	−1
(L, L)	TR	0	1	(OL, L)	−100	10	(L, OL)	−100	10
$(OL, *)$	$*$	0.5	0.5	(OL, OL)	−100	10	(OL, OL)	−100	10
$(OR, *)$	$*$	0.5	0.5	(OL, OR)	−100	10	(OR, OL)	−100	10
$(*, OL)$	$*$	0.5	0.5	(OR, L)	10	−100	(L, OR)	10	−100
$(*, OR)$	$*$	0.5	0.5	(OR, OL)	10	−100	(OL, OR)	10	−100
				(OR, OR)	10	−100	(OR, OR)	10	−100

i's observation function Growls				Creaks				
(a_i, a_j)	s	GL	GR	(a_i, a_j)	s	S	CL	CR
$(L, *)$	TL	0.85	0.15	$(*, L)$	$*$	0.9	0.05	0.05
$(L, *)$	TR	0.15	0.85	$(*, OL)$	$*$	0.05	0.9	0.05
$(OL, *)$	$*$	0.5	0.5	$(*, OR)$	$*$	0.05	0.05	0.9
$(OR, *)$	$*$	0.5	0.5	$(OL, *)$	$*$	1/3	1/3	1/3
				$(OR, *)$	$*$	1/3	1/3	1/3

j's observation function Growls				Creaks				
(a_i, a_j)	s	**GL**	**GR**	(a_i, a_j)	s	**S**	**CL**	**CR**
$(*, L)$	TL	0.85	0.15	$(L, *)$	$*$	0.9	0.05	0.05
$(*, L)$	TR	0.15	0.85	$(OL, *)$	$*$	0.05	0.9	0.05
$(*, OL)$	$*$	0.5	0.5	$(OR, *)$	$*$	0.05	0.05	0.9
$(*, OR)$	$*$	0.5	0.5					

The observation function is factored into *growls* and *creaks*

Table 2 Specification of the cooperative multi-agent Tiger Problem with observable actions

Transition function				i's reward function			j's reward function		
(a_i, a_j)	s	**TL**	**TR**	(a_i, a_j)	**TL**	**TR**	(a_i, a_j)	**TL**	**TR**
(L, L)	TL	1	0	$(L, *)$	-1	-1	$(*, L)$	-1	-1
(L, L)	TR	0	1	(OL, L)	-100	10	(L, OL)	-100	10
$(OL, *)$	$*$	0.5	0.5	(OL, OL)	-100	**50**	(OL, OL)	-100	**50**
$(OR, *)$	$*$	0.5	0.5	(OL, OR)	-100	10	(OR, OL)	-100	10
$(*, OL)$	$*$	0.5	0.5	(OR, L)	10	-100	(L, OR)	10	-100
$(*, OR)$	$*$	0.5	0.5	(OR, OL)	10	-100	(OL, OR)	10	-100
				(OR, OR)	**50**	-100	(OR, OR)	**50**	-100

i's observation function

Growls				Creaks				
(a_i, a_j)	s	**GL**	**GR**	(a_i, a_j)	s	**S**	**CL**	**CR**
$(L, *)$	TL	0.85	0.15	$(*, L)$	$*$	**1**	**0**	**0**
$(L, *)$	TR	0.15	0.85	$(*, OL)$	$*$	**0**	**1**	**0**
$(OL, *)$	$*$	0.5	0.5	$(*, OR)$	$*$	**0**	**0**	**1**
$(OR, *)$	$*$	0.5	0.5					

j's observation function

Growls				Creaks				
(a_i, a_j)	s	**GL**	**GR**	(a_i, a_j)	s	**S**	**CL**	**CR**
$(*, L)$	TL	0.85	0.15	$(L, *)$	$*$	**1**	**0**	**0**
$(*, L)$	TR	0.15	0.85	$(OL, *)$	$*$	**0**	**1**	**0**
$(*, OL)$	$*$	0.5	0.5	$(OR, *)$	$*$	**0**	**0**	**1**
$(*, OR)$	$*$	0.5	0.5					

The changes from the standard version are highlighted in bold

Table 3 Alternative reward models for the multiagent Tiger Problem

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

The differences from the standard formulation are highlighted in bold

Table 4 Specification of the cooperative multi-agent Tiger Problem in the "Follow the Leader" scenario

