View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by CWI's Institutional Repository

Auton Agent Multi-Agent Syst
DOI 10.1007/s10458-016-9352-6

CrossMark

Efficiently detecting switches against non-stationary opponents

Pablo Hernandez-Leal[1] · Yusen Zhan[2] · Matthew E. Taylor[2] · L. Enrique Sucar[3] · Enrique Munoz de Cote[3,4]

© The Author(s) 2016

Abstract Interactions in multiagent systems are generally more complicated than single agent ones. Game theory provides solutions on how to act in multiagent scenarios; however, it assumes that all agents will act rationally. Moreover, some works also assume the opponent will use a stationary strategy. These assumptions usually do not hold in real world scenarios where agents have limited capacities and may deviate from a perfect rational response. Our goal is still to act optimally in these cases by learning the appropriate response and without any prior policies on how to act. Thus, we focus on the problem when another agent in the environment uses different stationary strategies over time. This will turn the problem into learning in a non-stationary environment, posing a problem for most learning algorithms. This paper introduces DriftER, an algorithm that (1) learns a model of the opponent, (2) uses that to obtain an optimal policy and then (3) determines when it must re-learn due to an opponent strategy change. We provide theoretical results showing that DriftER guarantees to detect switches with high probability. Also, we provide empirical results showing that our

Most of this work was performed while the first author was a graduate student at INAOE.

✉ 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 Washington State University, Pullman, Washington, USA

3 Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), Puebla, Mexico

4 PROWLER.io Ltd., Cambridge, United Kingdom

Published online: 26 November 2016

approach outperforms state of the art algorithms, in normal form games such as prisoner's dilemma and then in a more realistic scenario, the Power TAC simulator.

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

1 Introduction

When different agents interact in real world scenarios they may use different behaviors depending on the context they encounter. For example, in domains such as poker playing [9] agents may use different strategies depending on the opponent's behavior, in patrolling tasks [4] opponents may use different actions to reduce the ability of the defender to predict its behavior. In trading and negotiation scenarios, opponents use different strategies and change among them. In this context, there is one domain which has been used recently to perform research in energy markets: the Power TAC simulator [30]. In the simulator, competing brokers (agents) are challenged to maximize their profits by buying energy from a wholesale market and then offering energy services to customers. A champion agent from a previous competition was TacTex [42], which uses an approach based on reinforcement learning and prediction methods. Even though TacTex learns to bid efficiently (in terms of profit), it is not capable of adapting quickly to non-stationary opponents (that change suddenly to a different strategy). In general, when agents can change among several stationary strategies, they turn the environment into a non-stationary one. This is especially problematic for most learning algorithms which assume a stationary environment and most algorithms will not to react rapidly to sudden changes or will adapt more slowly, causing sub-optimal performance.

Works from machine learning have studied detection of changes mostly in supervised learning settings, this area is commonly known a concept drift [44]. However, this is only a partial representation of our problem since ours is a multiagent setting where actions are taken from each agent and rewards are based on those actions. In the area of reinforcement learning some approaches have studied how agents should act against non-stationary agents in order to converge to an equilibrium [10]. However, they have not been analyzed against opponents that change from one strategy to another.

Against this background, this paper's main contribution is to introduce DriftER, Drift (based on) Error Rate, which leverages the idea of concept drift to detect when the opponent has changed strategies based on a measure on predictive error. In order to learn how to act, DriftER assumes no prior information of the opponents instead, DriftER assumes to know the set of attributes the opponent uses to define its strategy[1] and starts with an exploratory policy. DriftER treats the opponent as part of a stationary environment using a Markov decision process to model its behavior [6] and keeps track of the quality of the learned MDP model. We provide theoretical bounds for detecting switches with high probability by making two assumptions: the opponent will remain stationary for some rounds and it is possible to bound the probability of exploration/mistakes made by the opponent. We also empirically test our algorithm in two distinct settings: normal-form repeated games and the Power TAC simulator. We compare with two types of state of the art algorithms, one specifically designed for the scenario [28] and another general algorithm for interacting in non-stationary environments [10]. The results show the effectiveness of our approach, outperforming state of the art algorithms in terms of total reward and accuracy.

[1] These can be, for example, previous actions of the agents.

The rest of the paper is organized as follows, in Sect. 2 we present the formalism of repeated games and Markov decision processes. In Sect. 3 we review related work to learning in non-stationary environments in multiagent systems and machine learning. In Sect. 4 we present the DriftER algorithm as well as its theoretical results. In Sect. 5 we present experimental results in two domains: repeated games and the Power TAC simulator. Finally in Sect. 6 we present conclusions and ideas for future work.

2 Preliminaries

In this section, first we present repeated games and then some important concepts of reinforcement learning used to perform opponent modeling.

2.1 Repeated games

Our approach is tested in the repeated games formalism. Consider two players (A and 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 $R_i(a, b)$, $\forall (a, b) \in \mathcal{A} \times \mathcal{B}$, $\forall i \in$ {A,B}. A *stage game* is a single bimatrix game and a series of *rounds* of the same stage game form a *repeated game*.

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. Another common strategy is the *minimax strategy*, this is, maximizing its payoff assuming the opponent will make this maximum as small as possible. The *security level* is the expected payoff a player can guarantee itself using a minimax strategy.

In single-agent decision theory, the notion of optimal strategy is the one that maximizes the agent's expected payoff for a given environment. In multiagent settings the situation is more complex, and the notion of an optimal strategy for a given agent is not meaningful since the best strategy depends on the choices of others. To solve this problem game theory has identified a solution concept known as Nash equilibrium.

Suppose that all players have a fixed action in a given game, if any player cannot increase its utility by *unilaterally* changing its strategy, then the decisions are in Nash equilibrium. Formally:

Definition 1 *(Nash equilibrium* [34]) A set of strategies $s = (s_1, \ldots, s_n)$ is a Nash equilibrium if, for all agents i, s_i is a best response to s_{-i}.[2]

One well known game is the prisoner's dilemma (PD). This is a two player game where the interactions can be modeled by the payoff matrix in Table 1 (and 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. A well known strategy in the

[2] Where s_{-i} denotes the set of all agents except i.

Table 1 A bimatrix game representing the prisoner's dilemma (PD), two agents can choose between two actions, cooperate (C) and defect (D). Each cell represent rewards (c, d, s, p) obtained by the agents depending on their actions

	C	D
C	c, c	s, d
D	d, s	p, p

Table 2 A bimatrix game representing the battle of the sexes (BoS) game, two agents can chose between two actions, going to the opera (O) or going to a Football match (F). Values $v_1, v_2 > 0$ represent rewards obtained by the agents

	O	F
O	v_1, v_2	$0, 0$
F	$0, 0$	v_2, v_1

iPD is called Tit-for-Tat (TFT) [5]; it starts by cooperating, then does whatever the opponent did in the previous round. 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 strategy which in the iPD behaves as an always defecting player. It should be noticed that these strategies can be defined only by the current state (last joint action) and do not depend on the time index; they are *stationary* strategies.

Another well known game is called battle of the sexes (BoS). The matrix describing this game is presented in Table 2 where $v_1 > 0$ and $v_2 > 0$ represent rewards obtained by the agents, with the condition of $v1 \neq v2$. This is a two-player coordination game: two people will meet in certain place in the city, the opera (O) or at a football match (F). One prefers opera and the other prefers the football match. There is no possible communication and players have to select where to go. This game has two pure Nash equilibria (O,O) and (F,F) Both pure equilibria are unfair since one player obtains better scores than the other. There is also one mixed Nash equilibrium where players go more often to their preferred event.

2.2 Reinforcement learning and opponent modeling

In reinforcement learning (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 Markov decision process (MDP) [38]. An MDP is defined by $\langle S, A, T, R \rangle$, where S is the set of states, A is the set of actions, 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 can be used to learn the opponent's strategy by inducing a MDP [6]. A special type of learning can happen by using a particular type of information. *Bounded memory opponents* are agents that use the opponent's 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 [6] the authors propose the adversary induced MDP (AIM) model, which uses as states a function of the past actions

of the learning agent. Note that the agent, by just keeping track of its own past moves can infer the policy of the bounded memory opponent.

