MB-AIM-FSI : A Model Based Framework for exploiting gradient ascent MultiAgent Learners in Strategic Interactions

Doran Chakraborty
chakrado@cs.utexas.edu
Computer Sciences Department
University of Texas, Austin
Austin, Texas, USA

Sandip Sen
sandip@utulsa.edu
Mathematical & Computer Sciences Department
University of Tulsa
Tulsa, Oklahoma, USA

ABSTRACT

Future agent applications will increasingly represent human users autonomously or semi-autonomously in strategic interactions with similar entities. Hence, there is a growing need to develop algorithmic approaches that can learn to recognize commonalities in opponent strategies and exploit such commonalities to improve strategic response. Recently a framework [9] has been proposed that aims for *targeted optimality* against a set of finite memory opponents. We propose an approach that aims for targeted optimality against the set of all possible multiagent learning algorithms that perform gradient search to select a single stage Nash Equilibria of a repeated game. Such opponents induce a Markov Decision Process as the learning environment and appropriate responses to such environments are learned by assuming a generative model of the environment. In the absence of a generative model, we present a framework, *MB-AIM-FSI*, that models the opponent online based on interactions, solves the model off-line when sufficient information has been gathered, stores the strategy in the repository and finally uses it judiciously when playing against the same or similar opponent at a later time.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*

General Terms

Algorithms, Performance

Keywords

strategic interactions, multiagent learning

1. INTRODUCTION

Environments involving agents in strategic interactions are becoming increasingly prevalent. The problem of decentralized learning in strategic interactions occurs frequently in practice including areas such as electronic commerce and information technology. Learning in decentralized domains involving multiple agents is difficult because of the inherent non-stationarity of the environment imposed by other agents. Multiagent Learning (MAL) algorithms [1, 3, 4, 7, 10] can be used by agents to guarantee convergence to a fixed point in the strategy space. Most existing MAL algorithms aim at convergence in self-play [4] to the single stage Nash equilibrium played repeatedly as the solution concept. These algorithms strongly rely on self-play and therefore exhibit no guarantee against other opponents and especially those that are adaptive or non-stationary [6]. Some new criterion for MAL algorithms was proposed recently which emphasizes on *auto-compatibility*, *targeted optimality* and *safety* [9].

In this work we focus on *targeted optimality* against a set of MAL algorithms as opponents which aim at convergence to a single stage Nash Equilibrium in a repeated game (RG). For most learners [1, 3, 10], the dynamics are Markovian with the next step strategy depending completely on the joint-strategy in the previous time step [2]. By modeling the underlying MDP as an *A*dversary *I*nduced *M*DP (AIM) and solving the AIM, a learning agent can respond optimally to all such opponents. We begin by assuming the presence of a *generative model* for the underlying AIM but later propose a more general framework (*MB-AIM-FSI*) for interaction for more realistic setting that builds an opponent behavior model from online interactions, solves it offline when it has sufficient information about the model, and uses the corresponding derived strategy during further interactions. We validate the effectiveness of such targeted reuse of learned knowledge using experiments over all 57 structurally distinct 2×2 games [5] against popular MAL algorithms that perform gradient ascent in their strategy space. Our approach generates stable strategies and better average reward than other relevant benchmark approaches under similar settings.

2. BACKGROUND AND DEFINITIONS

In this section we introduce the definitions and concepts necessary for our work. We use bimatrix stage games for our analysis and experiments. A bimatrix game can be represented by a pair of matrices $\{M_i, M_j\}$ where each $M_{k|k\in\{i,j\}}$ is of size $|A_i| \times |A_j|$ and $M_k : A_i \times A_j \mapsto \Re$ maps every possible joint-action to a payoff for agent k. A_i and A_j are the corresponding set of actions available to agents i and j respectively. A strategy π_i of agent i is a probability distribution over i's action set, i.e., $\pi_i \in \Delta A_i$. The probability of

Cite as: Title, Author(s), *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16., 2008, Estoril, Portugal, pp. XXX-XXX.

playing action j following the policy π_i is given by $\pi_i(j)$. As we consider two action per player bimatrix games, we use π_i to refer to the probability of playing the first action by i. A pure strategy plays deterministically; any other strategy is referred to as a mixed strategy.

