Towards a fast detection of opponents in repeated stochastic games

Pablo Hernandez-Leal and Michael Kaisers
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
{Pablo.Hernandez, Michael.Kaisers}@cwi.nl

ABSTRACT

Multi-agent algorithms aim to find the best response in strategic interactions. While many state-of-the-art algorithms assume repeated interaction with a fixed set of opponents (or even *self-play*), a learner in the real world is more likely to encounter the same strategic situation with changing counter-parties. This article presents a formal model of such *sequential interactions*, and a corresponding algorithm that combines the two established frameworks *Pepper* and *Bayesian policy reuse*. For each interaction, the algorithm faces a repeated stochastic game with an unknown (small) number of repetitions against a random opponent from a population, without observing the opponent's identity. Our algorithm is composed of two main steps: first it draws inspiration from multiagent algorithms to obtain acting policies in stochastic games, and second it computes a belief over the possible opponents that is updated as the interaction occurs. This allows the agent to quickly select the appropriate policy against the opponent. Our results show fast detection of the opponent from its behavior, obtaining higher average rewards than the state-of-the-art baseline Pepper in repeated stochastic games.

CCS Concepts

•**Computing methodologies** → **Multi-agent systems;**

Keywords

Stochastic games; reinforcement learning; policy reuse

1. INTRODUCTION

Learning to act in multiagent systems has received attention mainly from game theory and reinforcement learning (RL). The former has proposed algorithms that converge under different scenarios [14] and the latter has focused on acting optimally in stochastic scenarios [12]. Interactions among several agents are usually modelled as a normal-form or stochastic game, and a wide variety of learning algorithms targets this setting [7, 9, 13]. However, results are typically based on the assumption of self-play (i.e., all participants use the same algorithm) and a long period of repeated interactions. In contrast, we focus on *sequential interactions*, i.e., the agent is paired with stochastically drawn opponents, with whom the agent interacts in short periods while observing joint actions, but without observing the opponent's identity.

Recent works have proposed algorithms for learning in repeated stochastic games [17, 20], however, it is an open problem how to act quickly and optimally when facing different opponents [20]. Recent work on Stochastic Bayesian Games has compared several ways to incorporate observations into beliefs over opponent types when those types are re-drawn after every state transition [1]. In contrast, we assume that opponents are redrawn for interactions over several repeated stochastic games. The learning algorithm needs to optimally reuse previously learned information from distinct but similar interactions – a challenge that has been largely studied by transfer learning (TL). TL has been applied mostly in single-agent domains where information from learned *source* tasks can be reused in a new target task [30]. Determining how two tasks are similar, what information to be transferred and when it should be transferred are open problems in TL. Related to TL there are different areas that also share a connection with the problem of how to efficiently reuse previously learned information, e.g., policy reuse [21], ad-hoc coordination [4] and learning in non-stationary environments [18].

We contribute to the state of the art in two ways: First, by providing a more natural formal model of *sequential interactions* and second with an algorithm for quick detection of opponents in that setting. Our proposed algorithm Bayes-Pepper builds on top of two previously successful frameworks:

- Pepper [15], a learning algorithm for repeated stochastic games, is used to obtain policies on how to act against the possible opponents. Pepper uses the paradigm of optimism under uncertainty together with a joint action learner to learn a policy in stochastic games.
- Bayesian policy reuse (BPR) [29] is used as a fast detection process to identify the opponent and select the appropriate acting policy. While previously BPR has been evaluated in single-agent tasks and repeated normal-form games, this is the first time it is extended to stochastic games.

Our setting assumes a population of opponents that can be divided into different groups. First, Bayes-Pepper needs to compute policies and models of the opponents, which we assume happens at an offline phase. Second, in an online phase a random process pairs the learning agent against one

Appears in: *Proceedings of the 1st Workshop on Transfer in Reinforcement Learning (TiRL) at the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017), A. Costa, D. Precup, M. Veloso, M. Taylor (chairs), May 8–9, 2017, São Paulo, Brazil.*

opponent for a stochastic game. The learning agent has no control over this process and does not observe the opponent identity. When the game finishes the learning agent receives an observation (reward) and updates the belief accordingly. Subsequently, the agent is paired with a new opponent. A formal definition of the game is given in Section 3.3.

This paper is presented as follows: Section 2 presents the related work in transfer learning, policy reuse and learning in non-stationary environments. Section 3 describe the formal models of reinforcement learning and game theory. Section 4 presents the proposed Bayes-Pepper algorithm. Section 5 presents experimental results in repeated stochastic games. Section 6 provides a discussion considering the results and provides directions for future research. Finally, Section 7 summarizes the conclusions of this work.

2. RELATED WORK

This article tackles the problem of finding a best response when being repeatedly paired with unknown opponents from an unknown population with known types. We propose an algorithm that aims to identify opponents while best-responding in face of residual uncertainty. Our setting and approach shares similarities to transfer learning, policy reuse and ad-hoc coordination which we review in this section.

Transfer learning was first used in machine learning to transfer between learning tasks in a supervised learning scenario. Recently, TL has gained attention in the RL community in particular in single-agent scenarios. An ideal fully autonomous RL transfer agent needs to complete three phases [30]:

- Given a target task, select an appropriate set of source tasks from which to transfer.
- Learn how the source task(s) and target task are related.
- Transfer knowledge from the source task(s) to the target task.

There are different evaluation metrics for TL algorithms (e.g., jumpstart, asymptotic performance, total rewards, among others) and even when these three steps are usually connected, TL has focused on them independently. For example, for the transfer step different ideas have been evaluated, e.g., models, instances and policies.