3 Related work

In this section we review recent works about learning in non-stationary environments.

The machine learning community has developed areas related to non-stationary environments, one of those is change point detection [2,29,47]. Another related area is called concept drift [44]. Here, the approach is similar to a supervised learning scenario where the relation between the input data and the target variable changes over time [24]. In particular, Gama et al. [23] studied the problem of learning when the class-probability distribution that generates the examples changes over time. A central idea is the concept of *context*: a set of contiguous examples where the distribution is stationary. The idea behind the concept drift detection method is to control the online error rate of the algorithm. When a new training instance is available, it is classified using the actual model. Statistical theory guarantees that while the distribution is stationary, the error will decrease. When the distribution changes, the error will increase. Therefore, if the error is greater than a defined threshold, it means that the context has changed. The method was tested on both artificial and real world datasets. However, the mentioned approaches are not directly applicable to our multiagent scenario since our agent needs to (1) learn a model of the opponent (2) learn a policy to maximize its rewards and (3) detect when the opponent changes strategies over time.

In game theory, one well known algorithm for learning in repeated games is fictitious play [12]. However, it assumes the opponent is playing a stationary strategy. Other works have considered how to play against classes of opponents. For example, Manipulator [37] is designed against adaptive opponents with bounded memory in normal form games. AWESOME [16] and weighted policy learner (WPL) [1] are designed to converge to a Nash equilibrium but not to adapt to switching opponents. Recent approaches have focused on learning in two-player stochastic games, one example is the fast adaptive learner in stochastic games (FAL-SG) [21]. It consists of three main parts: (1) a meta-game model to transform the stochastic game into a simpler (matrix) representation; (2) a prediction model where a set of hypotheses according to the history of observations is used to predict the opponent's next action; and (3) the reasoning model, which differs from our approach since FAL-SG uses a modified version of the Godfather strategy [33] which is not a general strategy against all opponents and in all games. Also, FAL-SG shows an exponential increase in the number of hypotheses (in the size of the observation history) which may limit its use in larger domains.

Some related works were developed in the area of multiagent reinforcement learning [13]. Littman [32] proposed to extend the Q-learning algorithm [43] to zero-sum stochastic games. The algorithm uses the minimax operator to take into account the opponent actions. 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-case opponent (the one who tries to minimize the learning agent's utility). However, this may be unnecessary if the agent is allowed to adapt continually to its opponent. This is also the reason the algorithm is not rational (does not converges to the best response). The WoLF (win or learn fast) principle was applied into WoLF-PHC [10] which is an extension of Q-learning that performs hill-climbing in the space of mixed policies. This algorithm is designed to converge against opponents that slowly change its behavior, not sudden changes of strategies like the one we are facing. Another related approach is the reinforcement learning with context detection (RL-CD) [18]. The idea

is to learn several partial models and decide which one to use depending on the context of the environment. However, it has not been tested in multiagent scenarios.

Hidden-mode Markov decision processes (HM-MDPs) [15] are also designed for non-stationary environments. They assume the environment can be represented in a small number stationary environments (modes), which have different dynamics and need a different policy. It is assumed that at each time step there is only one active mode. However, HM-MDPs need to know the number of modes beforehand and they are more complex than our approach since they need to solve a POMDP [36] to obtain a policy to act.

When treating an opponent as part of the stationary (Markovian) environment, it is possible to learn the opponent's dynamics by inducing a MDP [6]. Modeling non-stationary strategies requires the model to be updated frequently (every time a change occurs). One approach that has been successful in identifying sudden strategy switches is MDP-CL [28]. This is a model-based multiagent learning technique designed to handle non-stationary opponents. The approach learns a model of the opponent in the form of a MDP using the interaction history. Then it computes an optimal policy against the opponent and the switch detection process starts. Pairs of opponent models are compared[3] every w rounds—if the difference between the two models is greater than a threshold then a switch has occurred and the algorithm restarts the learning phase, discarding the previous model. One weakness of MDP-CL is its parameters that can only be fine tuned after the entire game has been played. Also, the approach does not provide formal guarantees of switch detection.

Another related approach is designed to play against a class of opponents: memory bounded opponents whose memory size is bounded by a known value [14]. Convergence with Model Learning and Safety (CMLeS) achieves three objectives: (1) converges to following a Nash equilibrium joint-policy in self-play, (2) achieves close to the best response when interacting with a set of memory bounded agents, and (3) ensures an individual return that is very close to its security value when interacting with any other set of agents. However, this approach does not detect opponent switches.

Finally, ad hoc teamwork [3,7] is an area where agents needs to coordinate and has similar characteristics to our domain. However, we focus on opponents that change among several stationary strategies. DriftER assumes no prior information of the opponents and only starts with an exploratory policy. Also, DriftER provides a model based approach that can learn online models and adapt quickly to possible changes while having theoretical guarantees for switch detection.

4 DriftER

DriftER leverages insights from concept drift and opponent modeling techniques to identify switches in an opponent's strategy. DriftER treats the opponent as part of a stationary (Markovian) environment but tracks the quality of the learned model as an indicator of a possible change in the opponent's strategy. When a switch in the opponent strategy is detected, DriftER resets its model and restarts the learning process. An additional virtue of DriftER is that it can check for switches at every timestep.

DriftER pseudocode is presented in Algorithm 1. Since DriftER starts with no prior information it uses an exploratory process [11] for learning an opponent model in the form of an MDP (lines 3–6). When a model has been learned a switch detection process starts which

[3] One model uses a fixed size window of past interactions while the other uses all historic interactions.

predicts the next state of the process (line 8). An error probability is computed, and keeping track of this error will decide when a switch has happened (lines 9–15). When this happens, the learning phase is restarted (lines 17–18).

Algorithm 1: DriftER algorithm

```
   Input: Learning phase size w, treshold n_init, δ
1  model=∅; countError=0;
2  for r=1, ...,T do
3      if model ==null then
4          Use R-max to explore
5          if learningSamples==w then
6              model ← learnOpponentModel()
7      if model ≠ ∅ then
8          s ← predictNextState(model)
9          observe real state s'
10         p̂ ←computeError(s,s');
11         f ← computeConfInterval(error)
12         for i ← r − 1, ..., r − m steps do
13             Δ_i ← f_upper(p̂_i) − f_upper(p̂_{i−1})
14             if Δ_i > 0 then
15                 countError++
16         n ← adjustN(n_init, p̂, δ)  //see Section 4.4
17         if countError ≥ n then
18             model ← ∅
19         countError ← 0
```

4.1 Model learning

DriftER learns a model of the opponent which is used to compute a policy to act against it. DriftER's interaction with a stationary opponent generates Markovian observations which can be used to learn an MDP that represents the opponent's strategy assuming to know the representation (attributes) used by the opponent (e.g., the most recent action). This is because the history of interactions define the transition among states and the learning agent can induce an MDP [6] that models the opponent strategy and can compute an optimal policy against it π^* (assuming the opponent will remain fixed).

The formal framework used throughout this work is defined by $\langle S, A, T, R\rangle$ where: $S := \times_{O_i \in O} O_i$, i.e. each state is formed by the cross product of the set of attributes. The set of attributes O used to construct the states is assumed to be given by an expert. A are the actions of the learning agent. $T : S \times \mathcal{A} \rightarrow S$, is the transition function which is learned using counts $T(s, a, s') = \frac{n(s,a,s')}{n(s,a)}$ 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, the reward function 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. Solving this MDP dictates a policy which prescribes how to act against that opponent.

In all settings, after interacting with the opponent for w timesteps/rounds, the environment is learned using the R-max exploration [11] and we can use techniques such as value iteration [8] to solve the MDP.

4.2 Drift exploration

Many learning techniques decrease their exploration rate over time so that the learned (optimal) policy can be exploited. However, when facing non-stationary opponents whose model has been learned, an agent must balance exploitation (to perform optimally against that strategy) and exploration (to attempt to detect switches in the opponent). Thus, exploration cannot be completely suspended so as to detect changes in the structure of the environment at all times [19,25]. Opponent strategy switches can be particularly hard to detect if the strategies used before and after the switch are very similar. Such similarities can produce a "shadowing" effect [22] in the agents perception—an agent's optimal policy $\pi^\star$ will produce an ergodic set[4] of states against some opponent strategy, but if the opponent's switching strategy induce a similar MDP[5] where the policy $\pi^\star$ produce the same ergodic set, the agent will not detect something has changed (unless some exploration occurs).