Now we briefly review some definitions related to Markov Decision Processes (MDP). An MDP M on a set of states S and with action set $A = \{a_1, \ldots, a_k, \ldots, a_{|A|}\}$ is defined by a transition function P and a reward function R. For each state-action pair (s, a), a next-state distribution $P_{s,a}(s')$ gives the probability of moving to state s' when action a is taken in state s. For each state-action pair (s, a), a reward distribution $R(s, a)$ specifies the probability distribution on a set of real numbers that can be achieved as reward given action a is taken in state s. For our work, we assume that M has a deterministic reward function. A policy π for M is given by $\pi : s \mapsto \Delta A$. A value function for policy π in the infinite horizon setting is given by $V^\pi : s \mapsto \Re$ where $V^\pi(s) = E(\sum_{\tau=1}^{\infty} \gamma^{\tau-1} r_\tau | s, \pi)$ where r_τ is the reward generated at the τ'th time step, and $0 < \gamma < 1$ is the discount factor. An action-value function, which gives an estimate of the utility of an action a at state s for a given policy π, is given by $Q^\pi(s, a) = R(s, a) + \gamma \times E_{s' \sim P_{s,a}(\cdot)} V^\pi(s')$. An optimal policy π^* governing M, which yields the maximum discounted return starting from any state $s \in S$, is given by $\pi^* = \arg\max_\pi V^\pi$. It can be shown that for any MDP there exists a deterministic π^*. A generative model G_M of M is a randomized algorithm that, on input of a state action pair (s, a), outputs $R(s, a)$ and a state s' where s' is randomly sampled according to the associated transition probability function $P_{s,a}(\cdot)$. A generative model obviates the critical issue of exploration in settings where experiences are irreversible. Generative models can be used as offline virtual substitute of the real environment, allowing agents the opportunity to learn how to adapt in the virtual environment.

3. ADVERSARY INDUCED MARKOV DECISION PROCESS (AIM)

For RGs, the dynamics of the system can often be modeled as an MDP whose transition probabilities and reward functions are determined by the model of the opponent. Hence the name *A*dversary *I*nduced *M*DP (AIM). Agent i's action set and strategy is given by A_i and π_i respectively while the opponent o's action set and strategy is given by A_o and π_o. For a state s_t of the play, the next state s_{t+1} and the reward received by i can be determined by the current state s_t, $\pi_o(s_t)$, and the action a_i chosen by the agent i. We aim at developing a universal approach that plays effectively against opponents selected from a restricted class of popular MAL algorithms. We focus on the class of MAL algorithms which compute the strategy at a particular time step based on the joint strategy played in the previous time step. If the strategies of such players are observable, their play can be modeled as an AIM. The assumption of agents observing the opponent's previous time step strategies is quite practical in such settings and the convergence guarantees of most of these algorithms to Nash Equilibrium in self-play hinges on this assumption [1, 3, 10].

We now formally present the AIM model ($\mathcal{M}$) for the above setting. The state space $\mathcal{S}$ of $\mathcal{M}$ is given by $\{0, 1\} \times \Delta A_o$ (Note, i plays deterministic strategies). The action space $\mathcal{A}$ of $\mathcal{M}$ is A_i. The transition probability of $\mathcal{M}$ is given by $\mathcal{P}_{s,a_i}(\cdot)$ where $\mathcal{P}_{<\pi_i^t, \pi_o^t>, a_i}(< \pi_i^{t+1}, \pi_o^{t+1} >) = 1$ if $\mathcal{D}(< \pi_i^t, \pi_o^t >) = \pi_o^{t+1}$ and $\pi_i^{t+1}(a_i) = 1$, or 0 otherwise. $a_i \in A_i$ is the action played by i , $\pi_i^t \in \{0, 1\}$, $\pi_o^t, \pi_o^{t+1} \in \Delta A_o$ and $\mathcal{D} : \{0, 1\} \times \Delta A_o \mapsto \Delta A_o$ is the decision function used by the opponent MAL algorithm in deciding the strategy at step $t + 1$. The reward function is given by $\mathcal{R}(< \pi_i^t, \pi_o^t > , a_i) = E_{a_o \sim \mathcal{D}(<\pi_i^t, \pi_o^t>)} M_i(a_i, a_o)$. The reason why $\mathcal{R}$ is an expectation over o's strategy and not the true $M_i(a_i, a_o)$ obtained as reward at step t is explained in next section.