One approach that transfers instances from similar tasks was proposed by Lazaric et al. [28]. They proposed a measure to identify which source tasks are more likely to have samples similar to those in the target task, namely *task compliance*. Moreover, to select which instances to transfer from a task they propose the *relevance* measure. However, the approach was proposed for single-agent domains with continuous state and action spaces, and does not naturally transfer to our setting. Closer to our approach, Boutsioukis et al. [8] proposed TL by extending the Q-learning reuse algorithm [31] to multiagent scenarios. In contrast to our ambition of transfer from interactions against different opponents, their goal is to transfer information learned from a task with n agents to a different task with $m \neq n$ agents. In particular, they propose an inter-task transfer approach (i.e., the state and action spaces are not the same in the target and source tasks) and the evaluation was performed on the predator-prey domain transferring information tasks learned with different number of predators (agents). Policy reuse techniques are another area with a similar spirit since these approaches assume to start with a set of policies to use, and the problem is to select among them when facing a new task. Fernandez and Veloso [21] use policy reuse as a probabilistic bias when learning of new, similar tasks in single agent domains. Bayesian Policy Reuse (BPR) [29] assumes prior knowledge of the performance of different policies over different tasks. BPR computes a belief over the possible tasks which is updated at every interaction and is used to select the policy which maximises the expected reward given the current belief. Bayesian reasoning has been used in RL to learn when there is a group of related tasks with similar structure. Lazaric and Ghavamzadeh [27] proposed an algorithm assuming tasks have common state and action spaces and their value functions are sampled from a common prior. In contrast, our approach extends the BPR algorithm (see Section 3.2) to identify opponents rather than tasks, and combine it with a multiagent learning algorithm.

Ad-hoc coordination is another related problem where an agent needs to coordinate with an unknown agent but when a set of previous models is known. In this setting, Barrett et al. [4] proposed the PLASTIC algorithm which learns how to cooperate with other teammates based on a collection of policies to select from, which is similar to our approach. The algorithm selects at each interaction the most likely teammate type and acts following the corresponding policy. However, this approach do not consider changing agents over the course of interactions as we do.

Learning in non-stationary environments is another related area since these approaches explicitly model changes in the environment. Their goal is to learn an optimal policy and at the same time detect when the environment has changed to a different one, updating the acting policy accordingly. One algorithm designed for single agent tasks with a changing environment is the Reinforcement Learning with Context detection (RL-CD) [18]. RL-CD learns a model of the specific task and assumes an environment that changes infrequently among different *contexts*. To detect a new context RL-CD computes a quality measure of the learned models. Hernandez-Leal et al. [23, 25] addressed a similar problem in two-player repeated normal-form games. In this case, the opponent has different stationary strategies to select from and the learning agent needs to learn online how to act optimally against each strategy while detecting when the opponent changes to a different one. Since the opponent might reuse one previous strategy at a later stage of the interaction the learning agent should keep previous models and policies in order to quickly detect them [24]. While this might be the closest state of the art, these approaches do not consider repeated stochastic games.

Experimental evidence suggests that people learn heuristics which later are transferred across different games [5]. Based on these results there is another category of algorithms that aims to learn in one game and generalize how to act in a different game (known as general game playing). Banerjee and Stone [3] proposed a transfer approach in two-player, alternate move, complete information games facing stationary opponents. The idea is to learn general features that can be reused across games, for example, they learned from played games on Tic-tac-toe and transferred information to more complex game like Othello.

In contrast with previous approaches we focus on two-

player repeated stochastic games. An agent faces an opponent whose identity is unknown to the agent and where every few interactions a random process selects a new opponent from the population (the agent does not know when these changes happen).

3. PRELIMINARIES

In this section, first we review the formal model of reinforcement learning. Then, we describe the Bayesian policy reuse framework [29]. Later, we present our sequential interactions model in the context of and stochastic games. Finally, we describe Pepper [15] which inspire our proposed Bayes-Pepper algorithm.

3.1 Reinforcement learning

Reinforcement learning (RL) is one important area of machine learning that formalizes the interaction of an agent with its environment using a Markov decision process (MDP). An MDP is defined by the tuple $\langle S, A, R, T\rangle$ where S represent the world divided up into a finite set of possible states. A represents a finite set of available actions. The transition function $T : S \times A \to \Delta(S)$ maps each state-action pair to a probability distribution over the possible successor states, where $\Delta(S)$ denotes the set of all probability distributions over S. Thus, for each $s, s' \in S$ and $a \in A$, the function T determines the probability of a transition from state s to state s' after executing action a. The reward function $R : S \times A \times S \to \mathbb{R}$ defines the immediate and possibly stochastic reward that an agent would receive for being in state s, executing action a and transitioning to state s'.

MDPs are adequate models to obtain optimal decisions in *single* agent environments. Solving an MDP will yield a policy $\pi : S \to A$, which is a mapping from states to actions. An optimal policy π^* is the one that maximises the expected reward. There are different techniques for solving MDPs assuming a complete description of all its elements. One of the most common techniques is the value iteration algorithm [6] which is based on the Bellman equation:

$$V^{\pi}(s) = \sum_{a \in A} \pi(s, a) \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V^{\pi}(s')],$$

with $\gamma \in [0, 1)$. This equation expresses the *value* of a state which can be used to obtain the optimal policy $\pi^* = \arg\max_{\pi} V^{\pi}(s)$, i.e., the one that maximises that value function, and the optimal value function $V^*(s)$.

$$V^*(s) = \max_{\pi} V^{\pi}(s) \quad \forall s \in S.$$

Value iteration requires complete and accurate representation of states, actions, rewards and transitions. However, this may be difficult to obtain in many domains. For this reason, RL algorithms learn from experience without having a complete description of the MDP a priori. In contrast, an RL agent interacts with the environment in discrete timesteps. At each time, the agent chooses an action from the set of actions available, which is subsequently executed in the environment. The environment moves to a new state and the reward associated with the transition is emitted. The goal of a RL agent is to maximise the expected reward. In this type of learning the learner is not told which actions to take, but instead must discover which actions yield the best reward by trial and error.

