Auton Agent Multi-Agent Syst
DOI 10.1007/s10458-016-9347-3

An exploration strategy for non-stationary opponents

Pablo Hernandez-Leal[1] · Yusen Zhan[3] · Matthew E. Taylor[3] · L. Enrique Sucar[2] · Enrique Munoz de Cote[2]

Abstract The success or failure of any learning algorithm is partially due to the exploration strategy it exerts. However, most exploration strategies assume that the environment is stationary and non-strategic. In this work we shed light on how to design exploration strategies in non-stationary and adversarial environments. Our proposed adversarial drift exploration (DE) is able to efficiently explore the state space while keeping track of regions of the environment that have changed. This proposed exploration is general enough to be applied in single agent non-stationary environments as well as in multiagent settings where the opponent changes its strategy in time. We use a two agent strategic interaction setting to test this new type of exploration, where the opponent switches between different behavioral patterns to emulate a non-deterministic, stochastic and adversarial environment. The agent's objective is to learn a model of the opponent's strategy to act optimally. Our contribution is twofold. First, we present DE as a strategy for switch detection. Second, we propose a new algorithm

Most of this work was performed while the first author was a graduate student at INAOE.
This paper extends the paper "Exploration strategies to detect strategy switches" presented at the Adaptive Learning Agents workshop [27].

✉ Pablo Hernandez-Leal
Pablo.Hernandez@cwi.nl

Yusen Zhan
yzhan@eecs.wsu.edu

Matthew E. Taylor
taylorm@eecs.wsu.edu

L. Enrique Sucar
esucar@inaoep.mx

Enrique Munoz de Cote
jemc@inaoep.mx

1 Centrum Wiskunde & Informatica (CWI), Science Park 123, Amsterdam, The Netherlands

2 Instituto Nacional de Astrofísica, Óptica y Electrónica, Luis Enrique Erro 1, Sta. María Tonantzintla, Puebla, México

3 Washington State University (WSU), Pullman, WA, USA

Published online: 13 October 2016

called R-MAX# for learning and planning against non-stationary opponent. To handle such opponents, R-MAX# reasons and acts in terms of two objectives: (1) to maximize utilities in the short term while learning and (2) eventually explore opponent behavioral changes. We provide theoretical results showing that R-MAX# is guaranteed to detect the opponent's switch and learn a new model in terms of finite sample complexity. R-MAX# makes efficient use of exploration experiences, which results in rapid adaptation and efficient DE, to deal with the non-stationary nature of the opponent. We show experimentally how using DE outperforms the state of the art algorithms that were explicitly designed for modeling opponents (in terms average rewards) in two complimentary domains.

Keywords Learning · Exploration · Non-stationary environments · Switching strategies · Repeated games

1 Introduction

Many learning algorithms perform poorly when interacting with non-stationary strategies because of the complexity related to handling changes in behavior, randomness and drifting. This is troublesome, since many applications demand an agent to interact with different acting strategies, and even to collaborate closely together with them.

Regardless of the task's nature and whether the agent is interacting with another agent, a human or is isolated in its environment, it is reasonable to expect that some conditions will change in time. These changes turn the environment into a non-stationary one and render many state of the art learning techniques futile. Examples abound and range from multiagent (competitive) settings like: charging vehicles in the smart grid [36], where behavioral patterns change across days; poker playing [5], where a common tactic is to change play style; negotiation tasks, where agents may change preferences or discount factors while interacting; and competitive-negotiation-coordination scenarios like the lemonade stand game tournament [52], where the winning strategy has always been the fastest to adapt to how other agents act. Cooperative domains include tutoring systems [39], where the user may change its behavior and the system needs to adapt; robot soccer teams [35], where agents may have different strategies to act that depend on the state of the game; and human-machine interaction scenarios like service robots aiding the elderly and autonomous car assistants where humans have different behaviors depending on humor and mental state. All these scenarios have one thing in common: agents must learn how their counterpart is acting and react quickly to changes in the opponent behavior.

Game theory provides the foundations for acting optimally under competitive and cooperative scenarios. However, its assumptions that opponents are fully rational and their strategies are stationary are too restrictive to be useful in many real interactions, like human interactions. This work relaxes some game theoretic and machine learning assumptions about the environment, and focuses on the concrete setting where an opponent switches from one stationary strategy to a different stationary strategy.

We start by showing how an agent facing a stationary opponent strategy in a repeated game can treat such an opponent as a stationary (Markovian) environment, so that any RL algorithm can learn the optimal strategy against such opponents. We then show that this is not the case for non-stationary opponent strategies, and argue that classic exploration strategies (e.g. ϵ-greedy or softmax) are the cause of such failure. First, classic exploration techniques tend to decrease exploration rates over time so that the learned (optimal) policy can be

applied. However, against non-stationary opponents it is well-known that exploration should not decrease so as to detect changes in the structure of the environment at all times [40]. Second, exploring "blindly" (random exploration) can be a dangerous endeavour because of two factors: (i) in diametrically opposed environments (e.g. zero sum games), any deviation from the optimal strategy is always worse, and more importantly (ii) some exploratory actions can cause the opponent to switch strategy which is not always a profitable option. Here we use recent algorithms that perform efficient exploration in terms of the number of suboptimal decisions made during learning [7] as stepping stone to derive a new exploration strategy against non-stationary opponent strategies. Our proposed adversarial drift exploration is able to *efficiently* explore the state space representation while being able to detect regions of the environment that have changed. As such, this proposed exploration can be readily applied in single agent non-stationary environments as well as in multiagent settings where the opponent changes its strategy. Our contribution is twofold. First, we present drift exploration (DE) as a strategy for switch detection and second, a new algorithm called R-MAX# (since it is sharp to changes) for learning and planning against non-stationary opponents. R-MAX# by making efficient use of exploration experiences deals with the multiagent non-stationary nature of the opponent behavior, which results in rapid adaptation and efficient DE. We provide theoretical results showing that R-MAX# is guaranteed to to detect the opponents' switches and learn the new model in terms of finite sample complexity.

The paper is organized as follows. Section 2 presents related work in exploration and non-stationary learning algorithms, Sect. 3 introduces basic background and the test scenarios used for this paper. Section 4 describes the DE and our proposed approach R-MAX#. Section 5 provides theoretical guarantees for switch detection and expected rewards for R-MAX#. Section 6 presents experiments in two different domains including the iterated prisoner's dilemma and a negotiation task. Section 7 provides the conclusions and ideas for future research.

2 Related work

The exploration-exploitation tradeoff has been studied in many different contexts, including: multi-armed bandits, reinforcement learning (both model-free and model-based) and multi-agent learning. In this section, we describe different approaches related to exploration and learning in non-stationary environments.

2.1 Multi-armed bandits and single-agent RL

In the context of multi-armed bandits [23], there are some works dealing with non-stationarity. In particular, two algorithms are based the upper confidence bounds (UCB) algorithm [1] to detect changes: one relies on the discounted UCB and the other one adopts a sliding window approach [21].

In single agent RL there are works that focus on non-stationary environments, one of those works are the hidden-mode Markov decision processes (HM-MDPs) [14]. In this model, the assumption is that the environment can be represented in a small number of *modes*. Each mode represents a stationary environment, which has different dynamics and optimal policies. It is assumed that at each time step there is only one active mode. The modes are hidden, which means they cannot be directly observed, and instead are estimated through past observations. HM-MDPs need an offline learning phase and they are computationally complex since they need to solve a POMDP [38] to obtain a policy to act. Another related approach is the reinforcement learning (RL) with context detection [15]. The algorithm's idea

is to learn several partial models and decide which one to use depending on the context of the environment. However, this approach does not provide any exploration for switch detection neither provides any theoretical guarantees.

2.2 Multiagent exploration

One simple approach to handle multiagent systems is to not explicitly model other agents, thus, even if there are indeed other agents in the world, they are not modeled as having goals, actions, etc. They are just considered part of the environment [46]. Notwithstanding, this presents problems when many learning agents are interacting in the environment and they explore at the same time, they may create some noise to the rest, called exploratory action noise [29]. HolmesParker et al. propose the coordinated learning exploratory action noise (CLEAN) rewards to cancel such noise. To achieve this, CLEAN assumes that agents have access to an accurate model of the reward function to compute a type of reward shaping [42] to promote coordination and scalability. Exploratory action noise will not appear in our setting for two reasons: (1) we assume only one opponent in the environment that is not a learning agent and (2) we explicitly model its behavior.

Exploration in multiagent systems has been used in specific domains such as economic systems [44] or foraging [37]. Recently, an application for interdomain routing used an exploration for adapting to changing environments [49]. Vrancx et al. refer to this exploration as *selective exploration* and it is based on computing a probability distribution of the past rewards and when the most recent one falls far away from the distribution a random action to explore is triggered.

2.3 Multiagent learning

In a different but related context there are some related works in the area of multiagent RL [8]. One work proposes to extend the Q-learning algorithm [50] to zero-sum stochastic games. The algorithm (coined as Minimax-Q) uses the minimax operator to take into account the opponent actions [32]. This allows the agent to converge to a fixed strategy that is guaranteed to be safe in that it does as well as possible against the worst possible opponent (one that tries to minimize the agent's rewards). However, this algorithm assumes the opponent has a diametrically opposed objective to the agent, and acts accordingly to this objective. When this assumption does not hold (which is reasonable in many scenarios), Minimax-Q does not converge to the best response.

The win or learn fast (WoLF) principle was introduced to make an algorithm that (1) converges to a stationary policy in multiagent systems and (2) obtains a best response against stationary policies (rational algorithm) [6]. The intuition of WoLF is to learn quickly when losing and cautiously when winning. One proposed algorithm that uses this principle was WoLF-IGA. The algorithm uses gradient ascent, so that with each interaction the player will update its strategy to increase its expected payoffs. WoLF-PHC is another algorithm that uses a variable learning rate. This one is based on Q-learning and performs hill-climbing in the space of mixed policies. Also, it has been proved theoretically to converge in self-play in a class of repeated matrix games. However, its policies have shown slow convergence in practice [47].

The Non-stationary Converging Policies algorithm [51] computes a best response to a restricted class of opponents, i.e., they are non-stationary with a limit (with decreasing changes). Thus, it is designed for slow changes, not drastic changes like our approach.

Recent approaches have tried to learn and adapt quickly to changes, one approach is the Fast Adaptive Learner (FAL) [18]. This algorithm focuses on fast learning in two-player repeated games. To predict the next action of the opponent it maintains a set of hypotheses according to the history of observations. To obtain a strategy against the opponent they use a modified version of the Godfather strategy.[1] However, the Godfather strategy is not a general strategy that can be used against any opponent and in any game. Also FAL shows an exponential growth in the number of hypotheses (in the size of the observation history) which may limit its use in larger domains. Recently, a version of FAL designed for stochastic games, FAL-SG [19], has been presented. In order to deal with this different setting, Elidrisi et al. propose to map the stochastic game into a repeated game and then use the original FAL approach. To transform a stochastic game into a repeated game, FAL-SG generates a meta-game matrix by means of clustering the opponent's actions. After obtaining this matrix the algorithm proceeds in the same way as FAL. Both FAL and FAL-SG use a modified Godfather strategy to act, which fails to promote a exploration for switch detection and this results in a longer time to detect these switches.

2.3.1 Model based approaches

Previous approaches used variations of the Q-learning algorithms, however, there are many algorithms which directly model the opponent.

Using a simple but well-known domain such as the iterated prisoner's dilemma, Carmel et al. propose a lookahead-based exploration strategy for model-based learning [9]. In this case, the opponent was modeled through a mixed model in a way to reflect uncertainty about the opponent. However, the approach does not analyze exploration in non-stationary environments (like strategy switches).

Exploring the environment efficiently in stationary domains has been widely studied by the computational learning theory community. R-max [7] is a well-known, model-based RL algorithm. The algorithm uses an MDP to model the environment which is initialized optimistically assuming all actions return the maximum possible reward, (r-max). With each experience of the form (s, a, s', r) R-max updates its model. The policy efficiently leads the agent to less known state-action pairs or exploits known ones with high utility. R-max promotes an efficient sample complexity of exploration [30]; R-max has theoretical guarantees for obtaining near-optimal expected rewards. However, R-max alone will not work when the environment is non-stationary (e.g. strategy switches). A recent approach based on R-max that takes into account possible changes in the dynamics of the environment is ζ-R-max [34], which does not assume that the visitation counts translate directly to model certainty. In contrast, they estimate this progress in terms of the loss over the training data used for model learning. The idea is to compute a ζ function which is based on the leave-one-out cross validation error. A limitation of this approach is the computational cost of computing ζ, since it depends on the number of states and actions at every iteration. Also, computing leave-one-out cross validation (extreme case of k-fold cross validation) is rarely adopted in large-scale applications because it is computationally expensive [10].