DriftER uses an exploration coined as "drift exploration" that solves this shadowing effect by continuously exploring the state space even after an optimal policy has been learned. The only requirement of such exploration strategy is to make the entire state space "reachable" (i.e., the ergodic set produced by the new policy $\pi^\star_{explore}$ should be the entire state space). For example, ϵ-greedy or softmax exploration can be used for this purpose.

4.3 Switch detection

Approaches such as MDP-CL that compare pairs of models in fixed timesteps need two parameters to be tuned: the window size ($w \in \mathbb{N}$) that controls how often comparisons are made and the *threshold* $\in \mathbb{R}$ that defines how different models should be to mark a switch. Both parameters depend heavily on the domain and opponent and therefore renders the algorithm futile unless good evidence suggest a good parameter setting *a priori*. In contrast, our proposed algorithm keeps track on the opponent at every timestep in an efficient manner with a measure (prediction error) independent of the model of the opponent.

After learning a model of the opponent, DriftER must decide on each timestep if the model is consistent (the predictions using that model are correct) or the opponent has changed to a different strategy (the model has consistently shown errors). Using the current MDP that represents the opponent strategy, DriftER predicts the next state of the MDP (which for example can correspond to the opponent next action). In the next timestep DriftER compares the predicted and the experienced true state. This comparison can be binarized with *correct/incorrect* values. A Bernoulli process $S_1, S_2, \ldots, S_T$ will be produced, assuming a sequence of independent identically distributed events where $S_i \in \{0, 1\}$ and T is the last timestep. Let $\hat{p}_i$ be the estimated error (probability of observing *incorrect*) from S_1 to S_i, $i = 1, \ldots, T$. Then, the 95% confidence interval $[f_{lower}(\hat{p}_i), f_{upper}(\hat{p}_i)]$ over $S_1, S_2, \ldots, S_i$ is calculated for each timestep i using the Wilson score [45] such that the confidence interval will improve as the amount of data grows, where $f_{lower}(\hat{p}_i)$ and $f_{upper}(\hat{p}_i)$ denote the lower bound and upper bound of the confidence interval, respectively.

The estimated error, and its associated confidence interval, can increase for two reasons:

[4] In an ergodic set it is possible to go from every state to every state.

[5] Other authors have seen a related behavior which is called *observationally equivalent models* [20].

– The opponent is exploring or make mistakes.
– A switch has occurred in the opponent's strategy.

To detect the latter, DriftER tracks the finite difference of the confidence interval using the upper bound $f_{upper}(\hat{p}_i)$ at each timestep i. The finite difference is defined by

$$\Delta_i = f_{upper}(\hat{p}_i) - f_{upper}(\hat{p}_{i-1}), \quad i = 1, \ldots, T.$$

If $\Delta_i > 0$, $\Delta_{i-1} > 0$,...,$\Delta_{i-n+1} > 0$, where $n = 1, 2, \ldots$ is a parameter determined by the domain (if DriftER detects the confidence interval is increasing in the last n steps), then DriftER decides to restart the learning phase.

4.4 Initial estimation and stochastic opponents

Once DriftER has learned a model of the opponent it starts computing the estimated error and confidence intervals. However, it may take some rounds before having an accurate estimate. In order to improve the initial estimation, interactions from the learning phase can be used. With this objective, during the learning phase DriftER keeps the information from those interactions. When finishing the learning process it uses that information to produce an initial estimation of the error and confidence values.

Also, using a fixed value of n for all types of opponents may not be the best option. This may be particularly problematic against stochastic opponents because there will be a non-zero probability of incorrectly predicting the opponent's next move. Since we still need to check when the error increases in what follows, we propose to adjust the value of n accordingly to the estimated error $\hat{p}$.

We set a value n_{init} assuming a deterministic opponent strategy can been learned and n is adjusted against stochastic opponents following the function:

$$adjustN(n_{init}, \hat{p}, \delta) = n_{init} + \frac{log(\delta)}{log(\hat{p})}$$

where $n \leq n_{init} + C\frac{log(\delta)}{log(\hat{p})}$, C a constant value and $\delta > 0$ (described in the next section).

4.5 Theoretical guarantee for switch detection

Now, we provide a theoretical result to justify this method is capable of detecting opponent switches with high probability. In so doing we make the following assumptions:

– The opponent does not switch strategies while DriftER is in the learning phase.
– The probability of exploration or mistake of the opponent is at most ϵ for each timestep.

Theorem 1 *Let $\epsilon > 0$ and $\delta > 0$ be small constants. If $\Delta_i > 0$, $\Delta_{i-1} > 0$,...,$\Delta_{i-n+1} > 0$ and we set $n = O(\log \delta / \log \epsilon)$, then DriftER detects the opponent switch with probability $1 - \delta$.*

Proof If $\Delta_i > 0$, $\Delta_{i-1} > 0$,...,$\Delta_{i-n+1} > 0$, then DriftER decides to learn a new model. However, we point out that the $\Delta_i > 0$ may be caused by opponent's exploration/mistake. The worst case happens when DriftER incorrectly detects a switch while the opponent only made mistakes or explores, this is $\Delta_{i-j+1} > 0$, for all $j = 0, \ldots, n-1$ due to opponent's exploration/mistake. Let A denote the above event. Given $\Delta_i > 0$, $\Delta_{i-1} > 0$,...,$\Delta_{i-n+1} > 0$, the probability of event A, is

$$\mathcal{P}[A | \Delta_i > 0, \Delta_{i-1} > 0, \ldots, \Delta_{i-n+1} > 0] \leq \epsilon^n,$$

Since we assume that the probability of exploration or mistake is at most ϵ for each timestep. By the chain rule, the result follows. Then, set $\epsilon^n = \delta$, where δ is the probability of incorrectly detecting the switch, so $1 - \delta$ is the probability of detecting the switch correctly, finally $n = \log \delta / \log \epsilon$ and we have the result. □

This result shows that if DriftER decides to restart the learning phase, it does so because it detected the opponent switch with high probability $(1 - \delta)$ which makes the method robust.

4.6 DriftER example

We now contrast the behavior of DriftER and a learning agent that does *not* include a switch detection mechanism against the same non-stationary opponent in the BoS game. The opponent has two possible actions (O, F). It starts with a mixed strategy of [0.8, 0.2] and changes (in the middle of the interaction) to a pure strategy [0.0, 1.0] (which is the pure Nash equilibrium that is most beneficial to the opponent). Assume both learning agents learn an opponent model in the first 200 rounds, from that point they compute the predictive error and confidence values. Figure 1 depicts the upper value of the confidence over the error (f_{upper}) for a learning algorithm without switch detection mechanism (blue line) and DriftER (black thick line). Since the opponent uses a stochastic policy the error is close to 0.2, this happens because the agent predicts the opponent will use one action (O) and with probability 0.2 the opponent chooses F. At round 750 (marked with a vertical red line) the opponent changes to a pure Nash equilibrium action which is to use F in every round. This will result in suboptimal performance for the agent without switch detection, in contrast DriftER detects the switch (first double arrow) and starts a learning phase (between arrows) after which DriftER produces a new opponent model that is consistent with the new opponent strategy, therefore its error will decrease.