Q-learning [32] is one well known algorithm for RL. It has been devised for stationary, single-agent, fully observable environments with discrete actions. In its general form, a Q-learning agent can be in any state $s \in S$ and can choose an action $a \in A$. It keeps a data structure $\hat{Q}(s, a)$ that represents the estimate of its expected payoff starting in state s, taking action a. Each entry $\hat{Q}(s, a)$ is an estimate of the corresponding optimal Q^* function that maps state-action pairs to the discounted sum of future rewards when starting with the given action and following the optimal policy thereafter. Each time the agent makes a transition from a state s to a state s' via action a receiving payoff r, the Q table is updated as follows:

$$\hat{Q}(s, a) = \hat{Q}(s, a) + \alpha[(r + \gamma \max_{b} \hat{Q}(s', b)) - \hat{Q}(s, a)]$$

with the learning rate α and the discount factor $\gamma \in [0, 1]$ being parameters of the algorithm, with α typically decreasing over the course of many iterations. Q-learning is proved to converge towards Q^* if each state-action pair is visited infinitely often under specific parameters [32].

3.2 Bayesian policy reuse

Bayesian policy reuse is a framework to quickly determine the best policy to select when faced with an unknown task. Formally, a task is defined as an MDP. A *policy* is a function $\pi(s)$ that specifies an appropriate action a for each state s. The return, or utility, generated from running the policy π in an interaction of a task instance is the accumulated reward, $U^{\pi} = \sum_{i=0}^{k} r_i$, with k being the length of the interaction and r_i being the reward received at step i.

Let an agent be equipped with a policy library Π for tasks in a domain. The agent is presented with an unknown task which must be solved within a limited and small number of trials. At the beginning of each trial episode, the agent can select one policy from $\pi \in \Pi$ to execute. The goal of the agent is thus to select policies to minimize the total regret incurred in the limited task duration with respect to the performance of the best alternative from Π in hindsight.

BPR assumes knowledge of *performance* models describing how policies behave on different tasks. A performance model, $P(U|\tau, \pi)$, is a probability distribution over the utility using π on a task τ. A signal σ is any information that is correlated with the performance of a policy and that is provided to the agent in an online execution of the policy on a task (e.g., immediate rewards). For a set of tasks $\mathcal{T}$ and a new instance $\tau^{\star}$ the *belief* β is a probability distribution over $\mathcal{T}$ that measures to what extent $\tau^{\star}$ matches the known tasks in their observation signals σ. The belief is initialized with a prior probability. After each execution on the unknown task the environment provides an observation signal to the agent, which is used to update beliefs according to Bayes' rule:

$$\beta^k(\tau) = \frac{\Pr(\sigma^k|\tau, \pi^k)\beta^{k-1}(\tau)}{\sum_{\tau' \in \mathcal{T}} \Pr(\sigma^k|\tau', \pi^k)\beta^{k-1}(\tau').} \tag{1}$$

Different mechanisms can be used to select a policy to execute. An always greedy policy selection mechanism would fail to explore, resulting in not reaching the global maximum. On the other hand a totally exploratory policy selection mechanism would not make an effort to improve performance. We thus require a balance, for which different policy

selection heuristics have been proposed [29]. A policy selection heuristic $\mathcal{V}$ is a function that estimates a value for each policy through the extent to which it balances exploration with a limited degree of look-ahead for exploitation.

The *probability of improvement* heuristic for policy selection [29] considers the probability with which a specific policy can achieve a hypothesized increase in performance over the current best estimate. Assume that $U^+ \in \mathbb{R}$ is some utility which is larger than the best estimate under the current belief,

$$\hat{U} = \max_{\pi \in \Pi} \sum_{\tau \in \mathcal{T}} \beta(\tau) E[U|\tau, \pi].$$

The heuristic thus chooses the policy

$$\arg\max_{\pi \in \Pi} \sum_{\tau \in \mathcal{T}} \beta(\tau) \Pr(U^+|\tau, \pi),$$

where $U^+ > \hat{U}$.

3.3 Games

In contrast to classical RL, which considers one single agent in a stationary environment, Game theory studies rational decision making when several agents interact [22]. The core concept of a *Game* captures the strategic conflict of interest in a mathematical model. Note that different areas provide different terminology. Therefore, we will use the terms player and agent interchangeably; similarly for reward and payoff. Finally, we will refer to other agents in the environment as opponents irrespective of the domain's or agent's cooperative or adversarial nature.

A stochastic game with two players i and $-i$ consists of a set of stage games S (also known as states). In each stage s players choose an action from the set $\mathbf{a} \in A(s)$. A game begins in a state $s_b \in S$. A joint action $\mathbf{a} = (a_i, a_{-i})$ is played at stage s and player i receives an immediate reward $r_i(s, \mathbf{a})$, the world transitions into a new stage s' according to the transition model $T(s, s', \mathbf{a})$. When a goal state $s_g \in S$ is encountered the game finishes, the accumulated reward during the game is called an *episodic* reward.

We formalize *Sequential Interactions* (SI) as a specific variation of repeated stochastic games, where at each episode $k \in \{1, 2, \ldots, K\}$ a process draws a set of players $P_k \subset I$ from the population of individuals I to play a finite stochastic game that yields a reward (accumulated over the game) to each player. After the stochastic game terminates, the subsequent interaction commences. We specifically discuss the setting where the selection process is *stochastic* (as opposed to being a *strategic choice* by the agents), and the population comprises an unknown distribution over types of strategies. While in the general case these types may be unknown, our new algorithm assumes access to a priori interactions with each (proto-) type. We consider P_k and opponent rewards within the stochastic game to be unobservable, while the joint actions are observable.