Another related approach is MDP4.5 [25] which is designed to learn against switching (non-stationary) opponents. MDP4.5 uses decision trees to learn a model of the opponent. The learned tree, along with information from the environment are transformed into a MDP

[1] *Godfather* [33] offers the opponent a situation where it can obtain a high reward. If the opponent does not accept the offer, Godfather forces the opponent to obtain a low reward.

[43] in order to obtain an acting policy. Since decision trees are known to be unstable for small datasets [16] this poses a problem for learning efficient models in a fast way.

Focusing on learning against classes of opponents is another related area. *Bounded memory opponents* are agents that use the opponent's $\mathcal{D}$ past actions to assess the way they are behaving. For these agents the opponent's history of play defines the *state* of the learning agent. In this context, a related model is the adversary induced MDP (AIM) [4], which uses the vector $\psi^{\mathcal{D}}$, a function of the past $\mathcal{D}$ actions of the agents. The learning agent, by just keeping track of $\psi^{\mathcal{D}}$ (its own past moves), can infer the policy of the bounded memory opponent. For this, the learning agent obtains the MDP that the opponent induces, and then it can compute an optimal policy π^*. These types of players can be thought of as a finite automata that take the $\mathcal{D}$ most recent actions of the opponent and use this history to compute their policy [41]. Since memory bounded opponents are a special case of opponents, different algorithms were specially developed to be used against these agents [11,13]. For example, the agent LoE-AIM [12] is designed to play against a memory bounded player (but does not know the exact memory length). Note that most of Chakraborty's work focuses on inferring the memory size used by the opponent, once this is defined, then one can learn optimally how to interact with the opponent. We instead assume we have the correct representation and are interested in opponents that change strategy in time (i.e. non-stationary strategies). Chakraborty's work cannot be used in this context. Therefore, our approaches are complementary, not competing.

Algorithm 1: MDP-CL [26]

Input: Size of the window of interactions w, threshold κ, number of rounds in the repeated game T.
Function: *compareModels()*, compare two MDPs
Function: *planWithModel()*, solves MDP to obtain a policy to act
Function: *playWithPolicy()*, play using the computed policy

```
1  j = 2; //initialize counter
2  model = π* = ∅ //Initialize opponent model and policy
3  for t = 1 to T do
4      if t == i · w, (i ≥ 1) and model == ∅ //*
5      [h]Learn exploration model then
6          Learn an exploration model with past w interactions
7          π* = planWithModel(model) //Compute a policy from the learned model
8      if t == j · w, where (j ≥ 2) //*
9      [h]Learn comparison model then
10         Learn another model′ with past interactions
11         d = compareModels(model, model′) //Compare models
12         if d > κ //*
13         [h]Opponent strategy has changed? then
14             π* = model = ∅, j = 2 //Reset models and restart exploration
15         else
16             j = j + 1
17     if π* == ∅ then
18         Play with random actions //No model, explore randomly
19     else
20         playWithPolicy(π*) //Use learned policy
```

Another approach that uses MDPs to represent opponent strategies is the MDP-CL approach [26]. We introduced MDP-CL in previous work to act against non-stationary opponents (see Algorithm 1). It starts with an exploratory phase with random actions for

a determined number of rounds (lines 14–15), after which it computes a model of the opponent in the form of an MDP (lines 4–6). Then, it begins to use the optimal policy, π^*, (lines 16–17) and at the same time it starts to learn another model. After some rounds (defined by a parameter w) the approach compares them to evaluate their similarity (lines 7–9). If the *distance* between models is greater than a given threshold, κ, (line 10) the opponent has changed strategy and the modeling agent must restart the learning phase, resetting both models and starting from scratch with a random exploratory strategy (line 11). Otherwise, it means that the opponent has not switched strategies and the same policy is used (line 13). However, MDP-CL lacks a DE scheme, which results in the failure of detecting some switches in the opponent's behavior, therefore resulting in a suboptimal policy and performance.

In contrast to these previous approaches, here we propose a general DE that can be used jointly with switch detection algorithms. This exploration detects switches that otherwise would have passed unnoticed. Next, we propose an efficient approach which implicitly handles DE called R-max# and provide theoretical guarantees. In the experimental section we compared our proposals against MDP4.5 [25], MDP-CL [26], R-max [7], FAL [18] and WOLF-PHC [6].

3 Background

In this section, we will introduce the necessary background for presenting the main contribution: R-MAX#. First, we start by introducing some test scenarios to use as examples throughout the paper.

Our setting consists of two players, our learning agent A and the opponent B, that face each other and repeatedly play a *bimatrix game*. A bimatrix game is a two player simultaneous-move game defined by the tuple $\langle \mathcal{A}, \mathcal{B}, R_A, R_B \rangle$, where $\mathcal{A}$ and $\mathcal{B}$ are the set of possible actions for player A and B, respectively. R_i is the reward matrix of size $|\mathcal{A}| \times |\mathcal{B}|$ for each agent $i \in \{A, B\}$, where the payoff to the ith agent for the joint action $(a, b) \in \mathcal{A} \times \mathcal{B}$ is given by the entry $R_i(a, b)$, $\forall (a, b) \in \mathcal{A} \times \mathcal{B}, \forall i \in \{\text{A,B}\}$. A *stage game* is a single bimatrix game and a series of rounds of the same stage game form a *repeated game*.

The opponent has a set of different strategies to use and can change from one to another during the interaction. A *strategy* specifies a method for choosing an action. One kind of strategy is to select a single action and play it, this is a *pure* strategy. In general, a *mixed* strategy specifies a probability distribution over actions. A *best response* for an agent is the strategy (or strategies) that produce the most favorable outcome for a player, taking other players' strategies as given. The objective of our learning agent is to maximize its cumulative rewards obtained during the repeated interaction.

3.1 Test scenarios

The prisoner's dilemma is a game where the interactions can be modeled as a symmetric two-player game defined by the payoff matrix in Table 1 using values $d = 4$, $c = 3$, $p = 1$, $s = 0$,

Table 1 A bimatrix game representing the prisoner's dilemma, two agents can chose between two actions cooperate (C) and defect (D)

	C	D
C	3, 3	0, 4
D	4, 0	1, 1

where the following two conditions must hold $d > c > p > s$ and $2c > d + s$. When both players cooperate they both obtain the reward c. If both defect, they get a punishment reward p. If a player chooses to cooperate (C) with someone who defects (D) the cooperating player receives the sucker's payoff s, whereas the defecting player gains the temptation to defect, d. The iterated version (iPD) has been subject to many studies, including human trials. We used the iPD to perform experiments because there are many handcrafted strategies that fare well in this domain and we want to test how our algorithms fare against these. A common strategy which won a tournament of the iPD is called Tit-for-Tat (TFT) [2]; it starts by cooperating, then does whatever the opponent did in the previous round (it will cooperate if the opponent cooperated, and will defect if the opponent defected). Another very successful strategy is called Pavlov and cooperates if both players coordinated with the same action and defects whenever they did not. Bully [33] is another well-known strategy involving always behaving as a defecting player. It should be noticed that these previous strategies can be defined only by the current state and do not depend on the time index; they are *stationary* strategies. In contrast, when a strategy needs to be indexed by round it is a *non-stationary* strategy.

The second domain used in the experiments is automated negotiation which has been a subject of active research in artificial intelligence. We focus on the principal bargaining protocol, i.e, the discrete time finite-horizon alternating offers bargaining protocol [45]. Alternating-offers bargaining with deadlines is essentially a finite horizon extensive form game with infinite actions and perfect information [31]. It consists of two players, a buyer and a seller. The seller wants to sell a single indivisible item to the buyer for a price. The buyer and the seller can act at times $t \in \mathbb{N}$. Their offers alternate each other, trying to agree on a price. Their possible actions are: *offer(x)* with $x \in \mathbb{R}$ representing the proposed price for the item; *exit*, which indicates that negotiation fails; and *accept* which indicates that buyer and seller have reached and agreement. At time $t = 0$, $accept$ is not allowed, at $t > 0$ if any of the players $accepts$ the game finishes and the outcome is (x, t), where x is the value *offer(x)* at time $t - 1$. If one of them plays $exit$ the bargaining stops and the outcome is 0 for both of them. Otherwise the bargaining continues to the next time point. Each agent has a utility function U_i that depends on three parameters of agent i: reservation price, RP_i (the maximum/minimum amount a player is willing to accept), discount factor, γ_i, and deadline, T_i (agents prefer an earlier agreement, and after the deadline they *exit*). In a complete information setting, (where the parameters are known by players) backward induction reasoning can be used to compute the one and only subgame perfect equilibrium, which is achieved at time $t = 1$. However, complete information is not always available as well as players can deviate from perfect strategies and switch among several of those, like we will present in the experimental section.

3.2 Finding optimal policies against stationary opponents

In the domains presented previously, if an agent knew a model of the opponent it could build a policy, i.e. the best action in each step of the interaction, to maximize the sum of rewards. Machine learning provides algorithms to learn an optimal policy by accumulating experience using the RL framework. In RL, an agent's objective is to learn an optimal policy in stochastic environments, in terms of maximizing its expected long-term reward in an initially unknown environment that is modeled as a MDP [43]. An MDP is defined by $\langle S, \mathcal{A}, \mathcal{T}, R \rangle$, where S is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{T}$ is the transition function and R is the reward function. A *policy* is a function $\pi(s)$ that specifies an appropriate action a for each state s.

The interaction of a learning agent with a *stationary* opponent can be modeled as an MDP. This occurs since the interaction between agents generates Markovian observations which

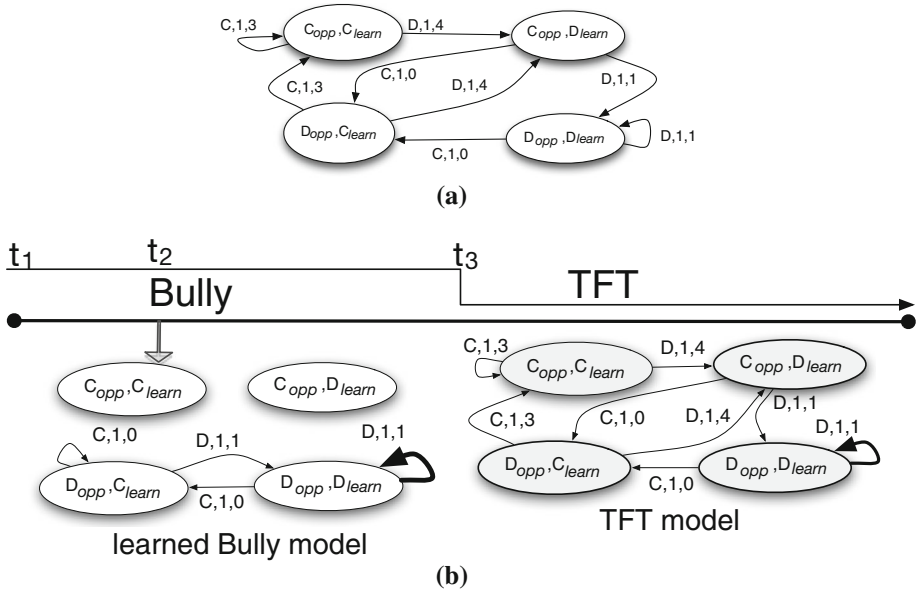


Fig. 1 **a** An MDP that represents the TFT strategy in the iPD. States are ovals, in this case the state space is formed by the last action (C and D) of the learning agent (*learn*) and the opponent (*opp*) each *arrow* has a corresponding triplet: learning agent action, transition probability and reward to the next state. **b** An example of a learning agent against a Bully–TFT switching opponent. The models represent two MDPs: opponent starts with Bully (at time t_1) and switches to TFT (at time t_3)

can be used to learn the opponent's strategy by inducing a MDP [4]. Banerjee et al. [4] propose the AIM model, which uses as states a function of the past actions of the learning agent. The learning agent can explore the opponent strategy by performing different actions. The state space is formed by attributes that can describe the opponent strategy. Rewards are obtained from the game matrix. After some rounds of interactions the information from the past interactions can be used to learn a MDP describing the opponent's strategy. In this case, the learning agent perceives a stationary environment whose dynamics can be learned. The learned MDP that represents the opponent strategy is defined by $\langle S, \mathcal{A}, \mathcal{T}, R\rangle$ where: $S := \times_{O_i \in O} O_i$, i.e. each state is formed by the cross product of the set of attributes that represent the opponent strategy. $\mathcal{A}$, are the actions of the learning agent. $\mathcal{T} : S \times \mathcal{A} \rightarrow S$, is learned using counts $\mathcal{T}(s, a, s') = \frac{n(s,a,s')}{n(s,a)}$ where $n(s, a, s')$ is the number of times the agent was in state s, used action a and arrived at state s', $n(s, a)$ is defined as the number of times the agent was in state s and used action a. R, is learned in a similar manner $R(s, a) = \frac{\sum r(s,a)}{n(s,a)}$ where $\sum r(s, a)$ is the cumulative reward obtained by the agent when being in state s and performing action a. The set of attributes O used to construct the states is assumed to be given by an expert and we assume the attributes are able to learn the correct opponent model. For example, assume an opponent playing the TFT strategy in the iPD, also assume our learning agent is using as attributes the last action of both players. A learned MDP representing the TFT opponent strategy is depicted in Fig. 1a. Each state is formed by the last play of both agents, C for cooperate, D for defect, subindices *opp* and *learn* refer to the opponent and the learning agent respectively (other attributes can be used for different domains), with arrows indicating the triplet: action, transition probability and immediate reward for the learning

agent. In any case, solving an MDP that represents the opponent stationary strategy dictates a policy which prescribes how to act against that opponent.