Transition function				i's reward function			j's reward function		
(a_i, a_j)	s	TL	TR	(a_i, a_j)	**TL**	**TR**	(a_i, a_j)	**TL**	**TR**
(L, L)	TL	1	0	$(L, *)$	-1	-1	$(*, L)$	-1	-1
(L, L)	TR	0	1	(OL, L)	-100	10	(L, OL)	-100	10
$(OL, *)$	TL	**0.96**	**0.04**	(OL, OL)	-100	10	(OL, OL)	-100	10
$(OL, *)$	TR	**0.04**	**0.96**	(OL, OR)	-100	10	(OR, OL)	-100	10
$(OR, *)$	TL	**0.96**	**0.04**	(OR, L)	10	-100	(L, OR)	10	-100
$(OR, *)$	TR	**0.04**	**0.96**	(OR, OL)	10	-100	(OL, OR)	10	-100
$(*, OL)$	TL	**0.96**	**0.04**	(OR, OR)	10	-100	(OR, OR)	10	-100
$(*, OL)$	TR	**0.04**	**0.96**						
$(*, OR)$	TL	**0.96**	**0.04**						
$(*, OL)$	TR	**0.04**	**0.96**						

i's observation function

Growls				Creaks				
(a_i, a_j)	s	**GL**	**GR**	(a_i, a_j)	s	**S**	**CL**	**CR**
$(L, *)$	TL	**0.96**	**0.04**	$(*, L)$	$*$	**1**	**0**	**0**
$(L, *)$	TR	**0.04**	**0.96**	$(*, OL)$	$*$	**0**	**1**	**0**
$(OL, *)$	$*$	0.5	0.5	$(*, OR)$	$*$	**0**	**0**	**1**
$(OR, *)$	$*$	0.5	0.5					

j's observation function

Growls				Creaks				
(a_i, a_j)	s	**GL**	**GR**	(a_i, a_j)	s	**S**	**CL**	**CR**
$(*, L)$	TL	**0.7**	**0.3**	$(L, *)$	$*$	**1**	**0**	**0**
$(*, L)$	TR	**0.3**	**0.7**	$(OL, *)$	$*$	**0**	**1**	**0**
$(*, OL)$	$*$	0.5	0.5	$(OR, *)$	$*$	**0**	**0**	**1**
$(*, OR)$	$*$	0.5	0.5					

The position of the tiger is persistent upon openings with probability 0.96 and the agents have asymmetric growl hearing abilities. The changes from the standard version are in bold

References

1. Albrecht, S., Crandall, J., & Ramamoorthy, S. (2016). Belief and truth in hypothesised behaviours. *Artificial Intelligence*, *235*, 63–94.
2. Balle, B., Quattoni, A., & Carreras, X. (2011). A spectral learning algorithm for finite state transducers. In D. Gunopulos, T. Hofmann, D. Malerba, M. Vazirgiannis (Eds.) Machine learning and knowledge discovery in databases, *Lecture Notes in Computer Science*, vol. 6911, (pp. 156–171). Berlin, Heidelberg: Springer.
3. Bernstein, D. S., Givan, R., Immerman, N., & Zilberstein, S. (2002). The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, *27*(4), 819–840.
4. Bowling, M., & Veloso, M. (2002). Multiagent learning using a variable learning rate. *Artificial Intelligence*, *136*, 215–250.
5. Brown, G. W. (1951). Iterative solutions of games by fictitious play. In *Activity analysis of production and allocation*, (pp. 374–376). London: Wiley.
6. Carmel, D., Markovitch, S. (1996). Learning models of intelligent agents. In *Proceedings of the 13th national conference on artificial intelligence*, (pp. 62–67).