3.4 Pepper

Pepper [15] (potential exploration with pseudo stationary restarts) was proposed as a framework to extend algorithms for learning in repeated normal-form games to repeated stochastic games. Pepper assumes it can observe its own immediate reward but not the opponents', and also assumes the maximum possible reward R_{max} known for each episode. It uses the principle of optimism in face of uncertainty [10] and combines it with a learning algorithm. Pepper computes the expected future rewards for a joint action $\mathbf{a}$ being in state s as:

$$R(s, \mathbf{a}) = r(s, \mathbf{a}) + \sum_{s' \in S} T(s, s', \mathbf{a}) V(s') \tag{2}$$

where $V(s')$ is the expected future rewards of being in state s'. Note that given $r(\cdot), T(\cdot)$ and $V(\cdot)$, value iteration can be used to compute Equation 2, and r and T can be learned from observations. Moreover, Pepper is initialized under the assumption that all states result in maximal reward. However, there is still the problem of updating $V(s')$ throughout the interaction. Pepper proposes a mechanism for estimating future rewards combining off-policy (e.g., Q-learning) and on-policy methods for estimating $V(s)$, i.e., an on-policy estimation based on the observed distribution of joint actions, using $n(s), n(s, \mathbf{a})$ for the number of visits to state s and the number of times joint action $\mathbf{a}$ was chosen in that state respectively:

$$V^{on}(s) = \sum_{\mathbf{a} \in A(s)} \frac{n(s, \mathbf{a})}{n(s)} R(s, \mathbf{a})$$

and a combined estimation

$$V(s) = \lambda(s) \hat{V}(s) + (1 - \lambda(s)) V^{on}(s).$$

Where $\hat{V}(s)$ represents an optimistic approximation given by

$$\hat{V}(s) = \max(V^{\text{off}}(s), V^{\text{on}}(s))$$

and where $\lambda \in [0, 1]$ represents a stationarity measure initialized to one but approaching zero when the agent gets more experience.[1] Pepper uses the concept of non-pseudo stationary restarts, i.e., when $R(s)$ is observed to not be pseudo stationary $\lambda(s)$ resets to one. Let $n'(s)$ be the number of visits to stage s since $R(s)$ was last observed to not be pseudo-stationary, then:

$$\lambda(s) = \max\left(0, \frac{C - n'(s)}{C}\right)$$

with $C \in \mathbb{N}^+$.

The Pepper framework is described in Algorithm 1 where different policy selection approaches can be plugged to compute π. For example, using

$$\max_{\mathbf{a}} R(s, \mathbf{a})$$

seems suitable for a friendly opponent, while

$$\max_{\mathbf{a}_i} \min_{\mathbf{a}_{-i}} R(s, \mathbf{a})$$

is a minimax approach that suits other types of opponents.

Next, we present our Bayes-Pepper approach which uses Pepper in an offline phase to obtain policies. During the online phase it computes a belief over the possible opponents to tackle the uncertainty over the opponent's identity in a sequential interaction.

4. BAYES-PEPPER

Bayes-Pepper is composed mainly of two phases which are depicted in Figure 1.

[1] Recall that $R(s, a)$ is initialized to R_{max} so it is likely to decrease in early episodes, but eventually will become pseudo-stationary.

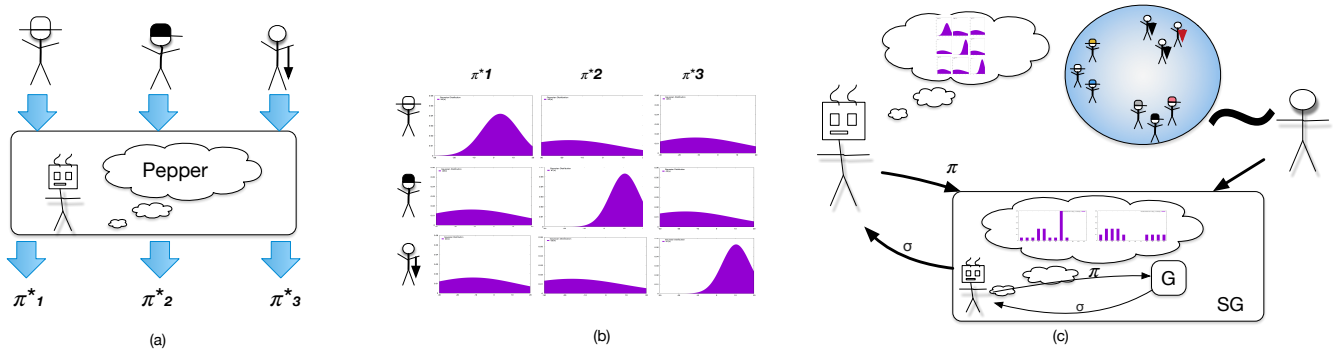


Figure 1: Bayes-Pepper algorithm: (a) in an offline phase, Pepper is used to generate policies against each opponent and (b) performance models and transition models are generated from the learned policies. (c) In the online phase (i.e., sequential interactions) we assume a population of agents and a random process that matches the Bayes-Pepper agent and one opponent. Bayes-Pepper selects a policy π to act, in a stochastic game (SG) the agent computes an intra-belief which might override the selected policy; when the game finishes Bayes-Pepper receives an observation σ that is used to update its belief.