The next section presents DE, first as a general exploration that can be added to other algorithms and then on the R-MAX# algorithm.

4 Drift exploration

Exploration in non-stationary environments has a special characteristic that is not present in stationary domains. As explained before, if the opponent plays a stationary strategy, the learning agent perceives a stationary environment (an MDP) whose dynamics can be learned. However, if the former plays a stationary strategy ($strat_1$) that induces an MDP (MDP_1) and then switches to $strat_2$ and induces MDP_2, if $strat_1 \neq strat_2$, then $MDP_1 \neq MDP_2$ and the learned policy is probably not optimal anymore. Against this background, we propose a *drift exploration*, which tackles this problem.

In order to motivate DE, take the example depicted in Fig. 1b, where the learning agent faces a switching opponent in the iterated prisoner's dilemma. Here, at time t_1 the opponent starts with a strategy that defects all the time, i.e., Bully. The learning agent using counts can recreate the underlying MDP that represents the opponent's strategy (learned Bully model) by trying out all actions in all states (exploring). At some time, (t_2 in the figure) it can solve for the optimal policy against this opponent (because it has learned a correct model), which is to defect in the iPD, which will produce a sequence of visits to state D_{opp}, D_{learn}. Now, at some time t_3 the opponent switches its selfish Bully strategy to a fair TFT strategy. But because the transition $((D_{opp}, D_{learn}), D) = D_{opp}, D_{learn}$ in both MDPs, the switch in strategy (Bully $\rightarrow$ TFT) will not be perceived by the learning agent. Thus, the learning agent will not be using the optimal strategy against the opponent. This effect is known as *shadowing*[2] [20] and can only be avoided by continuously checking far visited states.

Drift exploration deals with such shadowing explicitly. In what follows we will present how DE is used for switch detection. We use MDP-CL as an example RL algorithm that can be used with this exploration to detect switches but many other RL algorithms could be used as well. We then we propose a new algorithm called R-MAX# (since it is sharp to changes) for learning and planning against non-stationary opponents.

4.1 General drift exploration

One problem with non-stationary environments is that opponent strategies may share similarities in their induced MDP (specifically between transition functions) when they share the same state and action spaces. This happens when the agent's optimal policy produces the same ergodic set of states, for two or more opponent strategies. It turns out this is not uncommon; for example the optimal strategy against Bully produces the sole state D_{opp}, D_{learn}, however, this part of the MDP is shared with TFT. Therefore, if the agent is playing against Bully, and is stuck in that ergodic set, a change of the opponent strategy to (say) TFT will pass unnoticed, resulting in a suboptimal policy and performance. The solution to this is to explore even when an optimal policy has been learned. Exploration schemes like ϵ-greedy or softmax (e.g. a Boltzmann distribution), can be used for such purpose and will work as DE with the added cost of not efficiently exploring the state space. We present this general

[2] A related behavior called *observationally equivalent models* has been reported by Doshi et al. [17].

version of DE into the MDP-CL framework [26] which yields the MDP-CL(DE) approach, tested experimentally in Sect. 6.3.

Against this background, now we propose an approach for drift exploration that efficiently explores the state space looking for changes in the transition function and demonstrate this with a new algorithm called R-MAX#.

4.2 Efficient drift exploration

Efficient exploration strategies should take into account what parts of the environment remain uncertain. R-MAX is an example of such type of exploration. In this section we present R-MAX#, an algorithm inspired by R-MAX but designed for strategic interactions against non-stationary switching opponents. To handle such opponents, R-MAX# reasons and acts in terms of two objectives: (1) to maximize utilities in the short term while learning and (2) eventually explore opponent behavioral changes.

4.2.1 R-MAX#

The basic idea of R-MAX# is to enforce revisiting long gone state-action pairs. These pairs are those that (1) are considered known and (2) have not been updated in τ rounds, at which point the algorithm resets its reward value to $rmax$ in order to promote exploration of that pair which implicitly rechecks to determine if the opponent model has changed.

R-MAX# receives as parameters $(m, rmax, \tau)$ where m and $rmax$ are used in the same way as R-MAX, and τ is a threshold that defines when to reset a state-action pair. R-MAX# starts by initializing the counters $n(s, a) = n(s, a, a') = r(s, a) = 0$, rewards to $rmax$ and transitions to a fictitious state s_0 (like R-MAX) and set of known pairs $\mathcal{K} = \emptyset$ (lines 1–4). Then, for each round the algorithm checks for each state-action pair (s, a) that is labeled as known ($\in \mathcal{K}$) how many rounds have passed since the last update (lines 8–9), if this number is greater than the threshold τ then the reward for that pair is set to $rmax$, the counters $n(s, a)$, $n(s, a, s')$ and the transition function $\mathcal{T}(s, a, s')$ are reset and a new policy is computed (lines 10–14). Then, the algorithm behaves as R-MAX (lines 15–24). The pseudocode of R-MAX# is presented in Algorithm 2.

4.3 Running example

Let us take the example in Fig. 2 to show how R-MAX# will interact against an unknown switching opponent. The opponent starts with a Bully strategy (t_1). After learning the model, R-MAX# knows that the best response against such strategy is to defect (t_2) and the interaction will be a cycle of defects. At time (t_3) the opponent changes from Bully to TFT, and because some state-action pairs have not been updated for several rounds (more than the threshold τ), R-MAX# resets the rewards and transitions for reaching such states, at which point a new policy is computed. This policy will encourage the agent to re-visit far visited states (i.e. states not visited recently). Now, R-MAX# will update its model as shown in the transition model in Fig. 2b (note the thick transitions which are different from the Bully model). After enough rounds (t_4), the complete TFT model would be learned and an optimal policy against it is computed.

Algorithm 2: R-max# algorithm

Input: States S, actions $\mathcal{A}$, m value, $rmax$ value, threshold τ

$\forall(s,a,s')\ r(s,a) = n(s,a) = n(s,a,s') = 0$
$\forall(s,a,s')\ \mathcal{T}(s,a,s') = s_0$
$\forall(s,a)\ R(s,a) = rmax$
$\mathcal{K} = \emptyset$
$\pi = Solve(S, \mathcal{A}, \mathcal{T}, R)$ //initial policy
for *t:1, ..., T* **do**
  Observe state s, execute action from policy $\pi(s)$
  for *each* (s,a) **do**
    if $(s,a) \in \mathcal{K}$ and $t - lastUpdate(s,a) > \tau$ **then**
      $R(s,a) = rmax$ //reset reward to rmax
      $n(s,a) = 0$
      $\forall s'\ n(s,a,s') = 0$ //reset counters
      $\forall s'\ \mathcal{T}(s,a,s') = s_0$ //reset transitions
      $\pi = Solve(S, \mathcal{A}, \mathcal{T}, R)$ //Solve MDP and get new policy.
  if $n(s,a) < m$ **then**
    Increment counters for $n(s,a)$ and $n(s,a,s')$
    Update reward $r(s,a)$
    if *n(s,a)=m* **then**
      //pair is considered known
      $\mathcal{K} = \mathcal{K} \cup (s,a)$
      $lastUpdate(s,a) = t$
      $R(s,a) = r(s,a)/m$
      for $s'' \in S$ **do**
        $\mathcal{T}(s,a,s'') = n(s,a,s'')/m$
      $\pi = Solve(S, A, \mathcal{T}, R)$

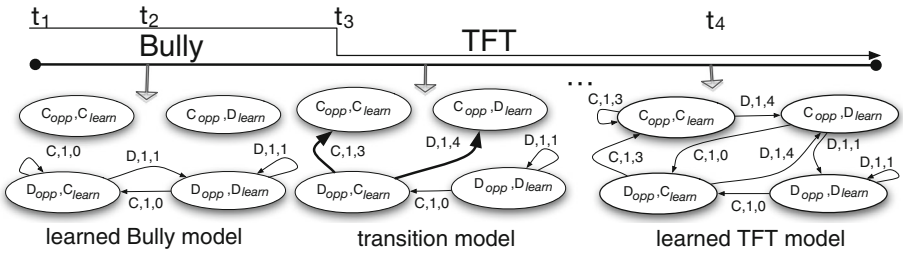


Fig. 2 An example of the learned models of R-MAX# against a Bully–TFT switching opponent. The models represent three learned MDPs: at the extremes opponent starts with Bully (at time t_1) and switches to TFT (t_4) and in the middle a transition model, learned after the switch (t_3) Bully → TFT

4.4 Practical considerations of R-MAX#

In Sect. 5 we present theoretical results about R-MAX#. Here we provide some insights about how to set the τ parameter. In contrast to stationary opponents, when acting against non-stationary opponents we need to perform drift exploration constantly. However, knowing when to explore is a more difficult question, especially if we know nothing about possible switching behavior (switching periodicity). However, even in this case we can still provide guidelines for setting τ: (1) τ should be large enough to learn a sufficiently good opponent model (or it could be a partial model when the state space is large). In this situation, the

algorithm would learn a partial model and optimize against that. (2) τ should be small enough to enable exploration of the state space. An extremely large value for τ will decrease the exploration for longer periods of time and will take longer to detect opponent switches. (3) If expected switching times can be inferred or learned then τ can be set accordingly to a value that is related to those timings. This is explained by the fact that after the opponent switches strategies, the optimal action is to re-explore at that time. For experimental results that support these guidelines please refer to Sects. 6.4 and 6.5.

Algorithm 3: R-MAX#CL algorithm

```
   Input: w window, κ threshold, m value, rmax value, threshold τ
   Function: compareModels(), compare two MDPs.
   Function: R-MAX#(), call R-max# algorithm.
1  Initialize as R-MAX#
2  model == ∅
3  for t : 1, ..., T do
4      R-MAX#(m, rmax, τ)
5      if t == i · w, (i ≥ 1) and model == ∅ then
6          Learn a model with past w interactions
7      if t == j · w, (j ≥ 2) then
8          Learn comparison model' with past interactions
9          d= compareModels(model,model')
10         if d > κ //*
11         [h]Strategy switch? then
12             Reset models, j = 2
13         else
14             j = j + 1
```

4.5 Efficient drift exploration with switch detection

As final approach we propose to use the efficient DE of R-MAX# together with a framework for detecting switches. In fact, these two approaches tackle the same problem in different ways and therefore should probably complement each other at the expense of some extra exploration. We call this algorithm R-MAX# CL, which combines MDP-CL [26] synchronous updates with the asynchronous rapid adaptation of R-MAX#.

The approach of R-MAX# CL is presented in Algorithm 3. It starts as R-MAX#, efficiently exploring the state space (line 4). The switch detection process of MDP-CL is performed concurrently, so that it learns an initial model with w interactions (lines 5–6). Then R-MAX# CL starts learning the comparison model (lines 7–9) and if a switch is detected the current model and policy are reset and R-MAX# restarts (line 11). Naturally, such combination of explorations will turn out to be profitable in some settings and in some others it is better to use only one.

Next section provides theoretical results for R-MAX# in terms of switch detection and expected rewards against non-stationary opponents.

5 Sample complexity of exploration for R-MAX#

In this section, we study the sample complexity of exploration for R-MAX# algorithm. Before presenting our analysis, we first revisit our assumptions.

1. Complete (self) information: the agent knows its states, actions and rewards received.

2. Approximation condition (from [30]): states that the policy derived by R-MAX is near-optimal in the MDP (Definition 2, see below).
3. The opponent will use a stationary strategy for some number of steps.
4. All state-action pairs will be marked known between some number of steps.

Given the aforementioned assumptions, we show that R-MAX# will eventually relearn a new model for the MDP after the opponent switches and compute a near-optimal policy.