4. REINFORCEMENT LEARNING IN JOINT STRATEGY STATE SPACE

The problem of computing the optimal policy governing $\mathcal{M}$ boils down to a reinforcement learning problem of computing the optimal policy in continuous joint-strategy state space. We assume the availability of a generative model $\mathcal{G}_\mathcal{M}$ governing $\mathcal{M}$ which the learner can use for computing efficient policies against the opponent. We use a linear gradient-descent version of Watkin's Q(λ) with Radial Basis function (RBF) approximation method [11] for encoding features to compute Q-values over the continuous state space. An RBF (representing a feature i) is represented by a Gaussian function g_i with mean μ_i and standard deviation σ_i. So the value of the feature at a particular point $s \in \mathcal{S}$ can be computed using the following function $g_i(s) = e^{-\frac{||s-\mu_i||^2}{2\sigma_i^2}}$. We use the Euclidean distance metric in computing the norm value in the above equation. An important property of RBF encoding of features is that the function generated is non-linear, continuous and differentiable over the state space which is always the case for Q-values. The Q-value of a state s for action a at time t is computed as $Q_t(s, a) = \sum_{i=1}^{F} w_i^t g_i(s)$, where F is the number of features, w_i^t is the weight associated with feature i at iteration t. We use six features with μ values respectively being (0, 0.25), (0, 0.50), (0, 0.75), (1, 0.25), (1, 0.50) and (1, 0.75) with $\sigma = 0.3$. The reason for considering features at the boundaries of i's strategies ($\pi_i \in \{1, 0\}$) is because i always plays a deterministic strategy since there always exists a deterministic π_i^* governing $\mathcal{G}_\mathcal{M}$. The learner will also maintain eligibility traces e_i^t for each feature i [11]. Eligibility traces give a measure of the impact that the current update would have on the feature weights. The w_i and e_i updates at each iteration is based on gradient descent on the Temporal Difference (TD) error seen by the learner [11]. The TD error (δ_t) at a particular iteration is given by

$$\delta_t = r_{t+1} + \gamma max_{a' \in \mathcal{A}} Q_t(s_{t+1}, a') - Q_t(s_t, a) \quad (1)$$

Weight vector, $\vec{w_t}$, and eligibility traces, $\vec{e_t}$, are updated as follows:

$$\vec{w_{t+1}} = \vec{w_t} + \alpha \delta_t \vec{e_t} \quad (2)$$

$$\vec{e_{t+1}} = \gamma \lambda \vec{e_t} + \frac{\partial Q_{t+1}(s_{t+1}, a)}{\partial \vec{w_{t+1}}} = \gamma \lambda \vec{e_t} + g(\vec{s_{t+1}}). \quad (3)$$

Note that Equations 1, 2 and 3 require that the learner knows its current state s_{t+1} to make an update at $t + 1$. However, the learner knows about s_{t+1} at time $t+2$ when o reveals its $t+1$ step strategy. Thus the updates are delayed by one time step and the learner can use the expected reward over the true reward in Equation 1 to compute the TD error. We use ϵ-greedy exploration where the player takes a

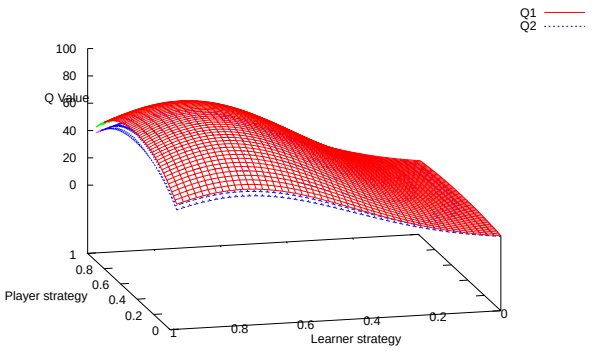

Figure 1: Converged Q-values for IGA in the game given by Table 1(a).

random action with probability ϵ and takes the greedy action, determined by the current Q-values, with probability $1-\epsilon$. Off-policy ($Q(\lambda)$) bootstrapping combined with linear function approximation in continuous state spaces can lead to divergence [11]. We use decaying ϵ and α with time leads to asymptotic convergence of w_is (Q-values) in the limit.

We focus on Gradient Ascent learning [1, 3, 4, 10, 12] opponents. For most learners like the IGA, WoLF-IGA and ReDVaLeR [1], the dynamics are completely Markovian with the next step strategy depending completely on the previous time step joint-strategy and hence they can be exactly modeled and learned using AIM. For others like PHC and WoLF-PHC, however, there are extra factors which makes the process non-Markovian. Now we show how they can be modeled or approximated as Markovian learners.

4.1 Incremental Gradient Ascent Learner (IGA)

IGA [10] converges to Nash equilibrium in self-play for a restricted class of two-action two-player games and for the other games, converges to an average payoff that can be sustained by some Nash equilibrium of the repeated game. For a bimatrix game given by

$$G = \begin{pmatrix} r_{11}, c_{11} & r_{12}, c_{12} \\ r_{21}, c_{21} & r_{22}, c_{22} \end{pmatrix} \quad (4)$$

the decision function $\mathcal{D}$ of an IGA player [1] can be formulated as $\mathcal{D}(<\pi_i^t, \pi_o^t>) = \pi_o^{t+1} = \pi_o^t + \eta \times (\pi_i^t u' - (c_{22} - c_{12}))$ where $u' = (c_{11} + c_{22}) - (c_{12} + c_{21})$, η is the learning rate. u is defined as $(r_{11} + r_{22}) - (r_{12} + r_{21})$. It is evident that the learner's next time step strategy π_o^{t+1} is completely determined by the current step joint strategy (π_i^t, π_o^t). Hence the learner is Markovian and can be modeled using an AIM in the continuous state space $\mathcal{S}$.