Algorithm 1: Pepper algorithmic framework

Input: States S, maximum possible reward R_{max}
1 Initialize $V(\cdot)$ with R_{max}
2 Random initial policy π
3 **for** *each episode of the stochastic game* **do**
4 Update $R(\cdot)$; *Eq.*2
5 Update policy π
6 Observe state
7 **while** *state is not goal* **do**
8 Select action a
9 Observe state
10 Receive observation r
11 **if** *enough visits to* $(s, \boldsymbol{a})$ **then**
12 Update rewards, $V(\cdot)$, transitions
13 Update $R(\cdot)$; *Eq.*2
14 Update policy π

- An offline phase where Bayes-Pepper generates policies and performance models. Here, the agent observes the opponent's identity.
- An online phase where a belief based approach is used to detect the opponent's identity and act with the corresponding policy. Here, the agent observes states and actions at every step of the game but only observes the accumulated reward when a stochastic game finishes. The belief is updated at every stochastic game (see Section 4.2) and at every stage in a stochastic game (see Section 4.3).

Next, we describe these two phases in more detail.

4.1 Policy and models generation

Bayes-Pepper needs to generate policies and models for each opponent. Bayes-Pepper assumes an offline learning phase (see Algorithm 2) in which Pepper algorithm is used to obtain a policy for every opponent (lines 3–5). Then, performance models are obtained by generating list of rewards for each opponent and policy and fitting the generated data into a distribution (in our experiments we used Gaussian

Algorithm 2: Bayes-Pepper models and policy generation

1 $\Pi = \emptyset$
2 **for** *every opponent* $\tau \in \mathcal{T}$ **do**
3 Opponent τ is announced
4 Bayes-Pepper learns a policy π_{new} facing τ
5 $\Pi = \Pi \cup \pi_{new}$
6 **for** *every opponent* $\tau \in \mathcal{T}$ **do**
7 **for** *every* $\pi \in \Pi$ **do**
8 Get list of rewards **r** and **[s,a]** pairs using π against τ
9 Fit **r** to a distribution to obtain $\Pr(U|\tau, \pi)$
10 Fit **[s,a]** to a distribution to obtain $\Pr(M|\tau, \pi)$

distribution). The generated set of performance models can be seen as a matrix of probability distributions, see Figure 1 (b). Similarly, a list of state-action pairs is generated and fitted to a multinomial distribution to generate a transition model (lines 6–10), used in the intra-game belief, see Figure 1 (c).

4.2 Opponent detection based on rewards

Once Bayes-Pepper has a set of policies Π and its associated models it can act in an online mode. The steps of Bayes-Pepper online detection phase are described in Algorithm 3. Bayes-Pepper starts with a set of policies Π, prior probabilities over the opponents $\Pr(\mathcal{T})$, performance models $\Pr(U|\mathcal{T}, \Pi)$ and transition models $\Pr(M|\mathcal{T}, \Pi)$. Bayes-Pepper initializes the belief over tasks with the prior probabilities $\Pr(\mathcal{T})$ (line 1). Then, for each episode of the sequential interaction a loop performs the following steps:

- select a policy to execute (according to the belief β and exploration heuristic $\mathcal{V}$) (lines 3–4),
- use the selected policy on the stochastic game (line 5),
- receive an observation signal σ, this is, the accumulated reward of the played game (line 6),

Algorithm 3: Bayes-Pepper detection algorithm

Input: Policy library Π, prior probabilities $\Pr(\mathcal{T})$, performance models $\Pr(U|\mathcal{T},\Pi)$, transition models $\Pr(M|\mathcal{T},\Pi)$, episodes K, exploration heuristic $\mathcal{V}$

1 Initialize beliefs $\beta^0(\mathcal{T}) = \Pr(\mathcal{T})$
2 **for** *episodes* $k = 1,\ldots,K$ **do**
3 Compute $v_\pi = \mathcal{V}(\pi, \beta^{k-1})$ for all $\pi \in \Pi$
4 $\pi^k = argmax_{\pi\in\Pi} v_\pi$
5 Start game with policy π_k and use intra-game belief (ζ) together with $\Pr(M|\mathcal{T},\Pi)$ (see Section 4.3)
6 Obtain observation signal σ^k (e.g., episodic reward)
7 Update belief $\beta^k(\tau) = \frac{\Pr(\sigma^k|\tau,\pi^k)\beta^{k-1}(\tau)}{\sum_{\tau'\in\mathcal{T}}\Pr(\sigma^k|\tau',\pi^k)\beta^{k-1}(\tau')}$

- update the belief with the observation using Eq. 1 (line 7).

Since we assume that only the accumulated reward for the game is observed when the game finishes, a basic approach is to select a policy to play for the entire stochastic game. However, this might result in suboptimal results for example when the opponent changes in the next interaction (as experimentally shown in Section 5.3). To overcome this issue, we propose to update the belief during a stochastic game using the state-action pairs (which are always observable). This information is also a signal of the opponent behavior and the process is similar to the one described previously.

4.3 Intra-game belief detection

Let $\zeta^\ell(\tau)$ be the *intra-game* belief which is initialized with the belief $\beta(\tau)$. The intra-game belief is updated in a similar way using Eq. 1 with two minor differences, the observation σ is the observed frequency over state-action pairs (or states) and the transition models $P(M|\mathcal{T},\Pi)$ are used to obtain the likelihood of a given observation.

Since the observed frequency might change more in early stages of the game we consider weighted approach, initially giving more weight to β and with each experience giving more weight to ζ as follows:

$$\zeta^\ell(\tau) = w\beta(\tau) + (1-w)\zeta^{\ell-1}(\tau) \quad (3)$$

with $w = 1$ initially and $w = w \cdot \eta^t$, with $\eta = [0,1)$ and t the number of experienced steps in the current stochastic game. Computing this updated belief might override the policy that was selected initially which is useful to avoid using a suboptimal policy for a complete stochastic game.