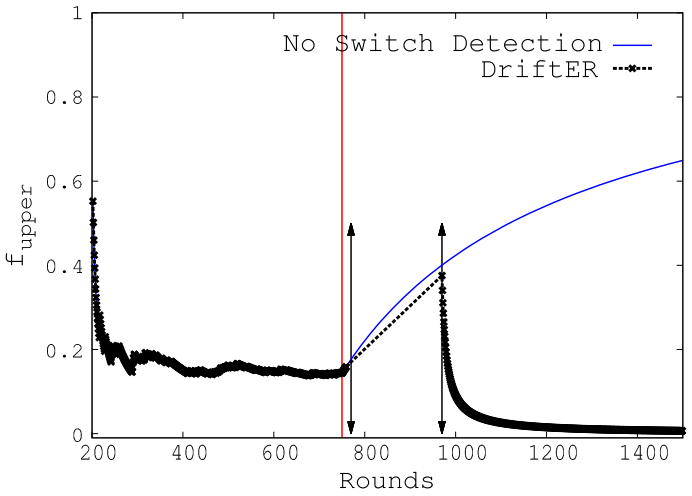

Fig. 1 Error probabilities of a learning algorithm with no switch detection and DriftER against an opponent that changes between two strategies in the middle of the interaction (*vertical bar*), *small arrows* represent DriftER learning phase after detecting the switch

5 Experiments

In this section we present experiments that show how DriftER performs in three different domains:

- Iterated prisoner's dilemma (iPD) (see Sect. 2.1) against non-stationary opponents using deterministic strategies. The iPD it is a well known domain where we can easily use different strategies for the opponent. We used the three most successful and known human crafted strategies that the literature has proposed: TFT, Pavlov and Bully as opponent strategies. These three strategies have different behaviors in the iPD and the optimal policy differs across them. Using the values $c = 3, d = 4, s = 0, p = 1$ (see Table 1) and a discount factor $\gamma = 0.9$, the optimal policy against a TFT opponent is always to cooperate, in contrast to the optimal policy against Bully which is always to defect. The optimal policy against Pavlov is to play the Pavlov strategy.
- Battle of the sexes and general-sum games against game theoretic strategies (see Sect. 2.1), including stochastic strategies. The most relevant strategies derived from game theoretic stability concepts that we found relevant to test are: pure Nash equilibria (when available), mixed Nash equilibria, minimax strategy and fictitious play [12].
- Energy markets in the Power TAC simulator. Power TAC models a retail electrical energy market, where brokers are challenged to maximize their profits by buying and selling energy in the wholesale and tariff markets:
  - In the tariff market brokers buy and sell energy by offering tariff contracts that specify price and other characteristics like early withdraw fee, bonus for subscription and expiration time. Customers choose among those different contracts and later they decide to continue or to change to a different one.
  - The wholesale market allows brokers to buy and sell quantities of energy for future delivery. It operates as a periodic double auction [46]. These markets are commonly known as day-ahead market, and are similar to many existing wholesale electric power markets, such as Nord Pool in Scandinavia or FERC markets in North America [31]. In this market, brokers make bids (offers) for buying or selling energy delivery between one and 24 h in the future. A wholesale broker can place a bid for buying or selling energy by issuing a tuple $\langle t, e, p \rangle$ that represents the timeslot t the broker makes a bid/ask for an amount of energy e (expressed in megawatt-hour MWh) at a limit price p of buying/selling. At each timeslot, Power TAC provides (as public information) market clearing prices and the cleared volume. It also provides as private information (only to each respective broker) the successful bids and asks [30]. A bid/ask can be partially or fully cleared. When a bid is fully cleared the total amount of energy will be sent at the requested timeslot, if a bid was partially cleared the offer was accepted but there is not enough energy and only a fraction of the requested energy will be sent.

The objectives of the experiments are threefold:

- Test DriftER against opponents that change among deterministic strategies and stochastic strategies.
- Test the behavior of its parameters.
- Show results of DriftER performance in a more realistic domain.

Comparisons were performed against MDP-CL [28] since it is an algorithm for acting against non-stationary opponents that change suddenly among strategies and WOLF-PHC [10] since it can learn to play against non-stationary opponents that slowly change behavior.

Table 3 Average rewards with standard deviation for different non-stationary opponents

Opponent	Omniscient	DriftER	MDP-CL	WOLF-PHC
Bully-TFT	2.0	**1.20 ± 0.30***	0.95 ± 0.01	1.09 ± 0.17
Bully-Pavlov	2.0	**1.67 ± 0.06**†	**1.67 ± 0.03**	1.47 ± 0.19
Pavlov-TFT	3.0	**2.82 ± 0.02**†	2.81 ± 0.01	2.09 ± 0.44
Pavlov-Bully	2.0	**1.78 ± 0.02***†	1.69 ± 0.07	1.51 ± 0.28
TFT-Bully	2.0	**1.77 ± 0.03***†	1.66 ± 0.13	1.34 ± 0.22
TFT-Pavlov	3.0	**2.82 ± 0.02**†	2.81 ± 0.01	2.08 ± 0.38
Average	2.33	**2.00 ± 0.10**	1.93 ± 0.26	1.60 ± 0.28

Bold values indicate the best scores
Each opponent change from one strategy to another in the middle of the interaction in a repeated game of 100 rounds. Results are the average of 100 iterations
* Indicates statistical significance with MDP-CL and † with WOLF-PHC (using Wilcoxon signed-rank test with $\alpha = 0.05$)

5.1 Deterministic strategies

In this section we present comparisons among DriftER, MDP-CL and WOLF-PHC in the iPD. The opponent changes from one strategy to another in the middle of the interaction and one repeated game consists of 100 rounds.

In Table 3 we show the average rewards of the three learning approaches and the Omniscient agent (that knows when the switch happens and best responds immediately) against switching opponents in the iPD (all possible pairs of strategies). Results show that DriftER obtained the best scores on average. DriftER obtained statistically significant better results than WOLF-PHC against most opponents and against MDP-CL in half of the cases. WOLF-PHC is slower than DriftER to adapt to changes obtaining suboptimal scores. MDP-CL is capable of detecting switches in the opponent faster than WOLF-PHC, however, since MDP-CL is not applying drift exploration it is not capable of detecting some changes between strategies, in particular against the opponent Bully-TFT (i.e., an opponent that first plays Bully and then switches to TFT). Thus, MDP-CL fails to detect the switch which results in results far from the optimal. In contrast, DriftER is capable of detecting switches in fast way and with drift exploration is capable of detecting all opponent switches.

5.2 Stochastic strategies

In the following setting the non-stationary opponents use common game theoretic strategies: pure Nash equilibria, mixed Nash, minimax strategy and fictitious play [12]. The opponents change from one to another during the interaction and DriftER's goal is to adapt quickly to these switches. Note that this type of opponents are found in the multiagent learning recent literature [16,17].

First, we present how the parameter n affects DriftER against switching opponents. The opponent will start with a mixed Nash equilibrium strategy $[0.2, 0.8]$ and will change to a minimax strategy $[0.8, 0.2]$ in the middle of the BoS game (round 750). We present experiments varying the parameter n with values $\{2, 4, 8, 20\}$ in a game of 1500 rounds. For each n we keep track of the round when a switch was detected (average of 100 trials). In Fig. 2 we depict a histogram showing the fraction of times when a switch was detected in certain interval of the game. From the figure we note that choosing a small value (2 in this case)

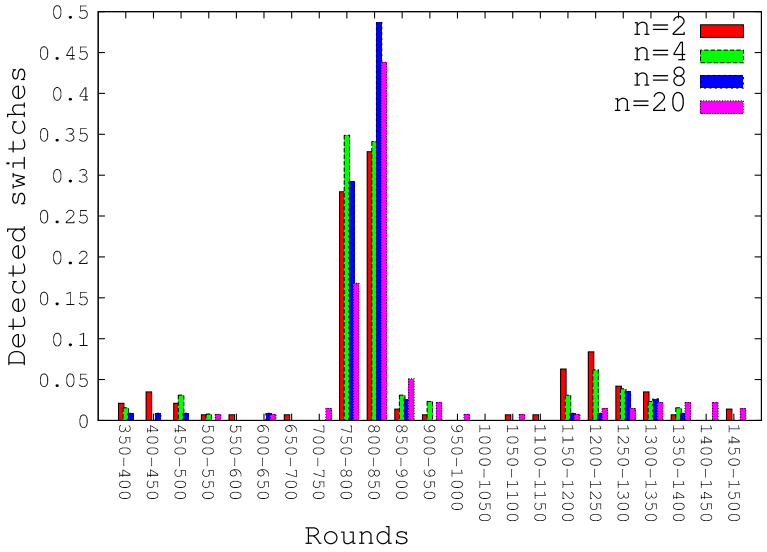

Fig. 2 Fraction of detected switches with different parameters of DritfER against a non-stationary opponent that changes from a minimax strategy to a mixed Nash strategy at round 750

may cause to erroneously detect switches (small red bars). A higher value (4 in this case) reduces the errors. If we increase the value to 8 (blue bars), the errors are almost reduced to zero, however, it may take more rounds to detect the switch. A large value (20 in this case) increases the number of rounds required to detect a switch in the opponent's policy.