Table 1(a) presents a game where $u' = 0$. For the player [2], action A1 strictly dominates action A2. The player learns in episodes spanning 1000 iterations. We initialize ϵ to 0.8 and decay it exponentially over successive iterations. The ϵ value is not refreshed for each episode as the intention for each episode is to learn from the knowledge gathered from

[1]Gradient ascent learner is the column player in all cases.

[2]Hereafter we refer to the AIM learner as *player* and the opponent learning algorithm as *learner*.

(a) Game with $u' = 0$

	$A1$	$A2$
$A1$	(4,0)	(4,0)
$A2$	(0,0)	(0,0)

(b) PD

	$A1$	$A2$
$A1$	(3,3)	(1,4)
$A2$	(4,1)	(1,1)

Table 1: Payoff matrices of two games.

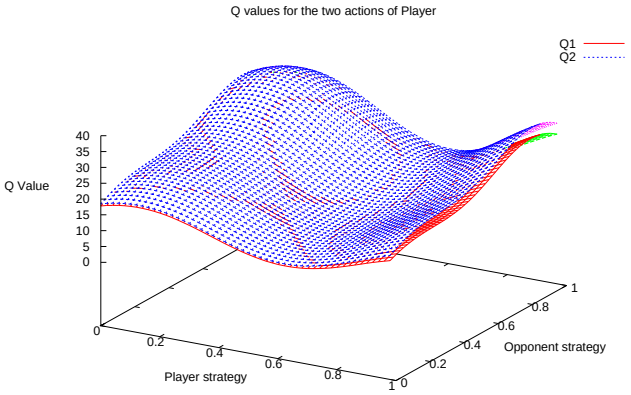

Figure 2: Converged Q-values of IGA in Prisoner's Dilemma

previous episodes. We use η=0.01 for the IGA opponent. We observe that the TD error [3] falls sharply and reaches zero in the third episode of play. The reason for such fast convergence of TD error is because the learner is stationary. As $u' = 0$ and $c_{22} = c_{12} = 0$, $\pi_o^{t+1} = \pi_o^t, \forall t$. The converged Q-values for the two actions of player is shown in Figure 1. It is evident that the player prefers playing A1 over A2 as the Q-value of A1 dominates the Q-value for A2 over the entire state space $\mathcal{S}$. Hence the player has learned to play optimally against the learner.

We next consider the game of PD (Table 1(b)). IGA converges in self-play to the single stage Nash Equilibrium of playing (*A2, A2*) repeatedly. For this game, action *A2* dominates action *A1*. As $u' = 0, c_{22} = 2, c_{12} = 1$, $\mathcal{D}$ for the learner is given by $\mathcal{D}(<\pi_i^t, \pi_o^t>) = \pi_o^{t+1} = \pi_o^t - \eta$. So the update to the learner's strategy is independent of the player's strategy and the probability of playing *A2* increases with each iteration until it converges to 1. The TD error never converges to 0, but oscillates steadily about 0.5 which is enough to ensure that the Q-value of playing *A2* dominates that of playing *A1* over the entire $\mathcal{S}$ (Figure 2 for the converged Q-values in the case of PD). Hereafter we use the term *AIM-play* to refer to the play after the player has learned the AIM.

As our third game of interest, we refer to the popular Chicken game (Table 2(a)). The game has three Nash equilibria: two in pure strategies, sustaining the outcomes (4,2) and (2,4), and one in mixed strategies where the players play each of their actions with equal probability with the corresponding expected payoff of 2.5 for each agent. The two pure strategy Nash equilibriums generate *pareto-optimal*

[3]TD error refers to the on-policy TD error, i.e, the TD error over states visited by the ensuing policy.

outcomes while the outcome generated by the mixed strategy Nash equilibria is pareto dominated by the (3,3) outcome. Note that an outcome is *pareto-optimal* if for all other outcomes one player is better off and the other player worse off compared to this outcome. >From Table 2(a)