5. EXPERIMENTS

In this section we present results on a stochastic game represented as a grid world. We performed experiments comparing our approach Bayes-Pepper with Pepper [15] and an Omniscient agent that knows the opponent's identities and plays optimal policies against them. First, we define the setting, then we present results of Bayes-Pepper against stochastic opponents and finally, we present results in sequential interactions against switching opponents.

5.1 Setting

Figure 2 depicts a graphical representation of the stochastic game used in the experiments. There are two players, the learning agent (A) and the opponent (O). The starting positions are marked with their initial. The learning agent receives a reward when it reaches a goal state r_A or R_A, with $r_a < R_A$. The agents can move upwards or horizontally, and the opponent has the possibility to stay in the same state; the learning agent moves always to the right and the opponent to the left; to avoid agents getting trapped the grid is a toroidal world. With every step that does not transition to a goal state the learning agent receives a penalty p_{min}. In case of collision the learning agent receives high penalty p_{max} with $p_{min} < p_{max}$.

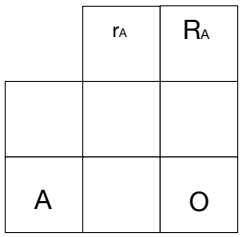


Figure 2: A stochastic games with two players, the learning agent (A) and one opponent (O). The learning agent receives a reward when it reaches state marked with r_A or R_A with $r_a < R_A$. In case of collision the opponent has priority over the state.

Note that the opponent directly influences the possible reward the learning agent can obtain. For example, since the opponent is closer to R_A it can block its way to the learning agent, in which case the best option would be to go for r_A.

For the experiments we set $r_A = 5, R_A = 17, p_{min} = -1, p_{max} = -5$. We test against two types of stochastic opponents:

- A *defecting* opponent, Opp_D, that aims to block the way to R_A. It stays in the blocking position with probability 0.8.
- A *cooperative* opponent, Opp_C, that ignores the learning agent's actions and moves upwards with probability 0.2, and left otherwise.

The optimal policy against Opp_D is to go directly to r_A obtaining an accumulated reward of 3. In contrast, when facing Opp_C the agent should go for R_A obtaining an accumulated reward of 14.

5.2 Opponent detection

In this experiment we evaluated how quickly Bayes-Pepper responds without knowing the opponent's identity in comparison with the learning process of Pepper. In this case, Bayes-Pepper starts with the policies against Opp_C and Opp_D and with a uniform prior over them.

Figure 3 depicts the average episodic rewards against the two types obtained over 10 independent trials facing the same type during the interaction. From the results we observe that Bayes-Pepper obtains higher rewards from the beginning of the interaction due to its fast detection. In contrast, Pepper takes more episodes to learn the appropriate policy. Table 1 shows average rewards where it can be seen that Bayes-Pepper obtained similar rewards to those of the the Omniscient agent.

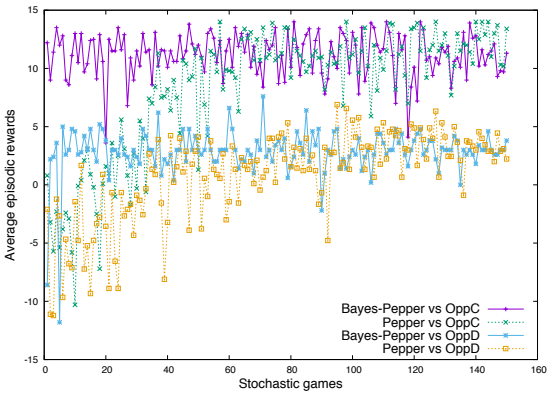


Figure 3: Comparison of Bayes-Pepper and Pepper against two stochastic opponents in a repeated SG for 150 episodes. Bayes-Pepper obtains rewards close to the best response faster than Pepper.

Table 1: Comparison of average rewards with std. dev. (±) obtained in 10 trials.

	Bayes-Pepper	Pepper	Omniscient
Opp_C	11.19 ±5.30	8.26± 8.80	12.40± 2.30
Opp_D	2.87 ±4.05	0.87± 8.87	3.02± 2.97
Switching	5.44 ±6.74	2.33± 8.01	8.48± 5.21

5.3 Switching opponents

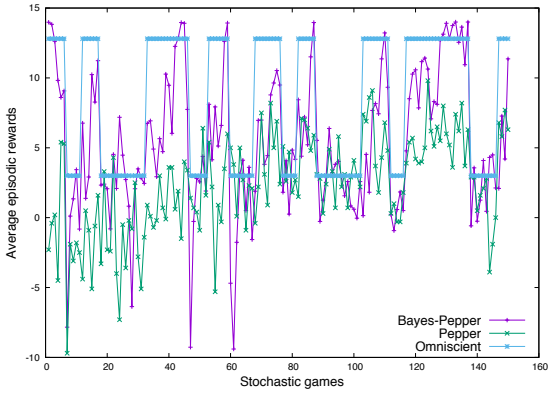
Now, we compare against switching opponents in sequential interactions, this is, during a repeated interaction of 150 games where the opponent changes frequently and the learning agent does not know when the switches happen. To model switching opponents an opponent is selected randomly and is paired with the learning agent for a random number of games (uniformly from 5 to 10 repetitions).