We first need some definitions and notations, as presented in [30], to formalize the proofs. Firstly, an L-round MDP $= \langle S, \mathcal{A}, \mathcal{T}, r \rangle$ is an MDP with a set of decision rounds $\{0, 1, 2, \ldots, L-1\}$, L is either finite or infinite. In each round both agents choose actions concurrently. A deterministic T-step policy π is a T-step sequences of decision rules of the form $\{\pi(s_0), \pi(s_1), \ldots, \pi(s_{T-1})\}$, where $s_i \in S$. To measure the performance of a T-step policy in the L-round MDP, t-value is used.

Definition 1 (*t-value*, [30]) Let M be an L-round MPD and π be a T-step policy for M. For a time $t < T$, the t-value $U_{\pi,t,M}(s)$ of π at state s is defined as

$$U_{\pi,t,M}(s) = \frac{1}{T}\mathbb{E}_{(s_t,a_t,\ldots,s_{T-1},a_{T-1})\sim Pr(\cdot|\pi,M,s_t=s)}\left[\sum_{i=t}^{T-1} r(s_i,a_i)\right],$$

where the T-path $(s_t, a_t, \ldots, s_{T-1}, a_{T-1})$ is from time t up until time T starting at s and following the sequence $\{\pi(s_t), \ldots, \pi(s_{T-1})\}$.

The optimal t-value at state s is

$$U^*_{t,M}(s) = \sup_{\pi\in\Pi} U_{\pi,t,M}(s),$$

where Π is the set of all T-step policies for MDP M. Finally, we define the Approximation Condition (assumption 2).

Definition 2 Let $\mathcal{K}$ be a set of known states and MDP $\hat{M}_\mathcal{K}$ be an estimation of the true MDP $M_\mathcal{K}$ with a set of states $\mathcal{K}$. The optimal policy $\hat{\pi}$ derived from $\hat{M}_\mathcal{K}$ is ϵ-optimal if, for all states s and times $t \leq T$,

$$U_{\hat{\pi},t,M_\mathcal{K}}(s) \geq U^*_{t,M_\mathcal{K}}(s) - \epsilon,$$

where $\epsilon > 0$ defines the accuracy.

Assumption 2 assure the the policy $\hat{\pi}$ derived by R-MAX from $\hat{M}_\mathcal{K}$ is near-optimal in the true MDP $M_\mathcal{K}$. For R-MAX#, we have the following main theorem:

Theorem 1 *Let* $\tau = \frac{2m|S||A|T}{\epsilon}\log\frac{|S||A|}{\delta}$ *and M' be the new L-round MDP after the opponent switches its strategy. For any* $\delta > 0$ *and* $\epsilon > 0$, *the R-MAX# algorithm guarantees an expected return of* $U^*_{M'}(c_t) - 2\epsilon$ *within* $O(\frac{m|S|^2|A|T^3}{\epsilon^3}\log^2\frac{|S||A|}{\delta})$ *timesteps with probability greater than* $1-\delta$, *given timesteps* $t \leq L$.

The parameter $\delta > 0$ is used to define the confidence $1-\delta$. We assume all $\delta > 0$ and we can always choose a proper δ to ensure $1 - c\cdot\delta$ is less than 1 and greater than 0, where $c > 0$ is a constant. The proof of Theorem 1 will be provided after we introduce some lemmas. Now we just give a sketch of the proof. R-MAX# is more general than R-MAX since it is capable of reseting the reward estimations of state-action pairs. However, the basic result of R-MAX# is derived from R-MAX. The proof relies on applying the R-MAX sample complexity theorem

to R-MAX# as a basic solver. With proper assumptions, R-MAX# can be viewed as R-MAX with periods, i.e., the running timesteps of R-MAX# is separated into periods. In each period, R-MAX# behaves as the classic R-MAX so that R-MAX# can learn the new state-action pairs by the R-MAX algorithm after the opponent switches its policy.

Theorem 2 (From [30]) *Let $M = \langle S, A, \mathcal{T}, r\rangle$ be a L-round MDP. If c is an L-path sampled from $Pr(\cdot|R\text{-MAX}, M, s_0)$ and the assumption 1 and 2 hold, then, with probability greater than $1-\delta$, the R-max algorithm guarantees an expected return of $U^*(c_t) - 2\epsilon$ within $\beta_1 \frac{m|S||A|T}{\epsilon} \log \beta_2 \frac{|S||A|}{\delta}$ timesteps $t \leq L$, where $\beta_1, \beta_2 > 0$ are constant.*

The proof of Theorem 2 is given in Lemma 8.5.2 in [30]. To simplify the notation, let $C = \beta_1 \frac{m|S||A|T}{\epsilon} \log \beta_2 \frac{|S||A|}{\delta}$.

Lemma 1 *After C steps, each state-action pair (s, a) is visited m times with probability greater than $1 - \delta/2$.*

Proof outline This is an alternative interpretation of Theorem 2 due to Hoeffding's bound, using $\delta/2$ to replace δ. See the details in the Lemma 8.5.2 from [30]. □

Lemma 2 *With properly chosen τ, the R-MAX# algorithm resets and visits each state-action pair m times with probability greater than $1 - \delta$.*

Proof Suppose the R-MAX# algorithm already learned a model. Lemma 1 states that within C steps, each state-action pair (s, a) is visited m times with probability greater than $1 - \delta/2$. That is, we learn a model and all state-action pairs are marked as known. Remember that τ measures the difference between the current time step and the time step at which each action-state pair is visited m times again. To make sure R-MAX# does not reset a state-action pair before all state-action pairs are visited with probability greater than $1 - \delta/2$, τ is at least C. Hoeffding's bound does not predict the order of the mth visit for each state-action pair. The worst situation is that all state-action pairs are marked known near $t = C$ (see Fig. 3 for details). According to the Lemma 1, we need an extra interval C to make sure the all the state-pairs are visited m times after reset with probability greater than $1 - \delta/2$. In all, we need to set $\tau = 2C$. Note that according to the assumption 4, all state-action pairs will be learned between $t = nC$ and $t = (n + 1)C$ after resetting. Then, R-MAX# restarts the learning process.

To simplify the proof, we introduce the concept of a *cycle* to help us to analysis the algorithm.

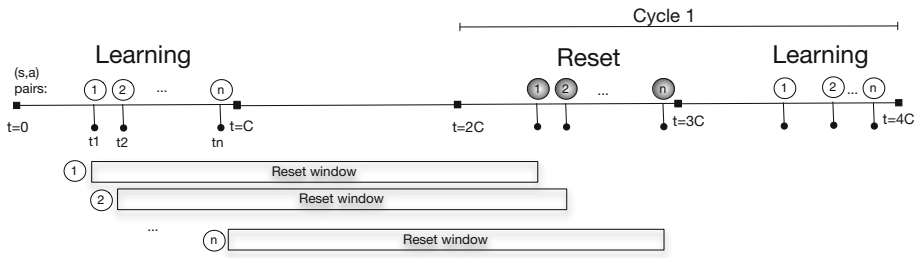


Fig. 3 An illustration for the running behavior of R-MAX#. The length of the reset window is $\tau = 2C$ in R-MAX#. In the learning stage, all state-action pairs may be marked known before $t = C$ with high probability. After resetting, we assume that all state-action pairs will be marked known in $[3C, 4C]$ with high probability

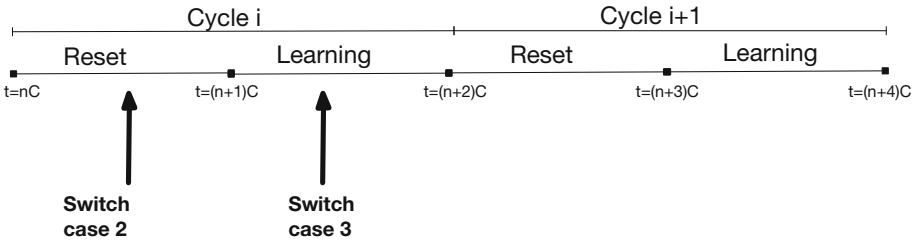


Fig. 4 Suppose an opponent switches in cycle i, there are two possible switch points. The R-MAX# will learn the new state-action pairs within two cycles

Definition 3 A cycle occurs when all reward estimations for each each state-action pair (s, a) are reset and then marked as known.

Intuitively, a cycle is the process where R-MAX# forgets the old MDP and learns the new one. According to the Lemma 2, $2C$ steps are sufficient to reset and visit each state-action pair (s, a) m times, with probability at least $1 - \delta$. Thus, we set the length of one cycle as $2C$. A cycle is a τ window so that we leave enough timesteps for R-MAX# reset and learn each state-action pair.

Lemma 3 *Let* $\tau = 2C$ *and* M' *be the new L-round MDP after the opponent switches its strategy the R-MAX# algorithm guarantees an expected return of* $U^*_{M'}(c_t) - 2\epsilon$ *within* $O\left(\frac{m|S||A|T}{\epsilon} \log \frac{|S||A|}{\delta}\right)$ *timesteps with probability greater than* $1 - 3\delta$.

Proof Lemma 2 states that if $\tau = 2C$, each state-action pair (s, a) are reset within $2C$ timesteps with probability greater than $1-\delta$, since $(1-\delta/2)(1-\delta/2) = 1-\delta+\delta^2/2 > 1-\delta$. If an opponent switched its strategy at any timestep in cycle i (see Fig. 4 for details), there are three cases: (1) R-MAX# does not reset the corresponding state-action pairs (s, a), since they are not considered known ($\notin \mathcal{K}$); (2) R-MAX# resets the corresponding state-action pairs (s, a) reward estimations but does not learn new ones; (3) R-MAX# has already reset and learned the state-action pairs (s, a).

Case 1 is safe for R-MAX# since it learns the new state-action pairs, with probability greater than $1 - \delta$. For cases 2 and 3, the worst is case 3 since R-MAX# is not able to learn new state-actions pairs within cycle i, whereas R-MAX# may have chance to learn new state-actions in case 2 (in the learning phase in the same cycle i). The assumption 3 states that the opponent adopts a stationary strategy at least $4C$ steps, which is exactly 2 cycles between two switch points. Although R-MAX# cannot learn new state-action pairs within cycle i when case 3 happens, it can learn them in cycle $i + 1$ by Lemma 2.

In all, R-MAX# will eventually learn the new state-action pairs in either cycle i or cycle $i + 1$ with probability greater than $1 - 2\delta$ since $(1 - \delta)(1 - \delta) = 1 - 2\delta + \delta^2 > 1 - 2\delta$. That is the R-MAX# requires 2 cycles or $4C$ to learn a new model to fit the new opponent policy. Apply the chain rules in probability theory and Theorem 2, the R-MAX# algorithm guarantees an expected return of $U^*(c_t) - 2\epsilon$ within $O\left(\frac{m|S||A|T}{\epsilon} \log \frac{|S||A|}{\delta}\right)$ (using big-O notation to eliminate all constants) timesteps with probability greater than $(1 - 2\delta)(1 - \delta) = 1 - 3\delta + 2\delta^2 > 1 - 3\delta$.

Note that we do not propose the value of m. How should the value of m be bounded? Kakade shows that $m = O\left(\frac{|S|T^2}{\epsilon^2} \log \frac{|S||A|}{\delta}\right)$ is sufficient, given error ϵ and confidence δ (Lemma 8.5.6 in [30]). With this result, we the have following proof for Theorem 1:

Proof of Theorem 1 Combining Lemma 3 and $m = O\left(\frac{|S|T^2}{\epsilon^2}\log\frac{|S||A|}{\delta}\right)$, Theorem 1 follows with setting $\delta \leftarrow \frac{\delta}{4}$.

The proofs heavily rely on the assumptions we state at the beginning of this section, however maybe it is too strong to capture the practical performance of R-MAX#. In particular, assumption 4 may not hold in some domains, nonetheless it provides a theoretical way to understand R-MAX#. Also, the theoretical result gives some bounds on the parameters, e.g., $\tau = 2C$ and $C = \beta_1 \frac{m|S||A|T}{\epsilon}\log\beta_2\frac{|S||A|}{\delta}$.