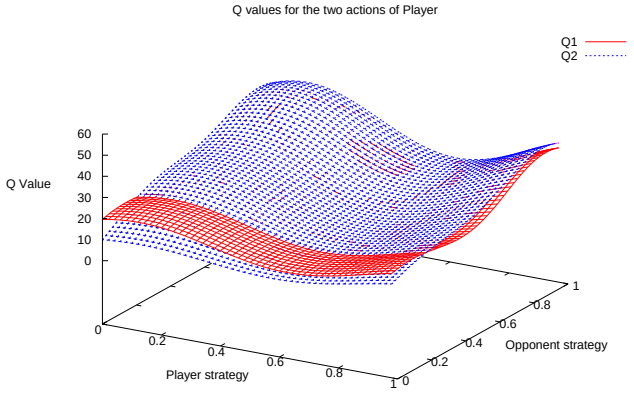

Figure 3: Converged Q-values of IGA in Chicken Game

$u' = -2, c_{22} = 1, c_{12} = 2$, $\mathcal{D}$ for the learner is given by $\mathcal{D}(< \pi_i^t, \pi_o^t >) = \pi_o^{t+1} = \pi_o^t + \eta(-2\pi_i^t + 1)$. The probability of the learner playing A1 increases for $\pi_i^t < 0.5$ and vice versa. Of all the Nash outcomes, the outcome (4,2) is most acceptable to the player. The objective behind AIM learning would be to derive a strategy that would force the learner to play A1 so that the player can keep playing A2 from there onwards, producing a Nash equilibria with an outcome of (4,2). The optimal strategy for the player in this case would be to continually play A2, which will lead the learner to climb the policy space towards playing A1 and thereafter keep repeating that action. Figure 3 shows the converged Q-values for the player. For most of $\mathcal{S}$, A2 is the dominant strategy showing that the player has learned to play the optimal strategy. For a small part of $\mathcal{S}$ ($\pi_o^t < 0.2$), A1 is the dominant action. If the play starts in this part of the state space, it converges to the other Nash equilibrium of (2,4) which is more beneficial to the learner. The loss of optimality in this situation can be attributed to the error in the convergence of w_i's over the episodes.

4.2 WoLFIGA

Bowling and Veloso further extended the convergence properties of IGA using the WoLF principle [3]. They showed that the use of a variable learning rate does in fact guarantee convergence to a unique Nash equilibrium strategy profile in all 2×2 general sum games. The proof follows the convergence proof presented by Singh and colleagues but differs in one special case [3]. Each player selects a Nash equilibrium (no requirement is made that the players select the same Nash equilibrium). If α_e and β_e are the respective halves of the Nash equilibrium that the players select, then the two learning rates used by the learner is given by,

$$\begin{aligned} l_o^t &= l_{min} \; if \; V_o(\pi_i^t, \pi_o^t) > V_o(\pi_i^t, \beta_e) \\ &= l_{max} \text{ otherwise.} \end{aligned}$$

(a) Chicken game ($uu' > 0$)

	$A1$	$A2$
$A1$	(3,3)	(2,4)
$A2$	(4,2)	(1,1)

(b) Matching Pennies game ($uu' < 0$)

	$A1$	$A2$
$A1$	(1,-1)	(-1,1)
$A2$	(-1,1)	(1,-1)

Table 2: Payoff matrices of two games

The decision function for he learner can be stated as $\mathcal{D}(< \pi_i^t, \pi_o^t >) = \pi_o^{t+1} = \pi_o^t + l_o^t \times (\pi_i^t u' - (c_{22} - c_{12}))$ where $l_o^t \in \{l_{min}, l_{max}\}$. As the value of l_o^t is completely dependent on π_i^t, π_o^t and the fixed constant β_e, WoLF-IGA is also completely Markovian and can be modeled as an AIM in $\mathcal{S}$. The behavior of WoLF-IGA is similar to IGA in all the cases discussed above except one. Hence we focus on only this special case when $uu' < 0$ and illustrate this with the game of Matching Pennies (Table 2(b)).