Having analyzed the behavior of DriftER with respect to its main parameter we now show a descriptive example contrasting it with other related algorithms. We compared the behavior of DriftER, WOLF-PHC and MDP-CL in the BoS game (see Table 2; $v_1 = 25$, $v_2 = 100$). The opponent starts with a pure Nash strategy, at round 1000 it changes to a mixed Nash strategy. At round 2000, again it changes to a different pure Nash strategy. Results are the average of 100 iterations. In Fig. 3 we depict the immediate rewards of the learning algorithms. The opponent starts selecting a pure Nash equilibrium which DriftER quickly learns (less than 50 rounds). In contrast, WOLF-PHC needs a more time to converge to the best action (120 rounds approximately) and it will not stay with the best possible score (25). At round 1000 the opponent changes to a mixed strategy and both algorithms adapt correctly to the opponent. At round 2000 the opponent changes to a different pure Nash equilibrium. DriftER is capable of quickly adapting its model (and its policy) to this change obtaining the best possible score. WOLF-PHC takes more rounds to adapt and it does not completely exploit the opponent. We also present rewards for MDP-CL ($w = 160$). In contrast to DriftER that uses R-max exploration, MDP-CL uses a fixed window of random exploration and in this case it takes more time to exploit the model. When the opponent switches at round 1000 it is capable of adapting. At round 2000 the opponent changes again and we can observe that MDP-CL adjusts by steps until finally converging to the new opponent model. These steps happen every w rounds when the model comparison is performed to detect switches.

Normal-form repeated games Finally, we compared the learning algorithms in general-sum games with different actions and values. The games were randomly generated using Gamut [35] with the following characteristics: had at least one pure and one mixed Nash equilibria, with number of actions from 3 to 5, with values between -100 and 100 (see Tables 4 and 5).

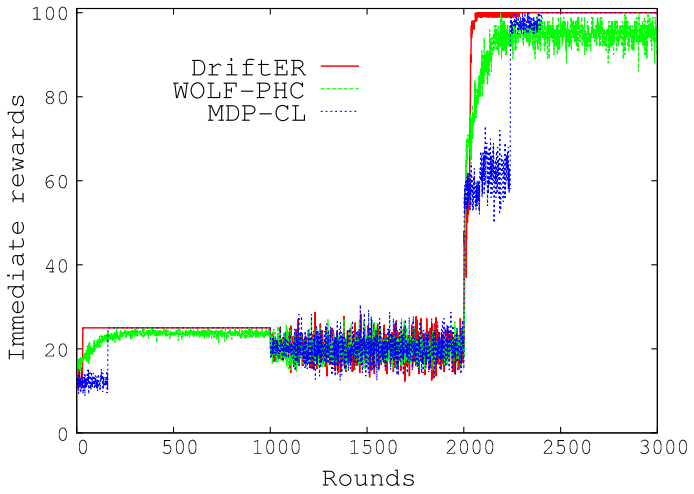

Fig. 3 Rewards obtained by DriftER, WOLF-PHC and MDP-CL in the BoS game against a non-stationary opponent that uses pure Nash and mixed Nash in a game of 3000 rounds (average of 100 trials). Switches happen every 1000 rounds

Table 4 Random games used in the experiments

	Game 1				Game 2		
	B_1	B_2	B_3		B_1	B_2	B_3
A_1	−29, −41	93, −56	56, −4	A_1	37, 35	45, 76	67,43
A_2	−17, −87	−70, −79	−44, −82	A_2	33, 94	38, 74	−94, −72
A_2	50, 49	−75, 76	27, −56	A_2	83, −61	−95, −5	99,32

	Game 3			
	B_1	B_2	B_3	B_4
A_1	−50, 20	73, −7	69, −45	83,22
A_2	−51, 89	88, 96	−55, 40	−26, −92
A_3	−58, 58	−41, 14	66, −46	0, −80
A_4	−62, 52	−94, −52	−40, −46	−94, −84

	Game 4				
	B_1	B_2	B_3	B_4	B_5
A_1	5, 7	32, 78	1,7	−55, −79	−1,0
A_2	89, 96	81, −45	−2661	73,78	−45, −68
A_3	29, 92	90, −53	−53, −46	45, −83	11,20
A_4	−89, 14	94, −99	−26, −10	89,22	67,−19
A_5	35, 84	67, 34	75,35	−6, 33	−16, −62

They have at least one pure and one mixed Nash equilibrium. Learning agents will play rows and opponents will play columns

Table 5 Pure and mixed Nash strategies for column player

Game Id	# actions	Pure Nash	Mixed Nash
Game 1	3	[0,0,1]	[0.680, 0.319,0]
Game 2	3	[0,1,0]	[0.0, 0.186, 0.813]
Game 3	4	[0,1,0,0]	[0.0, 0.879, 0.0, 0.120]
Game 4	5	[1,0,0,0,0]	[0.082, 0.0, 0.0, 0.917, 0.0]

Table 6 DriftER, MDP-CL and WOLF-PHC against non-stationary opponents in four random repeated games

Game Id	DriftER	MDP-CL	WOLF-PHC	Switch freq.
1	**35.69 ± 1.29**	35.24 ± 1.51	35.14 ± 1.11	1000
2	**58.03 ± 1.54**†	57.30 ± 0.67	56.21 ± 1.71	1000
3	71.76 ± 2.05	**75.34 ± 1.70**	71.68 ± 1.46	1000
4	**68.22 ± 2.19***	32.76 ± 8.41	68.03 ± 5.06	1000
Avg	**58.42 ± 1.77**	50.16 ± 3.07	57.72 ± 1.05	1000
1	37.72 ± 2.19	**37.76 ± 0.41**	35.60 ± 0.68	2000
2	**60.32 ± 0.85**†	59.78 ± 0.33	57.58 ± 1.06	2000
3	**75.61 ± 1.95**†	74.63 ± 1.95	72.87 ± 0.92	2000
4	**74.19 ± 0.74***†	53.67 ± 18.92	70.94 ± 2.58	2000
Avg	**61.96 ± 0.74**	56.46 ± 5.41	59.25 ± 1.31	2000

Bold values indicate the best scores
* Indicates statistical significance with MDP-CL and † with WOLF-PHC (using Wilcoxon rank-sum test with $\alpha = 0.05$)

In all cases, the opponent starts by playing a pure Nash strategy, then changes to a mixed Nash strategy and finally uses fictitious play. Switches happen every 1000 or 2000 rounds, the game consists of 3000 and 6000 rounds respectively. An * indicates statistical significance of DriftER with MDP-CL and † between DriftER and WOLF-PHC. Table 6 presents average rewards against non-stationary opponents with two different switching frequencies. Results show that DriftER obtained on average better results than WOLF-PHC. When switching frequency was 1000 rounds only one result is statistically significant. In contrast, when switching frequency increases to 2000 rounds DriftER can exploit the model for more rounds and therefore the results are statistically different.

The next section presents our last experimental domain in a more realistic application, double auctions in energy markets.

5.3 Energy markets in Power TAC

The champion agent from the inaugural competition of Power TAC was TacTex [42], which uses an approach based on reinforcement learning for the wholesale market and prediction methods for the tariff market. For modeling the wholesale market TacTex uses a modified version of Tesauro's representation of a double auction market [41]. The idea is that states represent agent's holdings, and transition probabilities are estimated from the market event

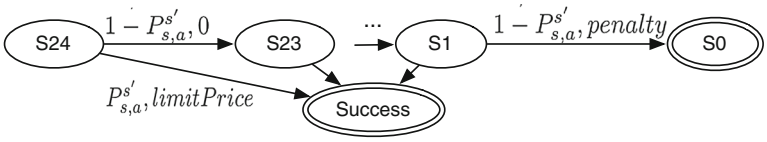

Fig. 4 Partial representation of the MDP broker in Power TAC, ovals represent states (timeslots for future delivery). *Arrows* represent transition probability and rewards as defined in Sect. 5.3

history. The model is solved via dynamic programming every time the agent had an opportunity to bid. TacTex uses a MDP to model the sequential bidding process. TacTex starts a game with no data and learns to bid online, and while acting its estimates are refined during the game. At each timeslot/timestep/round, it solves an MDP with all the data collected so far, providing the optimal limit price of the biddings for the next hours. Even though TacTex learns to quickly bid in an online environment, it does not adapt quickly to non-stationary opponents. However, many real-life strategies do not follow a static (stationary) regime throughout their interaction. Instead, they switch from one strategy to another (either to leave the opponent off-guard and guessing or just as a best response measure).