6 Experiments

In this section we present experiments with our proposed algorithms that use DE: MDP-CL(DE) (Sect. 4.1), R-MAX# (Sect. 4.2) and R-MAX# CL (Sect. 4.5). Comparisons are performed with state of the art approaches: two of our previous approaches MDP4.5 [25] and MDP-CL [26]; R-max [7] used as baseline; FAL [18] since it is a fast learning algorithm in repeated games, WOLF-PHC [6][3] since it can learn non-stationary environments; and the omniscient (perfect) agent that best responds immediately to switches. Results are compared in terms of average utility over the repeated game. Experiments were performed on two different scenarios: the iterated prisoner's dilemma and a negotiation task. In all cases the opponents were non-stationary in the sense that they used different stationary strategies for acting in a single repeated interaction. The experiments are presented as follows:

- We begin with a general result, showing that approaches with DE obtain better results than approaches without exploration against non-stationary opponents in both domains (Sect. 6.2). Then, we revise each algorithm in more detail.
- We compare how drift exploration in MDP-CL(DE) improves the results over MDP-CL (Sect. 6.3).
- Later, we present R-MAX# against deterministic switching opponents (Sect. 6.4) and show how the parameters affect its behavior.
- Finally, we present how R-MAX# CL combines advantages of the previous algorithms and with the appropriate parameters obtains the best results (Sect. 6.5).

Results in bold denote the best scores in the tables. Statistical significance is denoted with * and † symbols and we used Wilcoxon rank-sum test with a significance level $\alpha = 0.05$.

6.1 Experimental domains

First we present the experimental domains and its characteristics.

- Iterated Prisoner's Dilemma. Values for the game are presented in Table 1. We used the three most successful human-crafted strategies that the literature has proposed for this game: TFT, Pavlov [2] and Bully [33] as opponent strategies. These three strategies have different behaviors in the iPD and the optimal policy differs across them.[4] We emulate two different scenarios.
  - Deterministic switching opponent, the opponent switches strategies deterministically every 100 rounds.

[3] To ensure a DE there was a constant ϵ-greedy$= 0.2$ exploration and no decay in the learning rate (α).

[4] Optimal policies are always cooperate, Pavlov and always defect, against opponents TFT, Pavlov and Bully, respectively.

- Probabilistic switching opponent. Each repeated game consists of 1000 rounds and at every round the opponent either continues using the current strategy with probability $1-\eta$ or with probability η switches to a new strategy drawn randomly from the set of strategies mentioned before. We use switching probabilities $\eta = 0.02, 0.015, 0.01$ which translates to 10–20 strategy switches in expectation for one repeated game.

– Bilateral Negotiation. Our second domain is a more complex scenario. It consists of two players, a buyer and a seller. Their offers alternate each other, trying to agree on a price. Their possible actions are *offer(x)* with $x \in \mathbb{N}$, *exit* and *accept*. If any of the players *accepts* the game finishes with rewards for the players. If one of them plays $exit$ the bargaining stops and the outcome is 0 for both of them. Each utility function U_i depends on three parameters of agent i: reservation price, RP_i (the maximum/minimum amount a seller/buyer is willing to accept), discount factor, γ_i, and deadline, T_i (agents prefer an earlier agreement, and after the deadline they *exit*). For this domain the state space is composed of the last action performed. The parameters are $T_i = 4$, $\gamma_i = 0.999$ and offers are in the range $[0-10]$ (integer values), therefore $|S| = 13$, $|A| = 13$ (the iPD had $|S| = 4$, $|A| = 2$). The buyer valuates the item to buy in 10 (in a scale from 1 to 10). One complete interaction consists of alternating offers. In the experiments, our learning agent is the buyer and the non-stationary opponent is the seller. The opponent uses two different strategies:

  - a fixed price strategy where the seller uses a fixed price P_f for the complete negotiation,
  - a flexible price strategy where the seller initially valuates the object at P_f, but after round 2 he is more interested in selling the object, so it will accept an offer $P_l < P_f$.

  We represent that strategy by $P_l = \{x \rightarrow y\}$. For example, the optimal policy against $P_f = \{8\}$ is to offer 8 in the first round (recall the game is discounted by γ_i in every round, so it's better to buy/sell sooner rather than later), receiving a reward of 2. However against $P_l = \{8 \rightarrow 6\}$ the optimal policy is to wait until the third round to offer 6, receiving a reward of $4\gamma_i^2$.

6.2 Drift and non-drift exploration approaches

In this section we present a summary of the results for the two domains in terms of average rewards. In the iPD we compared our proposed algorithms which use DE: R-MAX# ($\tau = 55$), MDP-CL(DE) ($w = 30$, Boltzmann exploration), and R-MAX# CL ($\tau = 90, w = 50$), against state of the art approaches MDP-CL [25] ($w = 30$), MDP4.5 [25] ($w = 30$), FAL [18], WOLF-PHC ($\delta_w = 0.3, \delta_l = 2\delta_w, \alpha = 0.8, \epsilon$-greedy exploration $= 0.2$) and the perfect agent that knows exactly when the opponent switches and best responds immediately. In Table 2 we summarize the results in terms of average rewards obtained by each agent against the probabilistic switching opponent for different values of η (switch probability). All the scores were obtained using the best parameters for each algorithm and the results shown are based on the average of 100 statistical runs. In all the cases, R-MAX# CL obtained better scores than the rest. An * indicates statistical significance against MDP-CL(DE) and † against R-MAX#. MDP-CL(DE) obtains good results since it exploits the model and uses DE. However, note that we provided the optimal window w so it could remain competitive. Using only R-MAX# is not as good as R-MAX# CL since it explores continuously but it will not properly exploit the learned model. A limitation of R-MAX# CL is that as a heritage from MDP-CL it needs to find a good parameter w. WOLF-PHC shows almost the same performance against

Table 2 Iterated prisoner's dilemma

Average rewards of the test algorithms against an opponent that changes strategy with probability η
* and † represent statistical significance of R-MAX# CL against MDP-CL(DE) and R-MAX# respectively

Algorithm/η	0.02	0.015	0.01	Drift exp.
Perfect	2.323	2.319	2.331	–
R-MAX# CL	**2.051***†	**2.079***†	**2.086**†	Yes
MDP-CL(DE)	1.944	1.988	2.046	Yes
R-MAX#	1.691	1.709	1.725	Yes
WOLF-PHC	1.628	1.627	1.629	Yes
MDP-CL	1.696	1.790	1.841	No
MDP4.5	1.680	1.772	1.829	No
FAL	1.625	1.658	1.725	No

Table 3 Bilateral negotiation

Average rewards and percentage of successful negotiations of the test algorithms against an opponent with a probabilistic non-stationary opponent in the negotiation domain
* and † represent statistical significance of R-MAX# CL against MDP-CL(DE) and R-MAX# respectively

Algorithm	$AvgR(A)$	$SuccessRate$	Drift Exp.
Perfect	**2.70**	**100.0**	–
R-MAX# CL	**1.95***	74.9	Yes
R-MAX#	1.91	70.5	Yes
MDP-CL(DE)	1.73	82.0	Yes
WOLF-PHC	1.71	88.5	Yes
MDP-CL	1.70	85.5	No
R-max	1.67	**90.6**	No

different switch probabilities, however its results are far from the best. MDP-CL and MDP4.5 obtained better results than FAL, but since none of them use DE they are not as good as our proposed approaches.

We performed a similar analysis for the negotiation domain. We removed FAL and MDP4.5 since they obtained the worst scores and do not have DE. In Table 3 we summarize the results in terms of average rewards and the percentage of successful negotiations obtained by each learning agent against a switching opponent. In this case, each interaction consists of 500 negotiations. The opponent uses 4 strategies ($F_p\{8\}$, $F_p\{9\}$, $F_l = \{8 \rightarrow 6\}$, $F_l = \{9 \rightarrow 6\}$); the switching round and strategy were drawn from a uniform distribution. R-MAX# CL obtained the best scores in terms of rewards and R-MAX# obtained the second best. In this domain MDP-CL(DE) and WOLF-PHC take more time to detect the switch and adapt accordingly obtaining lower rewards. However, they have a higher percentage of successful negotiations than the R-MAX# approaches. In this domain we note that not using DE (MDP-CL and R-MAX) results in failing to adapt to non-stationary opponents which yields suboptimal rewards.

These results show the importance of performing DE in different domains. In the next sections we present detailed analysis of MDP-CL(DE), R-MAX# and R-MAX# CL.

6.3 Further analysis of MDP-CL(DE)

The MDP-CL framework does not use DE which results in failing to detect some type of switches. We present two examples where MDP-CL(DE) it is capable of detecting those switches. In Fig. 5a the cumulative rewards against a Bully opponent that switches to TFT

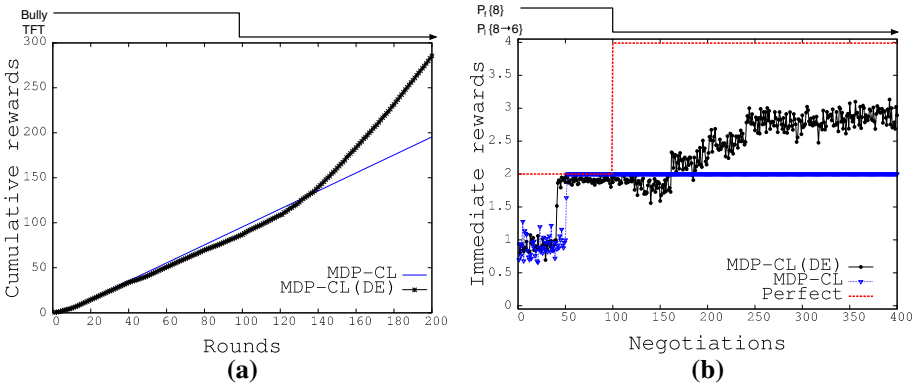


Fig. 5 On *top* of each figure we depict how the opponent changes between strategies during the interaction. Cumulative rewards of **a** MDP-CL ($w = 25$) with and without drift exploration, the opponent is Bully–TFT switching at round 100. **b** Immediate rewards of MDP-CL and MDP-CL(DE) against a non-stationary opponent in the alternating bargaining offers domain

at round 100 (deterministically) are depicted (average of 100 trials). The figure shows a slight cost associated to the drift exploration of MDP-CL (DE) before round 100 (thick line), after this point obtained rewards increase considerably since the agent has learned the new opponent strategy (TFT) and has updated its policy. In the negotiation domain, a similar behavior happens when the opponent starts with a fixed price strategy ($P_f = \{8\}$) and switches at round 100 (deterministically) to a flexible price strategy ($P_l = \{8 \rightarrow 6\}$). In Fig. 5b the immediate rewards of MDP-CL with and without DE are depicted (average of 100 trials), also we depict the rewards of a perfect agent which best responds at every round. The figure shows that MDP-CL is not capable of detecting the strategy switch, from rounds 50 to 400 it uses the same action and therefore obtains the same reward. In contrast MDP-CL (DE) explores with different actions (therefore it seems unstable) and due to the DE is capable of detecting the strategy switch. However, it needs several rounds to relearn the optimal policy after which it starts increasing its rewards (at round 175 approximately). After this round the rewards keep increasing and eventually converge around the value of 3.0 since even when it knows the optimal policy it keeps exploring.

In Table 4 we present results in terms of average rewards ($AvgR$) and percentage of successful negotiations ($SuccessRate$) while varying the ϵ parameter (of ϵ-greedy as DE) from 0.1 to 0.9 in MDP-CL (DE) in the negotiation domain (average of 100 trials). We used $w = 35$ since it obtained the best scores and an * represents statistical significance with respect to MDP-CL. These results indicate that using a moderate DE ($0.1 \leq \epsilon \leq 0.5$) increases the average rewards. A higher value of ϵ causes too much exploration. Thus, the agent cannot exploit the optimal policy which results in poor performance. On the one hand, increasing ϵ increases the average rewards of successful negotiations, this happens because using drift exploration it is possible to detect the switch and therefore updating the immediate reward to 4.0 (after the switch). On the other hand, the number of successful negotiations is reduced while increasing the exploration. This is the common exploration-exploitation trade-off which causes too much or too little exploration to obtain worse results in contrast to moderate exploration ($\epsilon = 0.4$) obtaining the best average rewards.

These results show that adding a general DE (for example with ϵ-greedy exploration) helps to detect switches in the opponent that otherwise would have passed unnoticed. However, as we said before, parameters such as window size, threshold (of MDP-CL), and ϵ (for DE)

Table 4 Comparison of MDP-CL and MDP-CL(DE) while varying the parameter ϵ (using ϵ-greedy as drift exploration) in terms of average rewards ($AvgR$), percentage of successful negotiations ($SuccessRate$)

ϵ	$AvgR(A)$	$SuccessRate$
0.1	1.796*	**92.1***
0.2	1.782*	91.0
0.3	1.776*	88.9
0.4	**1.801***	86.1
0.5	1.753	83.5
0.6	1.694	80.7
0.7	1.619	78.2
0.8	1.506	73.7
0.9	1.385	68.1
Average	1.679	82.4
MDP-CL	1.726	88.9

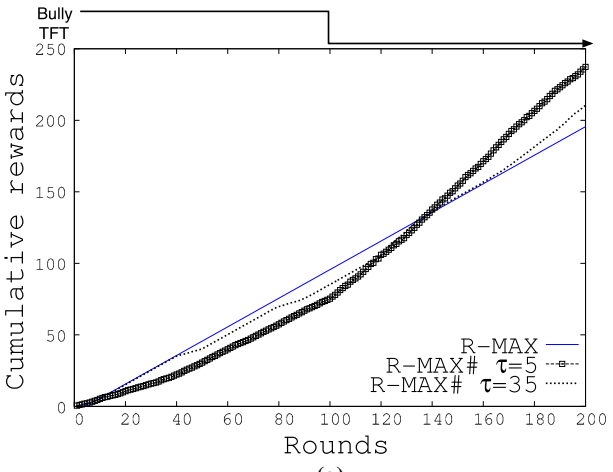
* Represents statistical significance with respect to MDP-CL

should be tuned in order to efficiently detect switches. In the next section we analyse R-MAX# and show how it helps minimizing the aforementioned drawbacks of parameter tuning present in MDP-CL(DE).