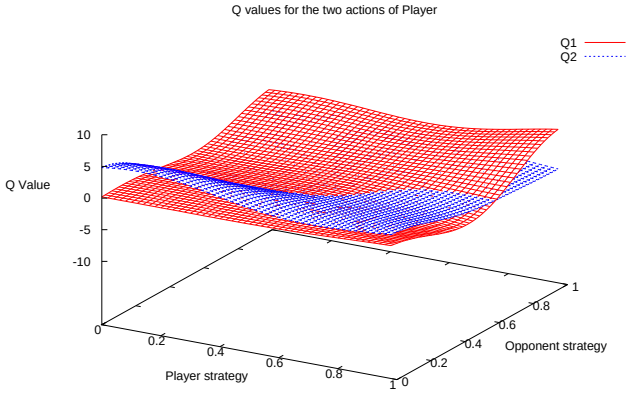

Figure 4: Converged Q-values of WoLF-IGA in Matching Pennies

Figure 4 shows the converged Q-values in this case. The learner has $\mathcal{D}(< \pi_i^t, \pi_o^t >) = \pi_o^{t+1} = \pi_o^t + l_o^t(-4\pi_i^t + 2)$. Let the first random action chosen by the player be A_1 and the initial strategy chosen by the opponent be $\pi_o^1 > 0.5$. The $\mathcal{D}$ function of the opponent prompts the opponent to move towards playing action *A2*. However as soon as $\pi_o^t < 0.5$, the player switches to playing *A2*. The opponent then starts moving towards playing *A1*. Hence the opponent's play converges asymptotically to $\pi_o^t = 0.5$ while the player switches between action *A1* and *A2* in successive iterations. The expected reward for the player thus remains 0. This is also the reward value sustained by the Nash equilibrium which is also the optimal outcome achievable by the player under these conditions. For this special case and against both IGA and WoLF-IGA, the AIM-play does not converge to the Nash equilibrium but maintains an average reward sustained by the Nash equilibrium. It is important to note that the purpose of AIM play is not to converge to an equilibrium but to obtain the optimal outcome for the player. Hence such a result is consistent with the goal of the AIM learner.

We now present another interesting result corroborating the usefulness of using an AIM learner. Figure 5 shows the expected reward achieved in AIM-play and self-play for all

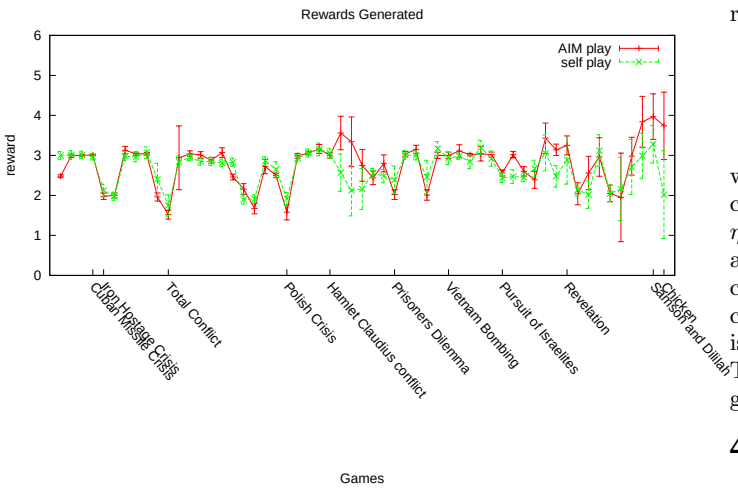

Figure 5: Result of AIM-play and self-play of WoLF-IGA for all 57 structurally distinct 2×2 games.

57 structurally distinct 2×2 games with ordinal payoffs [5]. There are 78 structurally distinct 2×2 strict ordinal games in which the two players can strictly rank the four states from best to worst. Of the 78 games, 21 are no-conflict games where the players mutually agree on the best outcome. We use the other 57 games for evaluation of our approach as these are the more interesting games where the players disagree over the most preferred outcome. Each point has been generated from 100 different instantiations of the starting state (joint strategy). It is evident from the plot, that AIM-play does better than self-play in most occasions as it targets the optimal outcome and not an equilibrium. In particular, AIM-play especially outperforms self-play in the rightmost end of the game spectrum. The reason for this is that the last four games in the spectrum have multiple Nash equilibria. While WoLF-IGA can converge to any Nash equilibrium depending on the starting strategy pair, AIM-play always searches for the best outcome (possibly the most favorable Nash equilibrium to the player). For example, in the case of the Chicken game, self-play has an average payoff close to 2, while in AIM-play, the expected reward is close to 4, which suggests that the player was successful in guiding the play to the *pareto* dominant (4,2) outcome in almost all occasions. The results for IGA are similar in nature and omitted due to space constraints.