The experiments were designed focusing on the wholesale market of Power TAC. We compare the performance of three learning agents: DriftER, TacTex-WM (TacTex-WM is the part of TacTex applied only to the wholesale market), the champion of the 2013 competition, and MDP-CL, which is not specific for Power TAC but is an algorithm designed for non-stationary opponents. The opponent is non-stationary in the sense that it uses two stationary strategies: it starts with a fixed limit priced P_l and then in the middle of the interaction changes to a different (higher) fixed limit price P_h. The timestep at which the opponent switches is unknown to the other broker agent. Although we define a fixed limit price, and there is only a single opponent (other buying broker), Power TAC includes seven wholesale energy providers as well as one wholesale buyer to ensure liquidity of the market [30], introducing additional uncertainty and randomness in the simulation.

The problem of submitting bids to obtain energy in the wholesale market in PowerTAC is modeled as an MDP [42] (a graphical representation is depicted in Fig. 4):

- States: $s \in \{0, 1, \ldots, n, success\}$, represent the timeslots for future delivery for the bids in the market, the initial state is $s_0 := n$ and there are two terminal states: $0, success$.
- Actions: values $\in \mathbb{N}$ that represent limit prices for the offers in the wholesale market.
- Transitions: a state $s \in \{1, \ldots, n\}$ transitions to one of two states. If a bid is partially or fully cleared, it transitions to the terminal state $success$. Otherwise, a state s transitions to state $s - 1$. The transition probability is initially unknown.
- Rewards: In state $s = 0$, the reward is a balancing penalty value. In states $s \in 1, \ldots, n$, the reward is 0. In state $success$, the reward is the limit price of the successful bid.

The MDP's solution determines an optimal limit-price for each of the n states. Using this MDP the agent is always in states $1, \ldots, n$ of n concurrent bidding processes. Therefore, it solves the MDP once per timeslot, and submits the n optimal limit-prices to the n auctions.

The MDP that models the opponent has following parameters: the number of states was set to $|s| = 6$ (timeslots for buying energy), and the actions represent discretized limit prices $\{15, 20, 25, 30, 35\}$. The opponent started with a $P_l = 20$ and then changed to $P_h = 34$. In the first case, the learning agent should bid a $price > 20$ to be assigned bids. Later, when the opponent uses a limit price of 34, the only bid that will be accepted by the producer is 35. Both the learning agent and the opponent experience the same demand which is greater than the average energy needed to supply all buyers.

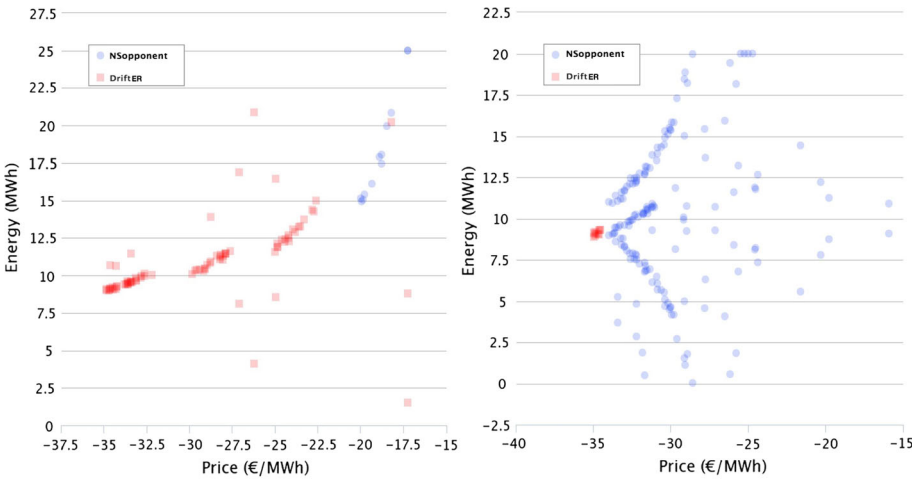

Fig. 5 Cleared transactions of DriftER (*red squares*) and the non-stationary opponent (*blue dots*) in the wholesale market in Power TAC. In **a** the opponent uses as limit price the value 20 whereas in **b** it uses a limit price of 34 (Color figure online)

Figure 5 depicts cleared transactions: red squares representing those of DriftER and blue circles for the non-stationary opponent (NSopponent). The more a transaction is on left the more the agent paid for that transaction. Figure 5a shows the behavior of DriftER against the first strategy the opponent uses, in this case there are three clusters for the cleared transactions of DriftER corresponding to the limit prices {25, 30, 35}.[6] This means that there are three possible actions that obtain a cleared bid. In contrast, cleared bids of the opponent always have a value lower than 20 (since it is the stationary limit price). In Fig. 5b, a similar graph is depicted showing the cleared transactions against the opponent's second strategy and after DriftER has updated its policy. Now, the opponent dominates the cleared transactions. The only limit price that produces cleared bids for DriftER is 35. These figures show how the optimal policy of the learning agent needs to be updated to cope with the opponent switching behavior.

An example of the behavior of DriftER and TacTex-WM against a switching opponent is depicted in Fig. 6. The figure shows that after round 100 (when the opponent changes its strategy), the error of TacTex-WM increases because it does not adapt rapidly to the opponent. In contrast, DriftER stops using its learned policy at timeslot 110 and restarts the learning phase, which ends at timeslot 135. At timeslot 135, DriftER shows a high f_{upper} value (since it is a new model) but the error decreases rapidly since at this point, DriftER has learned a new MDP and a new policy.

We now present results using independent-domain measures (accuracy and switch detection time) and specific-domain measures (traded energy and profit). The learned MDP contains a transition function for each (s, a) pair; comparing the predicted next state with the real (experienced) state gives an accuracy value. At each timestep the agent submits $|s|$ bids and its learned model predicts if those bids will be cleared or not. When the timestep finishes it receives feedback from the server and compares the predicted transactions with the real transactions. An average of the result of those predictions is the accuracy of each timestep.

[6] Power TAC takes these prices as negative since it as a buying action.

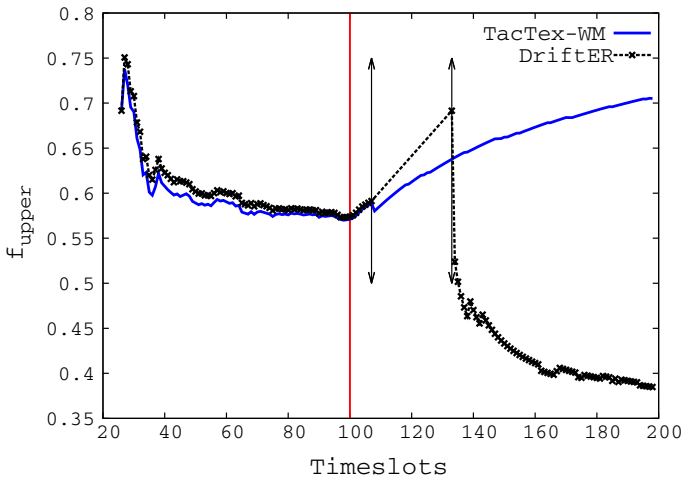

Fig. 6 Results from Power TAC, upper confidence over the error of TacTex-WM and DriftER against the non-stationary opponent. *Red line* shows when the opponent changed strategies and the *arrows* mark the learning phase of DriftER (Color figure online)

Table 7 Number of average timeslots for switch detection (average ± standard deviation), accuracy, and traded energy of the learning agents against a non-stationary opponent

	Avg. switch detection time	Accuracy	Traded energy
MDP-CL	85.0 ± 55.0	57.55 ± 28.56	2.9 ± 1.3
DriftER	**33.2 ± 13.6**	**67.60 ± 21.21**	**4.4 ± 0.5**

Bold values indicate the best scores

The switch detection times are the timesteps needed to detect a switch. In Power TAC there are several important measurements, but we focus on two: traded energy and profit. Traded energy is the total amount of energy obtained (in MWh) by the broker. This is a measure of indirect cost provided by Power TAC (the more time it takes to detect the switch, the less energy the agent successfully buys). Also, it is important to mention that whenever a broker does not fulfill its energy requirements it will be penalized proportionally (in €) by the balancing market with the amount of missing energy. Finally, profit is defined in Power TAC as the total income (in €) minus all the costs (balancing, wholesale, and tariff markets). We used default parameters for all other settings in Power TAC.