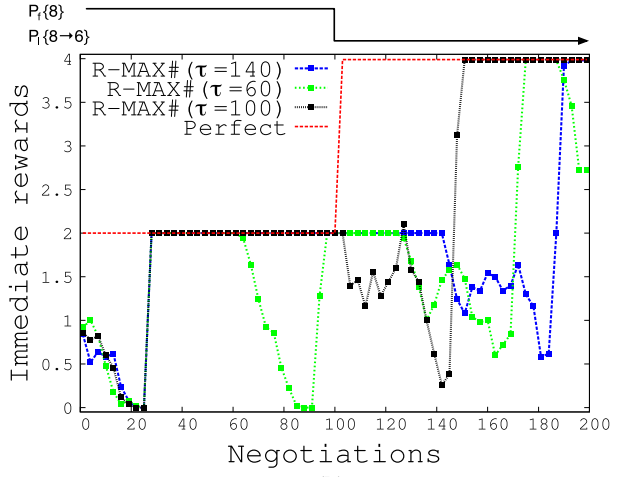
6.4 Further analysis of R-MAX#

We propose another way of performing DE using an implicit approach. R-MAX# has two main parameters: a threshold m that dictates whether a state-action pair is assumed to be known (same as R-MAX), and a threshold τ that controls how many rounds should have passed for a state-action pair to be considered forgotten. First we analyze the effect of τ. In Fig. 6a we present the cumulative rewards of R-max (straight line) and R-MAX# with $\tau = 5$ (thick line) and $\tau = 35$ (dotted line) against a Bully–TFT opponent (average of 100 trials), we used $m = 2$ for all the experiments since it obtained the best scores. For R-MAX#, a $\tau = 5$ makes the agent explore continuously causing a decrease of rewards from rounds 20 to 100. However, from round 100 rewards immediately increase since the agent detects the strategy switch. Increasing the τ value ($\tau = 35$) reduces the DE and also reduces the cost in rewards before the switch (at round 100). However, it also impacts the total cumulative rewards, since it takes more time to detect the switch (and learn the new model). Here we show an important trade-off when choosing τ: a small value causes a continuous exploration which quickly detects switches but has a cost before the switch occurs, while a large τ reduces the cost of exploration and therefore the switch will take more time to be noticed. It is important to note that R-MAX# is capable of detecting the switch in strategies. In contrast to R-max, which shows a linear result (in the figure) since it keeps acting against a Bully opponent, when in fact, the opponent is TFT.

In Fig. 6b we depict the immediate rewards of R-MAX# with $\tau = 60$, $\tau = 100$ and $\tau = 140$ against the opponent that changes from a fixed price to a flexible price strategy in the negotiation domain. In this figure we note that our agent starts with an exploratory phase which finishes at approximately round 25, then it uses the optimal policy to obtain the maximum possible reward. The opponent switches to a different strategy at round 100. For example, A $\tau = 100$ means that those state-action pairs which after 100 rounds have not been updated will be reset. The agent will re-explore (rounds 105–145) until learning a new optimal policy against the new opponent strategy (from round 145). Now, the optimal policy is different (since the opponent has changed) and R-MAX# is able to obtain the optimal



(a)



(b)

Fig. 6 On *top* of each figure we depict how the opponent changes between strategies during the interaction. **a** Cumulate rewards against the Bully–TFT opponent in the iPD using R-MAX# and R-MAX. **b** Immediate rewards of R-MAX# with $\tau = 60$, $\tau = 100$ and $\tau = 140$, and a perfect agent which best responds to the opponent in the negotiation domain

reward. The previous case shows, a $\tau = 100$ which was deliberately chosen to match the switching time of the opponent showing that R-MAX# can adapt to switching opponents and learn the optimal policy against each different strategy. The figure also shows what happens when τ does not match the switching time. Using a $\tau = 60$ means that the agent will re-explore the state space more frequently, which explains the decrease in rewards from round 65 to 95 and 125 to 160 (since both are exploratory phases), the first exploratory phase (at round 65) shows that what happens when no change has occurred where the agent returns to the same optimal policy. However, the second exploration phase shows a different result, since starting from round 160 it has updated its optimal policy and can exploit it. Using a $\tau = 140$, means less exploration which can be seen by the stability from round 25 to 145,

Table 5 Comparison of R-max and R-MAX# with different τ values in terms of average rewards ($AvgR$) and percentage of successful negotiations ($SuccessRate$)

τ	$AvgR(A)$	$SuccessRate$
20	1.101	61.2
40	1.375	61.4
60	1.643	71.6
80	2.034*	77.1
90	2.163*	80.4
100	**2.164***	80.9
110	2.043*	81.0
120	1.942*	81.0
140	1.746	80.8
160	1.657	83.5
180	1.736	88.9
Average	1.782	77.0
R-MAX	1.786	**90.6**

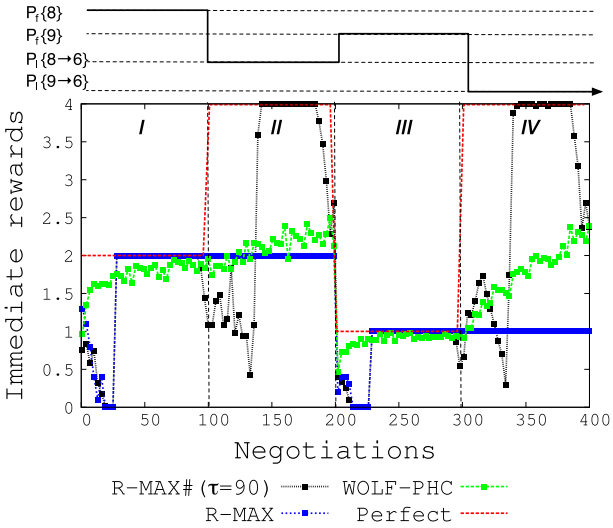
* Indicates statistical significance with R-MAX

from that point to round 190 the agent re-explores and updates its optimal policy at round 190.

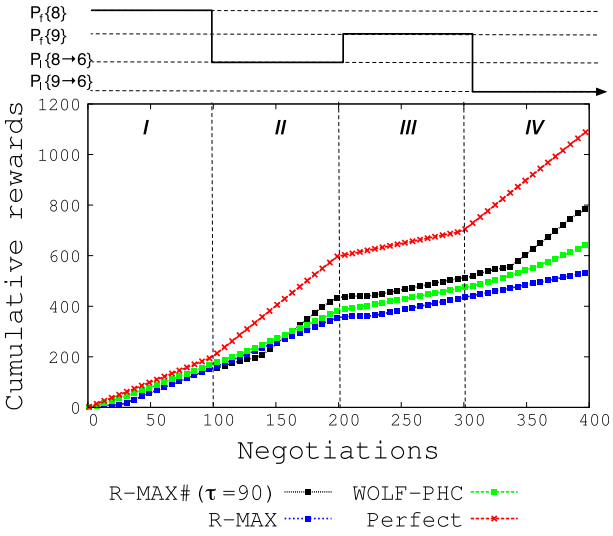
In Table 5 we present in more detail different results while using R-MAX# with different τ values against the switching opponent, an * indicates statistical significance with R-MAX. A small τ value is not enough to learn an optimal policy, which results in a low number of successful negotiations. In contrast a high τ does not explore enough and takes too much time to detect the switch which results in lower rewards. In this case a τ between 80–120 yields the best scores because it is enough to learn an appropriate opponent model while providing enough exploration to detect switches. R-max obtained the best score in successful negotiations because it learns an optimal policy and uses that for the complete interaction even when it is a suboptimal policy in terms of rewards for half of the interactions.

More complex opponents Previous experiments used only two opponent strategies, but we now consider four different strategies in the negotiation domain to compare the approaches of R-MAX#, R-max and WOLF-PHC. The strategies are (i) $F_p\{8\}$, (ii) $F_l = \{8 \rightarrow 6\}$, (iii) $F_p\{9\}$ and (iv) $F_l = \{9 \rightarrow 6\}$. The opponent switches every 100 rounds to the next strategy. In Fig. 7 we show (a) the immediate rewards and (b) cumulative rewards of R-MAX#, R-MAX, WOLF-PHC and the perfect agent in a game with 400 negotiations. Each opponent strategy is represented by a number zone (I-IV). Each curve is the result of averaging over of 50 statistical runs.

In zone I, from rounds 0 to 25 R-max and R-MAX# explore and obtain low rewards, after this exploration phase they have an optimal policy and use it to obtain the maximum possible reward (2.0) at that time. In zone II (at round 100) the opponent switches its strategy and R-max fails to detect this switch using the same (suboptimal) policy. Since R-MAX# re-explores the state space (rounds 95–135), it obtains a new optimal policy which in this case yields the reward of 4.0. In zone III (at round 200) the opponent switches its strategy and in this case R-max is capable of updating its policy. This happens because there is a new part of the state space that is explored, but again at zone IV (round 300) the opponent switches to a flexible strategy which R-max fails to detect and exploit. This is in contrast to R-MAX# which is capable of detecting all switches in the opponent strategy and reaching the maximum possible reward. WOLF-PHC starts with higher rewards than the other approaches. However,



(a)



(b)

Fig. 7 On *top* of each figure we depict how the opponent changes among strategies during the interaction. **a** Immediate and **b** cumulative rewards of R-MAX# ($\tau = 90$), R-max and WOLF-PHC in the alternating offers domain

when the opponent switches its strategy (rounds 100 and 300) WOLF-PHC is not capable of quickly adapting to these changes which results in lower rewards.

Analyzing the cumulative rewards we can see that during starting the second exploration phase of R-MAX# the cumulative rewards are lower than those of R-MAX. However, after updating its policy, cumulative rewards start increasing and from round 170. At the end of the interaction it can be easily noted in the difference in cumulative rewards at the final round: 534.9 for R-MAX, 647.3 for WOLF-PHC and 789.2 for R-MAX#.

Table 6 Average rewards against opponents that switch between two stochastic strategies in the middle of of the game of 500 rounds (iPD domain)

Opponents	R-MAX#			R-MAX	WOLF-PHC
	τ =100	τ =200	τ =250		
$Pavlov_{0.1}Bully_{0.9}$–$Bully_{0.9}Pavlov_{0.1}$	1.63	1.71	**1.78**∗†	1.58	1.68
$Pavlov_{0.2}Bully_{0.8}$–$Bully_{0.8}$ $Pavlov_{0.2}$	1.63	1.71	**1.76**∗†	1.60	1.73
$Pavlov_{0.3}Bully_{0.7}$–$Bully_{0.7}Pavlov_{0.3}$	1.64	1.73	**1.77**∗	1.70	1.75
$TFT_{0.1}Bully_{0.9}$–$Bully_{0.9}TFT_{0.1}$	1.41	1.45	**1.47**∗†	1.40	1.30
$TFT_{0.2}Bully_{0.8}$–$Bully_{0.8}$ $TFT_{0.2}$	1.38	1.40	**1.44**∗†	1.40	1.23
$TFT_{0.3}Bully_{0.7}$–$Bully_{0.7}$ $TFT_{0.3}$	1.33	1.38	**1.40**†	1.39	1.16
$TFT_{0.1}Pavlov_{0.9}$–$Pavlov_{0.9}TFT_{0.1}$	2.37	2.50	2.75	**2.91**	2.16
$TFT_{0.2}Pavlov_{0.8}$–$Pavlov_{0.8}TFT_{0.2}$	2.43	2.60	2.63	**2.88**	2.13
$TFT_{0.3}Pavlov_{0.7}$–$Pavlov_{0.7}TFT_{0.3}$	2.35	2.51	2.65	**2.86**	2.09
Average	1.79	1.88	**1.96**	**1.96**	1.69

* Indicates statistical significance of R-MAX# ($\tau = 250$) with R-MAX
† of R-MAX# ($\tau = 250$) with WOLF-PHC

Stochastic opponents Opponents in previous experiments have used deterministic strategies, however this is not a limitation for R-MAX#. For this reason we experimented against stochastic strategies in the iPD domain. In this case each strategy will be a mixture of two deterministic strategies. For example, one stochastic strategy is $Bully_{0.2}Pavlov_{0.8}$. This strategy represents an opponent which uses Bully with 0.2 probability and with 0.8 probability uses a Pavlov strategy.