4.3 ReDVaLeR

ReDVaLeR[1] was the first algorithm in MAL literature which converges to the Nash equilibrium policy of the one-shot game in repeated self-play and achieves no-regret payoff against any opponent. It uses *Replicator* rule for policy update with a WoLF-like modification and is named ReDVaLeR (*R*eplicator *D*ynamics with a *V*ariable *L*earning *R*ate). Like WoLF, ReDVaLeR too uses a variable learning rate which depends on the mutual Nash equilibrium profile chosen by the players. There is no restriction that the players choose the same Nash equilibrium profile. So if β_e be the Nash equilibrium profile chosen by the learner, the learning rate at each iteration is determined by,

$$\begin{aligned} l_o^t(j) &= 1 \text{ if } \alpha_t \text{ is fixed,} \\ &= 1+\sigma \text{ if } \pi_o^t(j) < \beta_e(j), \\ &= 1-\sigma \text{ if } \pi_o^t(j) \geq \beta_e(j), \end{aligned} \quad (5)$$

where $0 < \sigma << 1$.The decision function of the learner can be stated as $\mathcal{D}(< \pi_i^t, \pi_o^t >)(j) = \pi_o^{t+1}(j) = \pi_o^t(j) + \eta\pi_o^t(j)[l_o^t(j)V_o(j,\pi_i^t) - \sum_k l_o^t(k)\pi_o^t(k)V_o(k,\pi_i^t)]$,where the $l_o^t(k)$'s are determined by Equation (5). It is obvious that π_o^{t+1} is completely dependent on π_i^t , π_o^t, and l_o^t. Since l_o^t's are also completely dependent on π_o^t and the constant β_e, the update is Markovian and hence can be learned using an AIM in $\mathcal{S}$. The results of ReDVaLeR in 57 structurally distinct 2×2 games is similar to that of WoLF-IGA and hence omitted.

4.4 PHC and WoLF-PHC

Bowling et al. proposed a variant of the Q-learning algorithm called Policy Hill Climbing (PHC) that performs hill-climbing on the policy space and can play mixed policies [4]. The decision function of PHC learner can be stated as (Q^o refers to the Q estimates maintained by the learner and should not be confused with Q, which is for the player),

$$\begin{aligned} \mathcal{D}(< \pi_i^t, \pi_o^t >)(j) &= \pi_o^{t+1}(j) = \pi_o^t(j) + \eta \text{ if} \\ &\quad j = argmax_{a' \in A_o} Q^o(a'), \\ &= \pi_o^{t+1}(j) = \pi_o^t(j) - \eta \text{ otherwise.} \end{aligned} \quad (6)$$

The learner uses some exploration to choose the action to be taken at each time step. From Equation (6) it is evident that the process is not Markovian in $\mathcal{S}$ as π_o^{t+1} depends not only on π_o^t but also on the Q^o-values at t time step. For small values of η and due to the continuous and differentiable nature of Q-values, we can assume that the Q-values would be close for $\pi^{t+1} \in \{< \pi_i^{t+1}, \pi_o^t - \eta >, < \pi_i^{t+1}, \pi_o^t + \eta >\}$. For small values of η, the bootstrapping update given by Equation 1 should be largely similar for both possible values of π^{t+1}. Hence the dynamics can be approximated by a Markovian process in $\mathcal{S}$.

The decision function for WoLF-PHC is similar except now the learning rate is given by,

$$\begin{aligned} \eta &= l_{min} \text{ if } \sum_j \pi_o^t(j)Q(j) > \sum_j \hat{\pi_o^t}(j)Q(j), \\ &= l_{max} \text{ otherwise,} \end{aligned}$$

where $\hat{\pi_o^t}$ is the average policy played by the learner until time t. $\hat{\pi_o^t}$ depends on all $\pi_o^{t'}, t' \leq t$. WoLF-PHC is not a Markovian learner due to the same reasons as PHC and additionally due to the dependence on $\hat{\pi_o^t}$. However for small values of l_{min} and l_{max}, it can be approximated by a Markovian learner in $\mathcal{S}$. Note that neither PHC nor WoLF-PHC needs to observe the opponent player's strategy to compute their own strategy and hence they do not also need to disclose their own strategy. In our experiments we have assumed that the player observes the strategies played by the learner in the previous step. However, if they are not visible, then the player can estimate the opponent's past step strategy through a weighted average of the empirical distribution of the opponent's play with more weight given to recent actions to account for changing opponent strategy.

Figure 6 gives the average reward obtained when learners play head to head. The results are based on average reward received by the row player (AIM player) when faced