Figure 4 depicts the average (a) episodic and (b) cumulative rewards of the compared approaches, and Table 1 shows the average episodic rewards for the 150 games. From the results we note that Bayes-Pepper obtained higher cumulative rewards than Pepper. This happens because Bayes-Pepper knows how to act optimally against every opponent, however, it needs to identify it. In contrast, Pepper learns how to optimize against the mixed behaviour of the two types. Note that in many cases when an opponent changed Bayes-Pepper was capable of obtaining competitive scores, this happens mainly due to the intra-game detection, since it triggers a change to a different policy when the transitions are not consistent with the learned model. In contrast, even when Pepper is learning within stochastic games and is able to update its policy it obtains suboptimal results, because it fails to obtain the reward R_A against the cooperative opponent.

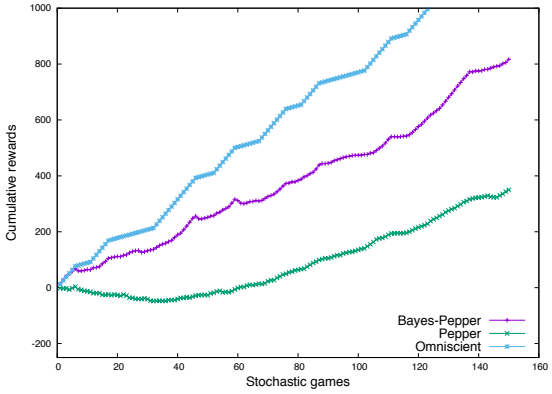
We note that the comparison of Bayes-Pepper against

Table 2: Average rewards with std. dev. (±) for the compared approaches against switching opponents (average of 10 trials).

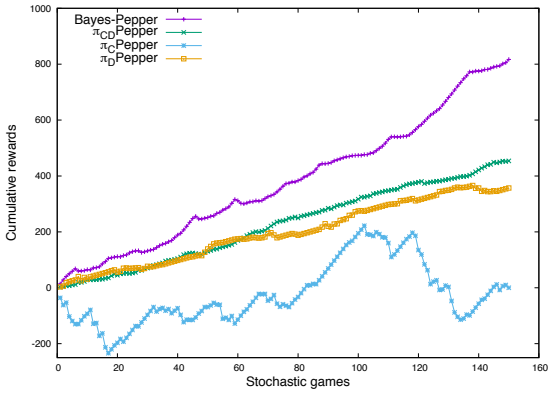
Bayes-Pepper	π_C^{Pepper}	π_D^{Pepper}	π_{CD}^{Pepper}
5.44 ± 6.74	0.46 ± 18.01	2.38 ± 4.21	3.02 ± 2.75



(a)



(b)



(c)

Figure 4: Bayes-Pepper, Pepper and the Omniscient agent against switching opponents in a repeated interaction of 150 episodes. Average (a) episodic and (b) cumulative rewards. (c) Cumulative rewards of Bayes-Pepper and policies learned with Pepper.

Pepper might not be fair since Bayes-Pepper has already policies to start. With this in mind, we also evaluated three baselines using policies learned by Pepper against switching opponents: the policies learned against a single types π_C^{Pepper}, π_D^{Pepper} and one policy learned after facing the two opponents sequentially π_{CD}^{Pepper}, in this case there is no learning during the interaction.

Table 2 shows average rewards over the 150 games and Figure 4 (c) depicts cumulative rewards of the compared approaches against the same switching opponents. Results show that Bayes-Pepper obtains better scores than the rest. On the one side π_D^{Pepper} is a cautious policy which never takes advantage of a cooperative opponent, on the other side π_C^{Pepper} quickly obtains the best scores against a cooperative opponent but also gets highly penalized against a defecting opponent. π_{CD}^{Pepper} obtains better scores than the previous two but is not as good as Bayes-Pepper.

6. DISCUSSION

We presented experiments with Bayes-Pepper against switching opponents in repeated stochastic games. The results suggests that our approach is capable of detecting the opponent type and act with the corresponding policy. Bayes-Pepper's main advantage is its quick detection in the online phase. However, its main limitation is the offline learning phase to obtain acting policies and models. To overcome this limitation we foresee different directions in which this work could be extended:

- State abstraction. CQ-learning has been used to reduce the state space representation in multi-agent systems by allowing a minimal state space representation and only expanding for conflicting states [19]. This same idea could be extended to our setting by having general policies and only update partial policies for some *dangerous* states.
- Lifelong learning [11] is another paradigm related to TL where information obtained from other sources should increase the performance on the target tasks and on the previous source tasks (reverse transfer). Currently, the offline learning phase is independent from the online detection phase, however, it would be interesting to use the information obtained in the online phase to update the policies and models learned in the offline phase.
- Multi-armed bandits [2] are a common formalism for selecting among different actions and this approach has been extended to select among experts [16]. Contextual multi-armed bandits are an extension in which the player also observes context information which can be used to determine the selection process [26]. How to incorporate contextual information in our setting is another idea for future work.

7. CONCLUSIONS

Many learning algorithms for multiagent systems assume self-play or stationary opponents. We focus on the scenario of repeated stochastic games but with the difference of assuming a population of opponents and a stochastic process that at every game matches the learning agent with an opponent. Our first contribution is to provide a formal model of the mentioned setting, namely, sequential interactions. Our second contribution is an algorithm for quick detection of opponents in repeated stochastic games. Our proposed Bayes-Pepper algorithm draws inspiration from multiagent learning algorithms and policy reuse approaches to detect opponents. One advantage is that Bayes-Pepper is capable of detecting the opponent and responding with the appropriate policy faster than learning algorithms for repeated stochastic games. The main limitation is the need of an offline learning phase where the policies and models can be obtained. As future work we propose to not only reuse policies but also transfer information from models and policies when facing unknown opponents, and eventually learning the set of opponent strategies in the population online.

Acknowledgments

This research has received funding through the ERA-Net Smart Grids Plus project Grid-Friends, with support from the European Union's Horizon 2020 research and innovation programme.