We tested against stochastic non-stationary opponents that switched among two stochastic strategies in the middle of a game of 500 rounds. We evaluated R-MAX# ($\tau = 100, 200, 250$), R-max and WOLF-PHC in terms of average rewards of 100 trials in Table 6. Results show that R-MAX# obtains better scores than R-max and WOLF-PHC ($\delta_w = 0.3, \delta_l = 2\delta_w$, $\alpha = 0.8$, ϵ-greedy exploration $= 0.1$) against different opponents. A special case happens against opponents that mix TFT and Pavlov—in that case, not updating the model after a switch (as R-max does) obtains the best scores. This happens because the optimal policy before and after the opponent switch did not change.

Incomplete learning attributes For R-MAX# we assume that an expert will set the attributes that can represent the opponent strategy. However, it is an strong assumption to make in many domains. For this reason, we present experiments showing how R-MAX# behaves when the attributes used are not enough to completely represent the opponent strategy.

We performed experiments in the iPD with variations of TFT. Recall that TFT uses the last action of the opponent in the next action. A variation is Tit for Two Tats (TFTT), that needs two defects from the opponent to act with that action. It is more *forgivable* than TFT. Moreover, note that the optimal policy against TFTT is not to always *cooperate* (as with TFT), but rather alternate among *cooperate* and *defect*. In the same way we can generalize to TFT^n (Tit for n Tats). For example, the optimal policy against TFTTT is to use the cycle *defect-defect-cooperate*. However, to able to obtain these optimal policies it is important to keep the history of interactions, in particular the last h steps of interaction of both agents and then use those to represent the state space.

We performed experiments with R-MAX# and R-max against different variations of TFT while also using different different sizes for the history of interactions h. Table 7 shows the

Table 7 Average rewards against opponents that switch between two strategies in the middle of of the game of 500 rounds (iPD domain)

Opponents	R-MAX#						R-MAX	
	$\tau = 100$		$\tau = 200$		$\tau = 250$			
	$h = 1$	$h = 2$	$h = 1$	$h = 2$	$h = 1$	$h = 2$	$h = 1$	$h = 2$
TFTT–TFT	**2.62**$*$	2.52	**2.72**$*$	2.59	**2.86**$*$	2.58	**2.59**$*$	2.52
TFTTT–TFT	2.65	**2.67**	2.61	**2.73**$*$	2.73	**2.83**$*$	2.47	**2.68**$*$
TFTTTT–TFT	2.66	**2.67**	2.55	**2.60**	2.65	**2.70**	2.38	**2.57**$*$
Average	**2.64**	2.62	2.63	**2.64**	**2.75**	2.70	2.48	**2.59**

$*$ Indicates statistical significance between using one step of history ($h = 1$) and two steps ($h = 2$)

average rewards against switching opponents in games of 500 rounds (average of 100 trials). From the results we note that against the TFTT–TFT opponent using only one step of history was enough to learn the optimal policy and therefore it obtained better scores than using $h = 2$, since it took more rounds to learn the optimal policy (the space state increased). However, the results are different against TFTTT–TFT. Now, 1 step of history ($h = 1$) is not enough to learn the optimal policy against TFTTT. Analyzing the computed policies for R-MAX# using only one step of history showed that the agent alternates between *cooperate* and *defect* actions, which is suboptimal. In contrast, when increasing to 2 steps of history, R-MAX# is now capable of correctly representing the opponent strategy and obtaining the optimal policy which increased the average rewards (statistically significant in some cases).

These results show that in order to obtain an optimal policy the set of correct attributes to represent the opponent strategy are needed. However, this is difficult to obtain in different scenarios and also this will increase the state space and therefore the learning times. Results in the iPD scenario showed that using a simpler (not complete) representation was able to obtain competitive results when facing switching opponents.

6.5 Efficient exploration + switch detection: R-MAX# CL

We previously analyzed the effect of adding a general drift exploration to MDP-CL and now we experiment with the R-MAX# CL, which provides an efficient DE together with the MDP-CL framework.

First, we tested the approach against an opponent that switches strategies deterministically every 100 rounds in the iPD representing all the possible permutation between pairs of the three strategies used (TFT–Bully–Pavlov–Bully–TFT–Pavlov–TFT). The duration of the repeated game is 700 rounds.

The immediate rewards (average of 50 statistical runs) of MDP-CL, R-MAX#, WOLF-PHC and R-MAX# CL are shown in Fig. 8. For comparison, the rewards of a perfect agent that best responds immediately to the switches are depicted as a dotted line. In Fig. 8a, MDP-CL shows a stable approach since it learns the opponent model and immediately exploits it, but since it lacks a DE it fails to detect some strategy switches (e.g. the change from Bully to TFT at round 400 in Fig. 8a). In contrast, R-MAX# shows peaks throughout since it is continuously exploring. Its advantage is that it detects all strategy switches, such as Bully–TFT. In Fig. 8c we show the results for WOLF-PHC, which in this domain shows that is capable of adapting slowly to the changes, therefore never reaching the perfect score. In Fig. 8d we depict the immediate rewards of R-MAX# CL with $w = 50$, $\tau = 90$ (since these parameters obtained the

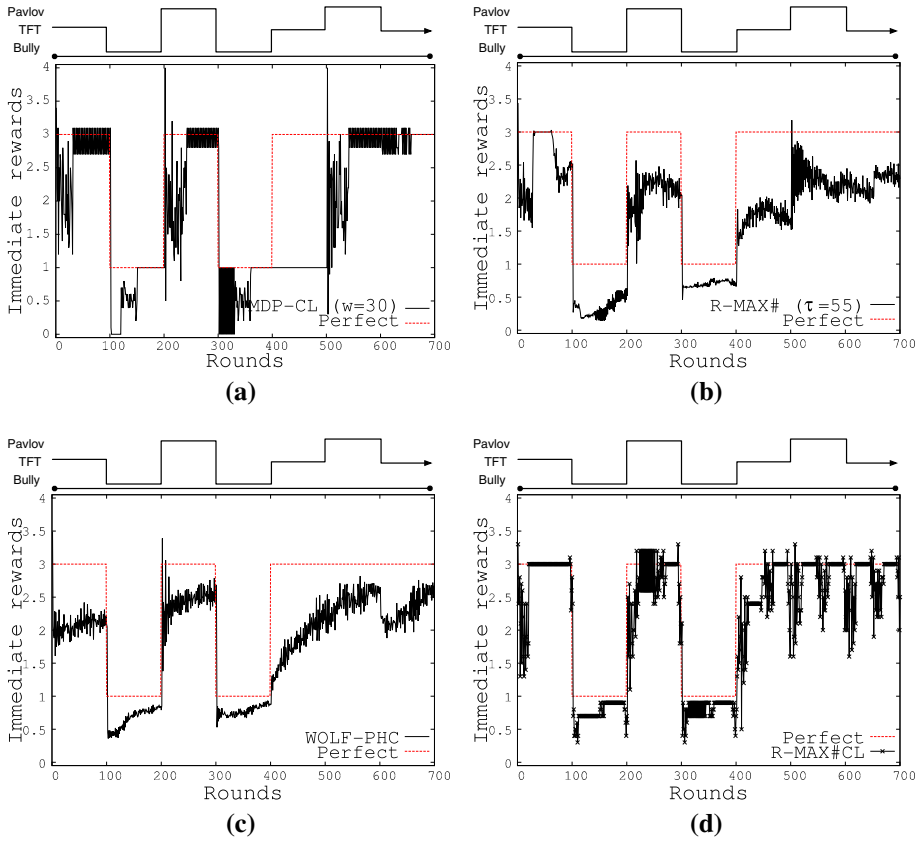


Fig. 8 On *top* of each figure we depict the opponent TFT–Bully–Pavlov–Bully–TFT–Pavlov–TFT that switches every 100 rounds in the iPD. Immediate rewards of **a** MDP-CL, **b** R-MAX#, **c** WOLF-PHC and **d** R-MAX# CL

best scores). This approach is capable of learning and exploiting the opponent model while keeping a DE, which enables the agent to detect switches. This experiment clearly shows the strengths of our approach, the need for an efficient DE and a rapid adaptation which results in higher utilities.

Lastly, we performed an experiment with random switching opponents in the negotiation domain. The interaction consists of 500 negotiations and the opponent uses 4 strategies ($F_p\{8\}$, $F_p\{9\}$, $F_l = \{8 \rightarrow 6\}$, $F_l = \{9 \rightarrow 6\}$); switching round and strategy were drawn from a uniform distribution. We compared the R-MAX# and R-MAX# CL approaches while varying the τ parameter: for R-MAX# CL we used $w = 50$ because it obtained the best results. A summary of the experiments is presented in Table 8, and each value is the average of 50 statistical runs.

As a baseline we used R-max in this setting (not shown in the table). We varied the parameter m from 2 to 20, and selected the best score ($m = 4$) nonetheless its results were the worst, with average rewards of 1.675 which are far from the obtained average rewards of 1.820 and 1.923 of R-MAX# and R-MAX# CL, respectively. Moreover, almost all values were statistically significantly better results than R-MAX. From Table 8 we can see that using almost any value of τ provides better results for R-MAX# CL

Table 8 Comparison of R-MAX# CL and R-MAX# with different τ values in terms of average rewards ($AvgR$), percentage of successful negotiations ($SuccessRate$)

τ	R-MAX#		R-MAX# CL $w = 50$	
	$AvgR(A)$	$SuccessRate$	$AvgR(A)$	$SuccessRate$
60	**1.791***	70.1	1.697	70.1
80	1.833*	69.5	**1.919***	**70.7**
90	1.716*	65.9	**1.898***	**70.3**
100	1.910*	70.5	**1.948***	**71.9**
110	1.875*	67.3	**1.947***	**71.1**
120	1.767*	66.9	**1.943***	**72.8**
140	1.847*	68.3	**2.110***	**75.1**
Average	1.820	68.4	**1.923**	**71.7**

* Indicates statistical significance with R-max which obtained an average reward of 1.675

than R-max# against a randomly switching opponent. This can be explained due to the fact that R-MAX# CL provides an efficient DE combined with a switch detection mechanism.

6.6 Summary of experiments

We tested two different domains in which DE is necessary to obtain an optimal policy, due to the non-stationary nature of the opponent's strategy. The iPD problem is a well-known and simple scenario and the negotiation task is a more realistic scenario (with a larger state and action spaces). In previous scenarios, the use of a switch detection mechanisms, such as the one used in MDP-CL, FAL or MDP4.5 works were not enough to deal with switching opponents (Sects. 6.2 and 6.3). When keeping a non-decreasing learning rate and exploration WOLF-PHC is capable of adapting in some scenarios, but it does so slowly (Sect. 6.4). The general approach of DE by means of ϵ-greedy or some softmax type of exploration, solves the problem since this exploration re-visits some parts of the state space that eventually will lead to detect switches in the opponent strategy (Sect. 6.3). However, the main limitation is that such algorithms need to be tuned for each specific domain and are not very efficient since they explore in an undirected way, not considering which parts of the state space need to be re-visited. Our approach, R-MAX#, which implicitly handles drift exploration, is generally better equipped to handle non-stationary opponents of different sorts. Its pitfall lies in its parameterization (parameter τ), which generally should be large enough so as to learn a correct opponent model, yet small enough to react promptly to strategy switches (Sect. 6.4). We performed experiments against stochastic opponents and with incomplete attributes to represent the opponent strategy. In the first case results showed that R-MAX# can adapt and optimize against non-stationary opponents that use stochastic strategies. In the second case, even when the attributes were not sufficient to to describe the perfect opponent strategy (and thus, optimal policy), R-MAX# showed competitive results against switching opponents. In realistic scenarios where we do not know the switching time of a non-stationary opponent, it is useful to combine both approaches, switch detection and implicit DE, as can be seen in R-MAX# CL (Sect. 6.5).

7 Conclusions and future work

In real world scenarios, agents must learn models of other agents (or people) whose behavior is not fully rational. Moreover, agents must rapidly adapt to behavioral changes. Here we shed light on what an exploration strategy should look like in non-stationary and adversarial environments. We propose a new type of exploration for detecting switches, *drift exploration*, that is able to efficiently explore the state space representation while being able to detect parts of the environment that have changed.