First, we evaluate the switch detection algorithms MDP-CL and DriftER. Additional experiments were performed to tune MDP-CL parameters. However, optimizing these parameters is time consuming since $w \in \mathbb{N}$ and $threshold \in \mathbb{R}$, $w = 25$, $threshold = 0.05$ were selected as the best values (based on accuracy). Table 7 reports the results with a competition of 250 timesteps. The opponent switched at timestep 100. Results are averaged over 10 independent trials. Results show that DriftER needs less time for detecting switches obtaining better accuracy (explained by the fast switch detection) and as a result is capable of trading more energy.

Now, we review the three approaches in terms of cumulative traded energy. Figure 7 depicts the learning agents and the switching opponent scores (timeslot when the opponent switches is displayed with a vertical line). From the figure we note that in the first part of the game (before the vertical line) TacTex-WM, MDP-CL and DriftER consistently increase their

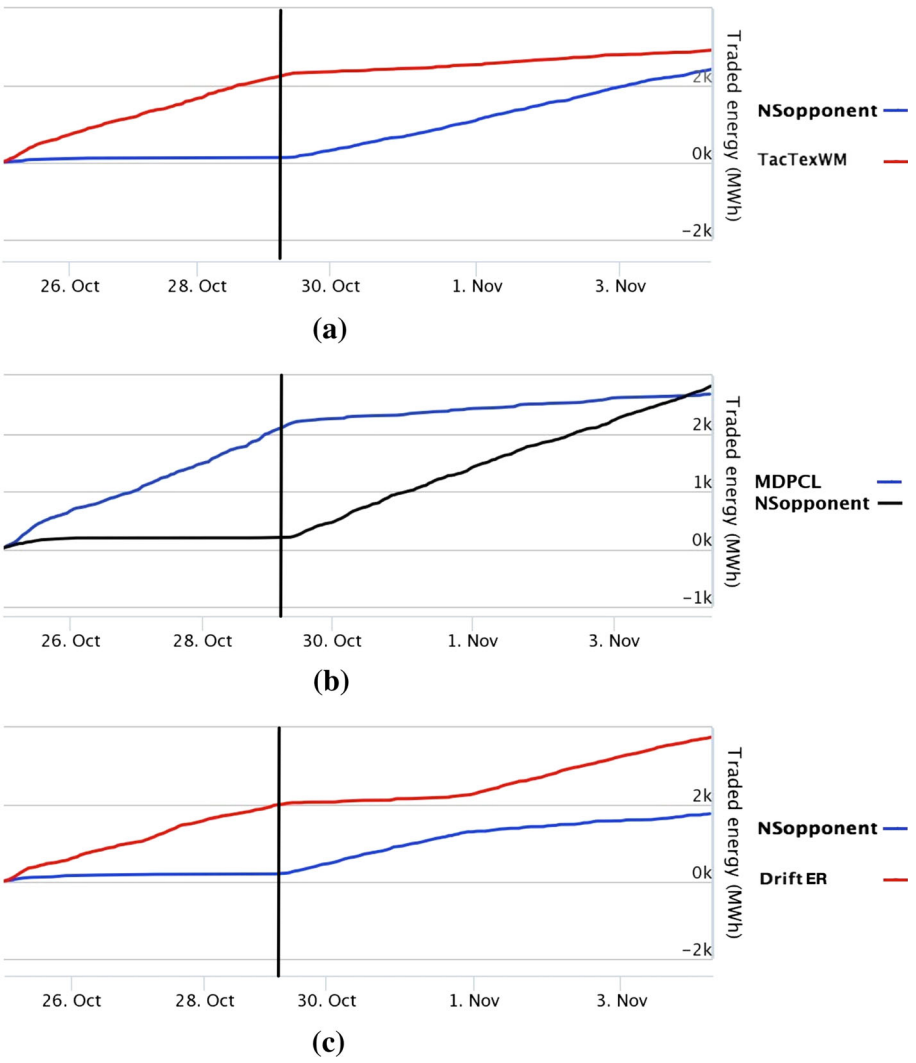

Fig. 7 Amount of traded energy for **a** TacTex-WM, **b** MDP-CL and **c** DriftER against the same type of non-stationary opponent in a competition of 250 timesteps. Timestep when the opponent switches is displayed with a *vertical line*

traded energy. In contrast, the traded energy for the opponent is severely limited since the learning agents are clearing most of their bids. However, at timeslot 100 the opponent changes its strategy and increases its cleared bids. Against TacTex-WM the opponent increases its traded energy consistently for the rest of the competition and its traded energy is almost the same at the end of the interaction (see Fig. 7a), which means that TacTex-WM lacks the capacity to adapt quickly to the new opponent strategy, reducing cumulative its traded energy. MDP-CL and DriftER (see Fig. 7b, c) are also affected by the change in the opponent strategy. However, note that DriftER starts increasing its traded energy after the switch detection, since DriftER now knows how to optimize against the new opponent strategy.

Table 8 Average profits of the learning agents against different types of non-stationary opponents

	Agent	Opponent
Fixed non-stationary opponent		
TacTex-WM	219.0 ± 7.5	228.7 ± 31.7
MDP-CL	261.6 ± 65.8	270.1 ± 75.5
DriftER	261.0 ± 38.9	228.7 ± 64.2
Noisy non-stationary opponent		
TacTex-WM	198.0 ± 41.3	197.6 ± 24.78
MDP-CL	250.1 ± 75.0	305.6 ± 41.18
DriftER	255.9 ± 39.9	229.0 ± 38.2

Now, we evaluate the algorithms in the complete Power TAC simulator measuring the profit of each agent. Recall that our approach considers bidding in the wholesale market. However, its not possible to isolate markets on Power TAC—thus, to perform experiments we propose to use the same strategy in the tariff market for all agents (TacTex-WM, MDP-CL and DriftER). This is, is to publish one flat tariff which is the average of the tariff's history. In this way, the profit results will mainly be affected by the performance on the wholesale market.

We evaluated the algorithms against two types of non-stationary opponents: using fixed and noisy strategies. With noisy strategies the opponent has a limit price $P_l = 20.0$ with a noise of ± 2.5 (bids are in the range [18.5–22.5]). Then, it switches to $P_h = 34.0$, with bids in the range $[31.5 - 36.5]$. The rest of the experimental setting remains the same as in the previous section. Table 8 shows the total profits of the learning agents against the non-stationary opponents with and without noise, averaged over 10 independent trails. TacTex-WM's profits are reduced and its standard deviation is increased when the opponent uses a range of values (noisy non-stationary opponent). MDP-CL shows competitive scores with fixed opponents, since it is capable of adapting to the non-stationary opponent. However, against a noisy opponent MDP-CL's results decrease on average and show a higher standard deviation, which is explained by the fact that in some cases MDP-CL failed to detect the opponent switch. DriftER shows a similar result as MDP-CL against fixed opponents and against noisy opponents it is still capable of detecting switches, showing better results than the rest.

5.4 Practical considerations of DriftER

In Sect. 4.5 we stated the assumptions made by DriftER to guarantee switch detection. However, in some domains these assumptions will not hold. For this reason, we briefly discuss what happen in those scenarios.