7. Celeux, G., Hurn, M., & Robert, C. P. (2000). Computational and inferential difficulties with mixture posterior distributions. *Journal of the American Statistical Association*, *95*(451), 957–970.
8. Chakraborty, D., & Stone, P. (2008). 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, (pp. 211–226).
9. Choi, J., & Kim, K. E. (2011). Inverse reinforcement learning in partially observable environments. *Journal of Machine Learning Research*, *12*, 691–730.
10. Conitzer, V., & Sandholm, T. (2007). 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.
11. Conroy, R., Zeng, Y., Cavazza, M., & Chen, Y. (2015). Learning behaviors in agents systems with interactive dynamic influence diagrams. In *Proceedings of the twenty-fourth international joint conference on artificial intelligence, IJCAI 2015*, Buenos Aires, Argentina, July 25–31, 2015, (pp. 39–45).
12. Dennett, D. C. (1971). Intentional systems. *Journal of Philosophy*, *68*(February), 87–106.
13. Doshi, P., & Gmytrasiewicz, P. J. (2006). On the difficulty of achieving equilibrium in interactive POMDPs. In *Proceedings of the 21st national conference on artificial intelligence, vol. 2*, AAAI'06, (pp. 1131–1136). AAAI Press.
14. Doshi, P., & Gmytrasiewicz, P. J. (2009). Monte Carlo sampling methods for approximating interactive POMDPs. *Journal of Artificial Intelligence Research*, *34*(1), 297–337.
15. Doshi, P., & Perez, D. (2008). Generalized point based value iteration for interactive POMDPs. In D. Fox, & C. P. Gomes (Eds.) AAAI, (pp. 63–68). AAAI Press.
16. Doshi, P., Zeng, Y., & Chen, Q. (2009). Graphical models for interactive POMDPs: Representations and solutions. *Autonomous Agents and Multi-Agent Systems*, *18*(3), 376–416.
17. Doshi-Velez, F., Pfau, D., Wood, F., & Roy, N. (2013). Bayesian nonparametric methods for partially-observable reinforcement learning. In *IEEE transactions on pattern analysis and machine intelligence***99**(PrePrints), 1.
18. Doucet, A., & Johansen, A. M. (2009). A tutorial on particle filtering and smoothing: Fifteen years later. In D. Crisan & B. Rozovsky (Eds.), *The oxford handbook of nonlinear filtering*. Oxford: Oxford University Press.
19. Escobar, M. D., & West, M. (1994). Bayesian density estimation and inference using mixtures. *Journal of the American Statistical Association*, *90*, 577–588.
20. Fudenberg, D., & Levine, D. K. (1998). The theory of learning in games. In *MIT Press series on economic learning and social evolution*. The MIT Press, Cambridge (Mass.), London.
21. Gelman, A., Carlin, J. B., Stern, H. S., & Rubin, D. B. (2003). *Bayesian data analysis* (2nd ed.). London: Chapman and Hall/CRC.
22. Gmytrasiewicz, P. J. (1995). 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, (pp. 143–155).
23. Gmytrasiewicz, P. J., & Doshi, P. (2005). A framework for sequential planning in multi-agent settings. *Journal of Artificial Intelligence Research*, *24*(1), 49–79.
24. Green, P. J., & Richardson, S. (2001). Modelling heterogeneity with and without the Dirichlet process. *Scandinavian Journal of Statistics*, *28*(2), 355–375.
25. Hansen, E. (1998). Solving POMDPs by searching in policy space. In *Proceedings of the 14th international conference on uncertainty in artificial intelligence*, (pp. 211–219).
26. Harsanyi, J. (1967). Games with incomplete information played by "Bayesian" players. *Management Science*, *14*(3), 159–182.
27. Hastings, W. K. (1970). Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, *57*(1), 97–109.
28. de la Higuera, C. (2010). *Grammatical inference: Learning automata and grammars*. New York, NY: Cambridge University Press.
29. Hjort, N. L., Holmes, C., Müller, P., & Walker, S. G. (Eds.). (2010). *Bayesian nonparametrics*. Cambridge: Cambridge University Press.
30. Jain, S., & Neal, R. M. (2004). A split-merge markov chain Monte Carlo procedure for the Dirichlet process mixture model. *Journal of Computational and Graphical Statistics*, *13*(1), 158–182.
31. Jain, S., & Neal, R. M. (2007). Splitting and merging components of a nonconjugate Dirichlet process mixture model. *Bayesian Analysis*, *2*(3), 445–472.
32. Kadane, J. B., & Larkey, P. D. (1982). Subjective probability and the theory of games. *Management Science*, *28*(2), 113–120.
33. Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, *101*, 99–134.