Our contribution is twofold. First, we present DE as a strategy for switch detection and encode this in a new algorithm, MDP-CL(DE). This approach keeps exploring during the complete interaction and performs an undirected exploration of the state space. The second contribution is a new implicit DE and encode this in an algorithm called R-MAX# for dealing with non-stationary opponents. R-MAX# makes efficient use of exploration experiences, taking into account which parts of the space state need to be re-visited, which results in rapid adaptation and efficient DE, to deal with the non-stationary nature of the opponent behavior. We provided theoretical results of R-MAX# that guarantee switch detection and near-optimal expected rewards. Also we performed a set of experiments in two domains showing that R-MAX# with a good parameterization converges to the optimal policy and is capable of adapting quickly to non-stationary opponent strategies either deterministic or stochastic. Its parameter, τ, shows a tradeoff: a large value enough to learn a sufficiently good model, yet small value to promote exploration for possible switches.

We now mention some limitations of the proposed approach and provide ideas on how to tackle them. We focus on drastic changes of behavior (from one stationary strategy to another from one step to another) rather than slow and continuous changes. In case of having learning opponents, special care needs to be taken to avoid learning noise [29]. Also, it is worth mentioning that there are scenarios where some actions can yield higher risks than others. In this cases, a hard-reset like we propose may not be the best approach and we leave that as an open question where ideas from safe-exploration [22,24] or social reward shaping [3] might be worth exploring. In more complex scenarios, where the space and action spaces will increase, we think it may be useful to leverage transfer learning techniques to obtain even faster learning rates when facing opponents that switch between strategies that have already been experienced by the learner, or are close to strategies that have been seen in the past [28,48]. Also, we propose as future work to use learning to select an appropriate τ parameter by keeping a record of the switching frequencies. Finally we want to perform experiments with humans since they naturally show changing and irrational behaviors in many scenarios.

Acknowledgments The first author was supported by a scholarship grant 329007 from the National Council of Science and Technology of Mexico (CONACYT). This research has taken place in part at the Intelligent Robot Learning (IRL) Lab, Washington State University. IRL research is supported in part by grants AFRL FA8750-14-1-0069, AFRL FA8750-14-1-0070, NSF IIS-1149917, NSF IIS-1319412, USDA 2014-67021-22174, and a Google Research Award.

References

1. Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time analysis of the multiarmed Bandit problem. *Machine Learning*, *47*(2/3), 235–256.
2. Axelrod, R., & Hamilton, W. D. (1981). The evolution of cooperation. *Science*, *211*(27), 1390–1396.
3. Babes, M., Munoz de Cote, E., & Littman, M. L. (2008). Social reward shaping in the prisoner's dilemma. In *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems*, (pp. 1389–1392). Estoril: International Foundation for Autonomous Agents and Multiagent Systems.

4. Banerjee, B.,& Peng, J. (2005). Efficient learning of multi-step best response. In *Proceedings of the 4th International Conference on Autonomous Agents and Multiagent Systems*, (pp. 60–66). Utretch: ACM.
5. Bard, N., Johanson, M., Burch, N., & Bowling, M. (2013). Online implicit agent modelling. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems*, (pp. 255–262). Saint Paul, MN: International Foundation for Autonomous Agents and Multiagent Systems.
6. Bowling, M., & Veloso, M. (2002). Multiagent learning using a variable learning rate. *Artificial Intelligence*, *136*(2), 215–250.
7. Brafman, R. I., & Tennenholtz, M. (2003). R-MAX a general polynomial time algorithm for near-optimal reinforcement learning. *The Journal of Machine Learning Research*, *3*, 213–231.
8. Busoniu, L., Babuska, R., & De Schutter, B. (2008). A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man and Cybernetics, Part C Applications and Reviews*, *38*(2), 156–172.
9. Carmel, D., & Markovitch, S. (1999). Exploration strategies for model-based learning in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, *2*(2), 141–172.
10. Cawley, G. C., & Talbot, N. L. C. (2003). Efficient leave-one-out cross-validation of kernel fisher discriminant classifiers. *Pattern Recognition*, *36*(11), 2585–2592.
11. Chakraborty, D., Agmon, N., & Stone, P. (2013). Targeted opponent modeling of memory-bounded agents. In *Proceedings of the Adaptive Learning Agents Workshop (ALA)*. Saint Paul, MN, USA.
12. Chakraborty, D., & Stone, P. (2008). Online multiagent learning against memory bounded adversaries. In *Machine Learning and Knowledge Discovery in Databases* (pp. 211–226). Berlin: Springer.
13. Chakraborty, D., & Stone, P. (2013). Multiagent learning in the presence of memory-bounded agents. *Autonomous Agents and Multi-Agent Systems*, *28*(2), 182–213.
14. Choi, S. P. M., Yeung, D. Y., & Zhang, N. L. (1999). An environment model for nonstationary reinforcement learning. In *Advances in Neural Information Processing Systems*, (pp. 987–993). Denver, CO, USA.
15. Da Silva, B. C., Basso, E. W., Bazzan, A. L., & Engel, P. M. (2006). Dealing with non-stationary environments using context detection. In *Proceedings of the 23rd International Conference on Machine Learnig*, (pp. 217–224). Pittsburgh, PA, USA.
16. Dietterich, T. G. (2000). Ensemble methods in machine learning. In *Multiple Classifier Systems*, (pp. 1–15). Berlin: Springer
17. Doshi, P., & Gmytrasiewicz, P. J. (2006). On the difficulty of achieving equilibrium in interactive POMDPs. In *Twenty-first National Conference on Artificial Intelligence*, (pp. 1131–1136). Boston, MA, USA.
18. Elidrisi, M., Johnson, N., & Gini, M. (2012). Fast learning against adaptive adversarial opponents. In *Proceedings of the Adaptive Learning Agents Workshop (ALA)*, Valencia, Spain.
19. Elidrisi, M., Johnson, N., Gini, M., & Crandall, J. W. (2014). Fast adaptive learning in repeated stochastic games by game abstraction. In *Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems*, (pp. 1141–1148). Paris, France.
20. Fulda, N., & Ventura, D. (2007). Predicting and preventing coordination problems in cooperative Q-learning systems. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, (pp. 780–785). Hyderabad, India.
21. Garivier, A., & Moulines, E. (2011). On upper-confidence bound policies for switching bandit problems. In *Algorithmic Learning Theory*, (pp. 174–188). Berlin: Springer.
22. Geibel, P. (2001). Reinforcement learning with bounded risk. In *Proceedings of the Eighteenth International Conference on Machine Learning*, (pp. 162–169). Williamstown, MA: Morgan Kaufmann Publishers Inc.
23. Gittins, J. C. (1979). Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society*, *41*(2), 148–177.
24. Hans, A., Schneegaß, D., Schäfer, A. M., & Udluft, S. (2008). Safe exploration for reinforcement learning. In *European Symposium on Artificial Neural Networks*, (pp. 143–148). Bruges, Belgium.
25. Hernandez-Leal, P., Munoz de Cote, E., & Sucar, L. E. (2013). Modeling non-stationary opponents. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems*, (pp. 1135–1136). International Foundation for Autonomous Agents and Multiagent Systems, Saint Paul, MN, USA.
26. Hernandez-Leal, P., Munoz de Cote, E., & Sucar, L. E. (2014). A framework for learning and planning against switching strategies in repeated games. *Connection Science*, *26*(2), 103–122.
27. Hernandez-Leal, P., Munoz de Cote, E., & Sucar, L. E. (2014). Exploration strategies to detect strategy switches. In *Proceedings of the Adaptive Learning Agents Workshop (ALA)*. Paris, France.
28. Hernandez-Leal, P., Taylor, M. E., Rosman, B., Sucar, L. E., & Munoz de Cote, E. (2016). Identifying and tracking switching, non-stationary opponents: a Bayesian approach. In *In Multiagent Interaction without Prior Coordination Workshop at AAAI*. Phoenix, AZ, USA.

29. HolmesParker, C., Taylor, M. E., Agogino, A., & Tumer, K. (2014). CLEANing the reward: counterfactual actions to remove exploratory action noise in multiagent learning. In *Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems*, (pp. 1353–1354). International Foundation for Autonomous Agents and Multiagent Systems, Paris, France.
30. Kakade, S. M. (2003). *On the sample complexity of reinforcement learning*. Ph.D. thesis, Gatsby Computational Neuroscience Unit, University College London.
31. Lazaric, A., Munoz de Cote, E., & Gatti, N. (2007). Reinforcement learning in extensive form games with incomplete information: The bargaining case study. In *Proceedings of the 6th International Conference on Autonomous Agents and Multiagent Systems*. Honolulu, HI: ACM.
32. Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the 11th International Conference on Machine Learning*, (pp. 157–163). New Brunswick, NJ.
33. Littman, M. L., & Stone, P. (2001). Implicit Negotiation in Repeated Games. In *ATAL '01: Revised Papers from the 8th International Workshop on Intelligent Agents VIII*.
34. Lopes, M., Lang, T., Toussaint, M., & Oudeyer, P. Y. (2012). Exploration in model-based reinforcement learning by empirically estimating learning progress. In *Advances in Neural Information Processing Systems*, (pp. 206–214). Lake Tahoe, NV.
35. MacAlpine, P., Urieli, D., Barrett, S., Kalyanakrishnan, S., Barrera, F., Lopez-Mobilia, A., Ştiurcă, N., Vu, V., & Stone, P. (2012). UT Austin Villa 2011: a champion agent in the RoboCup 3D Soccer simulation competition. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, (pp. 129–136). International Foundation for Autonomous Agents and Multiagent Systems, Valencia, Spain.
36. Marinescu, A., Dusparic, I., Taylor, A., Cahill, V., & Clarke, S. (2015). P-MARL: Prediction-based multi-agent reinforcement learning for non-stationary environments. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems.
37. Mohan, Y., & Ponnambalam, S. G. (2011). Exploration strategies for learning in multi-agent foraging. In *Swarm, Evolutionary, and Memetic Computing 2011*, (pp. 17–26). Springer.
38. Monahan, G. E. (1982). A survey of partially observable Markov decision processes: Theory, models, and algorithms. *Management Science*, *28*, 1–16.
39. Mota, P., Melo, F., & Coheur, L. (2015). Modeling students self-studies behaviors. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems*, (pp. 1521–1528). Istanbul, Turkey
40. Munoz de Cote, E., Chapman, A. C., Sykulski, A. M., & Jennings, N. R. (2010). Automated planning in repeated adversarial games. In *Uncertainty in Artificial Intelligence*, (pp. 376–383). Catalina Island, CA.
41. Munoz de Cote, E., & Jennings, N. R. (2010). Planning against fictitious players in repeated normal form games. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, (pp. 1073–1080). International Foundation for Autonomous Agents and Multiagent Systems, Toronto, Canada.
42. Ng, A. Y., Harada, D., & Russell, S. J. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, (pp. 278–287). Bled, Slovenia.
43. Puterman, M. (1994). *Markov decision processes: Discrete stochastic dynamic programming*. New York: Wiley.
44. Rejeb, L., Guessoum, Z., & M'Hallah, R. (2005). An adaptive approach for the exploration–exploitation dilemma for learning agents. In *Proceedings of the 4th international Central and Eastern European conference on Multi-Agent Systems and Applications*, (pp. 316–325). Springer, Budapest, Hungary.
45. Stahl, I. (1972). *Bargaining theory*. Stockolm: Stockolm School of Economics.
46. Stone, P., & Veloso, M. (2000). Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, *8*(3), 345–383.
47. Suematsu, N., & Hayashi, A. (2002). A multiagent reinforcement learning algorithm using extended optimal response. In *Proceedings of the 1st International Conference on Autonomous Agents and Multiagent Systems*, (pp. 370–377). ACM Request Permissions, Bologna, Italy.
48. Taylor, M. E., & Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research*, *10*, 1633–1685.
49. Vrancx, P., Gurzi, P., Rodriguez, A., Steenhaut, K., & Nowe, A. (2015). A reinforcement learning approach for interdomain routing with link prices. *ACM Transactions on Autonomous and Adaptive Systems*, *10*(1), 1–26.
50. Watkins, C., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*, 279–292.

51. Weinberg, M., & Rosenschein, J. S. (2004). Best-response multiagent learning in non-stationary environments. In *Proceedings of the 3rd International Conference on Autonomous Agents and Multiagent Systems*, (pp. 506–513). New York: IEEE Computer Society.
52. Zinkevich, M. A., Bowling, M., & Wunder, M. (2011). The lemonade stand game competition: Solving unsolvable games. *SIGecom Exchanges*, *10*(1), 35–38.