REFERENCES

[1] S. V. Albrecht, J. W. Crandall, and S. Ramamoorthy. Belief and truth in hypothesised behaviours. *Artificial Intelligence*, 235:63–94, June 2016.

[2] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, 47(2/3):235–256, 2002.

[3] B. Banerjee and P. Stone. General game learning using knowledge transfer. In *International Joint Conference on Artificial Intelligence*, pages 672–677, Dec. 2007.

[4] S. Barrett and P. Stone. Cooperating with Unknown Teammates in Complex Domains: A Robot Soccer Case Study of Ad Hoc Teamwork. In *Proceedings of the 29th Conference on Artificial Intelligence*, pages 2010–2016, Austin, Texas, USA, Dec. 2014.

[5] J. Bednar, Y. Chen, T. X. Liu, and S. Page. Behavioral spillovers and cognitive load in multiple games: An experimental study. *Games and Economic Behavior*, 74(1):12–31, 2012.

[6] R. Bellman. A Markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684, 1957.

[7] D. Bloembergen, K. Tuyls, D. Hennes, and M. Kaisers. Evolutionary Dynamics of Multi-Agent Learning: A Survey. *Journal of Artificial Intelligence Research*, 53:659–697, 2015.

[8] G. Boutsioukis, I. Partalas, and I. Vlahavas. Transfer learning in multi-agent reinforcement learning domains. In *Recent Advances in Reinforcement Learning*, pages 249–260, Athens, Greece, 2011.

[9] M. Bowling and M. Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136(2):215–250, 2002.

[10] R. I. Brafman and M. Tennenholtz. R-MAX a general polynomial time algorithm for near-optimal reinforcement learning. *The Journal of Machine Learning Research*, 3:213–231, 2003.

[11] E. Brunskill and L. Li. PAC-inspired Option Discovery in Lifelong Reinforcement Learning. In *Proceedings of the 22nd Conference on Artificial Intelligence*, pages 1599–1610, 2014.

[12] L. Busoniu, R. Babuska, and B. De Schutter. A Comprehensive Survey of Multiagent Reinforcement Learning. *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, Mar. 2008.

[13] D. Chakraborty and P. Stone. Multiagent learning in the presence of memory-bounded agents. *Autonomous Agents and Multi-Agent Systems*, 28(2):182–213, Feb. 2013.

[14] V. Conitzer and T. Sandholm. 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, Sept. 2006.

[15] J. W. Crandall. Just add Pepper: extending learning algorithms for repeated matrix games to repeated markov games. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, pages 104–111, 2012.

[16] J. W. Crandall. Towards minimizing disappointment in repeated games. *Journal of Artificial Intelligence Research*, 49(1):111–142, Jan. 2014.

[17] J. W. Crandall. Robust Learning for Repeated Stochastic Games via Meta-Gaming. In *International Joint Conference on Artificial Intelligence*, pages 3416–3422, 2015.

[18] B. C. Da Silva, E. W. Basso, A. L. Bazzan, and P. M. Engel. Dealing with non-stationary environments using context detection. In *Proceedings of the 23rd International Conference on Machine Learnig*, pages 217–224, Pittsburgh, Pennsylvania, 2006.

[19] Y. M. De Hauwere, P. Vrancx, and A. Nowe. Learning multi-agent state space representations. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, pages 715–722, Toronto, Canada, 2010.

[20] M. Elidrisi, N. Johnson, M. Gini, and J. W. Crandall. Fast adaptive learning in repeated stochastic games by game abstraction. In *Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems*, pages 1141–1148, Paris, France, 2014.

[21] F. Fernández and M. Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the 5th International Conference on Autonomous Agents and Multiagent Systems*, pages 720–727, Hakodata, Hokkaido, Japan, May 2006. ACM.

[22] D. Fudenberg and J. Tirole. *Game Theory*. The MIT Press, Aug. 1991.

[23] P. Hernandez-Leal, E. Munoz de Cote, and L. E. Sucar. A framework for learning and planning against switching strategies in repeated games. *Connection Science*, 26(2):103–122, Mar. 2014.

[24] P. Hernandez-Leal, M. E. Taylor, B. Rosman, L. E. Sucar, and E. Munoz de Cote. Identifying and Tracking Switching, Non-stationary Opponents: a Bayesian Approach. In *Multiagent Interaction without Prior Coordination Workshop at AAAI*, Phoenix, AZ, USA, Feb. 2016.

[25] P. Hernandez-Leal, Y. Zhan, M. E. Taylor, L. E. Sucar, and E. Munoz de Cote. Efficiently detecting switches against non-stationary opponents. *Autonomous Agents and Multi-Agent Systems*, June 2016.

[26] J. Langford and T. Zhang. The epoch-greedy algorithm for multi-armed bandits with side information. In *Advances in Neural Information Processing Systems*, pages 817–824, 2008.

[27] A. Lazaric and M. Ghavamzadeh. Bayesian multi-task reinforcement learning. In *Proceedings of the 27th International Conference on Machine Learning*, Haifa, Israel, 2010.

[28] A. Lazaric, M. Restelli, and A. Bonarini. Transfer of samples in batch reinforcement learning. In *International Conference on Machine Learning*, pages 544–551, Helsinki, Finland, 2008. ACM.

[29] B. Rosman, M. Hawasly, and S. Ramamoorthy. Bayesian Policy Reuse. *Machine Learning*, 104(1):99–127, Feb. 2016.

[30] M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research*, 10:1633–1685, 2009.

[31] M. E. Taylor, P. Stone, and Y. Liu. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8:2125–2167, 2007.

[32] J. Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, UK, Apr. 1989.