34. Kalai, E., & Lehrer, E. (1993). Rational learning leads to nash equilibrium. *Econometrica*, *61*(5), 1019–1045.
35. Kocsis, L., & Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In *Proceedings of the 17th European conference on machine learning*, ECML'06, (pp. 282–293). Berlin, Heidelberg: Springer.
36. Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of 11th international conference on machine learning*, (pp. 157–163). Morgan Kaufmann.
37. Liu, M., Amato, C., Liao, X., Carin, L., & How, J. P. (2015). 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, (pp. 2011–2018).
38. Liu, M., Liao, X., & Carin, L. (2011). The infinite regionalized policy representation. In L. Getoor, T. Scheffer (Eds.) *Proceedings of the 28th international conference on machine learning*, (pp. 769–776).
39. Lopes, H., Carvalho, C. M., Johannes, M. S., & Polson, N. G. (2011). Particle learning for sequential Bayesian computation. In J. M. Bernardo, M. J. Bayarri, J. O. Berger, A. P. Dawid, D. Heckerman, Smith, A. F. M., West, M. (Eds.) Bayesian Statistics 9, (pp. 317–360). Oxford: Oxford University Press.
40. Mccallum, A. K. (1996). *Reinforcement learning with selective perception and hidden State*. Ph.D. Thesis, The University of Rochester
41. Meuleau, N., Peshkin, L., Kim, K. E., & Kaelbling, L. P. (1999). Learning finite-state controllers for partially observable environments. In *Proceedings of the 15th international conference on uncertainty in artificial intelligence*, (pp. 427–436).
42. Miller, J. M., & Harrison, M. T.: Mixture models with a prior on the number of components. CoRR arXiv:1502.06241v1 [stat.ME] (2015). Preprint
43. Neal, R. M. (2000). Markov chain sampling methods for Dirichlet process mixture models. *Journal of Computational and Graphical Statistics*, *9*(2), 249–265.
44. Ng, A. Y., & Russell, S. (2000). Algorithms for inverse reinforcement learning. In *Proceedings of the 17th international conference on machine learning*, (pp. 663–670). Morgan Kaufmann.
45. Oncina, J., García, P., & Vidal, E. (1993). Learning subsequential transducers for pattern recognition interpretation tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *15*(5), 448–458.
46. Paisley, J., & Carin, L. (2009). Hidden Markov models with stick-breaking priors. *IEEE Transactions on Signal Processing*, *57*(10), 3905–3917.
47. Papadimitriou, C., & Tsitsiklis, J. N. (1987). The complexity of Markov decision processes. *Mathematics of Operations Research*, *12*(3), 441–450.
48. Pfau, D., Bartlett, N., & Wood, F. (2010). Probabilistic deterministic infinite automata. In *Advances in neural information processing systems*, (pp. 1930–1938).
49. Pineau, J., Gordon, G., & Thrun, S. (2003). Point-based value iteration: an anytime algorithm for POMDPs. In *Proceedings of the 18th international joint conference on artificial intelligence, IJCAI'03*, (pp. 1025–1030). San Francisco, CA: Morgan Kaufmann Publishers Inc.
50. Polich, K., & Gmytrasiewicz, P. (2007). Interactive dynamic influence diagrams. In *Proceedings of the 6th international joint conference on autonomous agents and multiagent systems*, AAMAS '07, (pp. 341–343). New York, NY: ACM.
51. Poupart, P., Boutilier, C. (2003). Bounded finite state controllers. In *Advances in neural information processing systems* 16.
52. Powers, R., & Shoham, Y. (2005). Learning against opponents with bounded memory. In *Proceedings of the 19th international joint conference on artificial intelligence, IJCAI'05*, (pp. 817–822). San Francisco, CA: Morgan Kaufmann Publishers Inc.
53. Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, (pp. 257–286).
54. Ramachandran, D., & Amir, E. (2007). Bayesian inverse reinforcement learning. In *Proceedings of the 20th international joint conference on artical intelligence, vol. 51*, pp. 2586–2591.
55. Ross, S., Draa, B. C., & Pineau, J. (2007). Bayes-adaptive POMDPs. In *Proceedings of the conference on neural information processing systems*.
56. Russell, S., & Norvig, P. (2009). *Artificial intelligence: A modern approach* (3rd ed.). Englewood Cliffs, NJ: Prentice Hall.
57. Shoham, Y., & Leyton-Brown, K. (2008). *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. New York, NY: Cambridge University Press.
58. Silver, D., & Veness, J. (2010). Monte-Carlo planning in large POMDPs. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, & A. Culotta (Eds.), *Advances in neural information processing systems 23* (pp. 2164–2172). Curran Associates Inc.
59. Sondik, E. J. (1978). The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. *Operations Research*, *26*(2), 282–304.

60. Sonu, E., & Doshi, P. (2012). Generalized and bounded policy iteration for interactive POMDPs. In *International symposium on artificial intelligence and mathematics (ISAIM)*.
61. Wright, J. R., & Leyton-Brown, K. (2012). 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), (pp. 921–930).
62. Yoshida, W., Dolan, R. J., & Friston, K. J. (2008). Game theory of mind. *PLoS Comput Biol*, *4*(12), e1000,254+.
63. Zeng, Y., & Doshi, P. (2012). Exploiting model equivalences for solving interactive dynamic influence diagrams. *Journal of Artificial intelligence Research*, *43*(1), 211–255.
64. Ziebart, B. D., Maas, A., Bagnell, J. A., & Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd national conference on artificial intelligence, vol. 3, AAAI'08*, (pp. 1433–1438). AAAI Press.