If the opponent does not remain stationary during DriftER's learning phase, the computed model will not be accurate. Since the optimal policy is derived from the learned model, DriftER will not obtain an optimal policy. Similarly, it is also important to consider which is the worst opponent that can play against DriftER. This is the opponent that uses a strategy during the learning phase of DriftER and changes immediately after this phase. The amount of reward lost in this circumstances (i.e., regret) will depend on the particular reward matrix. Finally, many algorithms put consideration into what happen when in self-play. While a more extensive analysis is needed, we note that in self-play DriftER agents will try to learn at the same time which will lead to poor models and performance.

6 Conclusions and future work

In real world scenarios where different agents interact witch each other it is reasonable to expect that they have different characteristics and different behaviors. Moreover, they probably will change behaviors during the interaction. In particular, we focus on the problem when another agent in the environment use different stationary strategies over time. This will turn the problem into learning in a non-stationary environment, posing a problem for most learning algorithms. This paper introduced DriftER, an algorithm that models an opponent as an MDP in order to compute an optimal policy against it. Then, it uses the learned model to estimate the opponent's behavior and tracks its error rate to detect opponent switches. When the opponent changes its strategy, the error rate increases and DriftER must learn a new model. Theoretical results provide a guarantee of detecting switches with high probability. Empirical results in repeated games show that DriftER can exploit the opponent model and quickly detect switches, obtaining better scores than the state of art algorithms for non-stationary environments. Results in the Power TAC simulator show that DriftER can be adapted to different and more realistic scenarios. Future work will address using transfer learning [40] ideas so that the previous model can be leveraged to promote fast learning of new opponent strategies. In particular, we are interested to reuse policies, similar to the work of Bayesian Policy Reuse [39] and BPR+ which works in multiagent settings [26,27].

Acknowledgements This research was supported partially by project CB-2012-01-183684 and scholarship 335245/234507 granted by Consejo Nacional de Ciencia y Tecnologia (CONACyT) Mexico. 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 NSF IIS-1149917, NSF IIS-1319412, USDA 2014-67021-22174, and a Google Research Award.

References

1. Abdallah, S., & Lesser, V. (2008). A multiagent reinforcement learning algorithm with non-linear dynamics. *Journal of Artificial Intelligence Research*, *33*(1), 521–549.
2. Adams, R. P., & MacKay, D. (2007). Bayesian online changepoint detection. arXiv:0710.3742v1 [stat.ML]
3. Albrecht, S. V., & Ramamoorthy, S. (2013). A game-theoretic model and best-response learning method for ad hoc coordination in multiagent systems. In *Proceedings of 15th international conference on autonomous agents and multiagent systems* (pp. 1155–1156).
4. Almeida, A., Ramalho, G., Santana, H., Tedesco, P., Menezes, T., Corruble, V., Chevaleyre, Y. (2004). Recent advances on multi-agent patrolling. In *Advances in artificial intelligence—SBIA 2004* (pp. 474–483). IEEE.
5. Axelrod, R., & Hamilton, W. D. (1981). The evolution of cooperation. *Science*, *211*(27), 1390–1396.
6. 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.
7. Barrett, S., & Stone, P. (2014). Cooperating with unknown teammates in complex domains: A robot soccer case study of Ad Hoc teamwork. In *Twenty-ninth AAAI conference on artificial intelligence* (pp. 2010–2016). Austin, Texas.
8. Bellman, R. (1957). A Markovian decision process. *Journal of Mathematics and Mechanics*, *6*(5), 679–684.
9. Bowling, M., Burch, N., Johanson, M., & Tammelin, O. (2015). Heads-up limit hold'em poker is solved. *Science*, *347*, 145–149.
10. Bowling, M., & Veloso, M. (2002). Multiagent learning using a variable learning rate. *Artificial Intelligence*, *136*(2), 215–250.
11. 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.
12. Brown, G. W. (1951). Iterative solution of games by fictitious play. *Activity Analysis of Production and Allocation*, *13*(1), 374–376.

13. 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.
14. Chakraborty, D., & Stone, P. (2013). Multiagent learning in the presence of memory-bounded agents. *Autonomous Agents and Multi-agent Systems*, *28*(2), 182–213.
15. 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, Colorado.
16. Conitzer, V., & Sandholm, T. (2006). AWESOME: a general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. *Machine Learning*, *67*(1–2), 23–43.
17. Crandall, J. W., & Goodrich, M. A. (2011). Learning to compete, coordinate, and cooperate in repeated games using reinforcement learning. *Machine Learning*, *82*(3), 281–314.
18. 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, Pennsylvania.
19. de Cote, E. M., 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, California.
20. 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.
21. 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.
22. 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.
23. Gama, J., Medas, P., Castillo, G., Rodrigues, P. (2004). Learning with drift detection. In *Advances in artificial intelligence—SBIA* (pp. 286–295).
24. Gama, J., Zliobaite, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)*, *46*(4), 44.
25. 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
26. Hernandez-Leal, P., Rosman, B., Taylor, M. E., Sucar, L. E., Munoz de Cote, E.(2016). A Bayesian approach for learning and tracking switching, non-stationary opponents (extended abstract). In *Proceedings of 15th international conference on autonomous agents and multiagent systems* (pp. 1315–1316). Singapore.
27. 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 *Multiagent interaction without prior coordination workshop at AAAI*. Phoenix, AZ
28. 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.
29. Hido, S., Idé, T., Kashima, H., Kubo, H., Matsuzawa, H. (2008). Unsupervised change analysis using supervised learning. In *Advances in knowledge discovery and data mining* (pp. 148–159). Berlin: Springer.
30. Ketter, W., Collins, J., & Reddy, P. P. (2013). Power TAC: A competitive economic simulation of the smart grid. *Energy Economics*, *39*, 262–270.
31. Ketter, W., Collins, J., Reddy, P. P., & Weerdt, M. D. (2014). *The 2014 power trading agent competition*. Rotterdam: Department of Decision and Information Sciencies, Erasmus University.
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. Nash, J. F. (1950). Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, *36*(1), 48–49.
35. Nudelman, E., Wortman, J., Shoham, Y., Leyton-Brown, K. (2004). Run the GAMUT: A comprehensive approach to evaluating game-theoretic algorithms. In *Proceedings of the 3rd international conference on autonomous agents and multiagent systems* (pp. 880–887). New York, NY.
36. Papadimitriou, C. H., & Tsitsiklis, J. N. (1987). The complexity of Markov decision processes. *Mathematics of Operations Research*, *12*(3), 441–450.
37. Powers, R., & Shoham, Y. (2005). Learning against opponents with bounded memory. In *Proceedings of the 19th international joint conference on artificial intelligence* (pp. 817–822). Edinburg: Morgan Kaufmann Publishers Inc.

38. Puterman, M. (1994). *Markov decision processes: Discrete stochastic dynamic programming*. New York: Wiley.
39. Rosman, B., Hawasly, M., Ramamoorthy, S. (2016). Bayesian policy reuse. *Machine Learning, 104*(1), 99–127.
40. Taylor, M. E., & Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research*, *10*, 1633–1685.
41. Tesauro, G., & Bredin, J. L. (2002). Strategic sequential bidding in auctions using dynamic programming. In Proceedings of the 1st international conference on autonomous agents and multiagent systems (p. 591). Bologna: ACM Request Permissions.
42. Urieli, D., & Stone, P. (2014). TacTex 13: A champion adaptive power trading agent. In *Proceedings of the twenty-eighth conference on artificial intelligence* (pp. 465–471). Quebec.
43. Watkins, C., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*, 279–292.
44. Widmer, G., & Kubat, M. (1996). Learning in the presence of concept drift and hidden contexts. *Machine Learning*, *23*(1), 69–101.
45. Wilson, E. B. (1927). Probable inference, the law of succesion, and statistical inference. *Journal of the American Statistical Association*, *22*(158), 209–212.
46. Wurman, P. R., Walsh, W. E., & Wellman, M. (1998). Flexible double auctions for electronic commerce: Theory and implementation. *Decision Support Systems*, *24*(1), 17–27.
47. Yamada, M., Kimura, A., Naya, F., Sawada, H. (2013). Change-point detection with feature selection in high-dimensional time-series data. In *Proceedings of the 23rd international joint conference on artificial intelligence* (pp. 1827–1833). Bellevue, Washington.