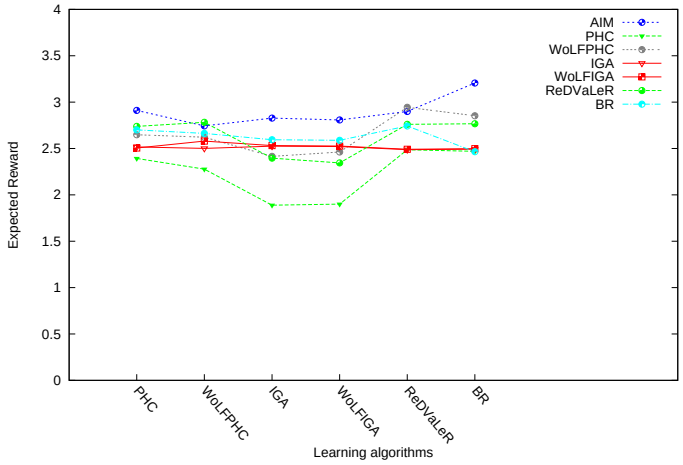

Figure 6: Result of learners playing head-to-head

against a column player which employs one of the learning algorithms discussed in this section. The results have been averaged over all 57 structurally distinct 2×2 games with each game run for 100 different instantiations of initial strategies. We introduce the Best Response (BR) opponent, which deterministically plays the best response to its adversary's last step strategy, for benchmarking. Most of the learners do very well in self play (apart from BR which is not really a MAL algorithm). Some do very well against similar learners, e.g., ReDVaLeR and WoLF-PHC perform well against each other due to both following almost similar mechanisms involving variable learning rate. However AIM learner does better than all other learners against all of these learners. This result highlights the overarching motivation of our work of developing a universal learning scheme that does well against all possible MAL based on gradient ascent approaches.

5. MODEL BASED AIM LEARNING

The learning proposed in Section 4 assumed the presence of a generative model that models the underlying MDP that the player would face given an opponent. The assumption of the existence of a generative model constrains the application of the AIM learning approach in real environments. Learning to play optimally against any practical opponent in real time is, however, infeasible as there can be innumerable opponent types which cannot be meaningfully classified into a finite set of classes. For example, consider the *Tit-for-Tat* and the *Grim* strategies for playing the Prisoner's Dilemma game(Table 1(b)). *A1* and *A2* are commonly referred as *cooperate* and *defect* actions. The *Tit-for-Tat* agent plays the action chosen by the opponent in the previous round. While the *Grim* strategy agent will defect forever if its opponent defected even once. It is impossible to learn to play optimally against both these opponents in one shot as the learner has to *defect* once just to differentiate between them. One defection, however, is sufficient to spoil the learner's chance of achieving the *cooperate-cooperate* payoff against the *grim* strategy opponent. A generative model gives the learner the chance to learn against an opponent offline through repeated episodes of learning. In this section we present a framework that helps the learner to build a model online and then use it to learn a near optimal strategy offline for future play against identical opponents. We now present a motivating scenario where our framework can be applied in practice and then present the framework.

5.1 Model Based AIM Framework For Strategic Interactions (MB-AIM-FSI)

A motivating example for such a scenario is a market with multiple sellers where the buyer is interested to learn an optimal negotiation strategy for buying items. The play between the buyer and seller can be modeled as one of the 57 structurally distinct games discussed before. Chicken game is a very popular market game where the two parties try to pressurize each other until one chickens out. The buyer negotiates in turn with different sellers and learns from these experiences. It is expected that the sellers would have similar negotiation strategies and the learner, through efficient mining of his knowledge of past interactions, can build a set of possible models that the sellers employ in choosing their strategies. These models can then be used later by the buyer to choose effective negotiation strategies in future negotiations. In our case, the sellers choose from a fixed distribution of the set of gradient ascent learning algorithms discussed in the previous section. For a fixed set of runs at the start of each cycle of interaction with a seller (episode), the buyer tries to map the strategy employed by the seller to one of its learned models. If it finds a matching model for its current opponent, the buyer switches to playing the optimal strategy response to that model to maximize its benefit in the ensuing episode.

Algorithm 1 gives an outline of our approach towards learning in setting discussed above. The framework captures learning in a repeated setting where an agent i gets to interact with an opponent $o \sim \mathcal{O}$ at each episode where $\mathcal{O}$ represents a fixed distribution over the possible set of gradient ascent opponents discussed in section 4. *Phase 1* is the exploration phase where i takes exploratory actions and records each possible transition that o makes (*steps 12-13*). Transition in this case is a tuple $< \pi_i^{prev}, \pi_o^{prev}, \pi_o^{curr} >$ representing a decision made by o. Note that the model an agent needs to learn of the opponent is its $\mathcal{D}$ function. In *Phase 2*, i tries to map the recorded transitions to one of its stored models (*step 17*). The method *GetMLModel* gives the maximum likelihood (ML) model that could have generated the transitions. However, if the probability of the ML model fitting the transitions is below a threshold κ, then null is returned. In such a case, a new model is generated (*step 19*) else, the old model is updated with the new set of transitions (*step 21*). Finally if the agent has enough information of the model, it solves it offline using the *SolveModel* method (*steps 22-23*). The *SolveModel* method approximates the $\mathcal{D}$ function of the opponent using an efficient supervised learning algorithm [8] over the stored transition points, e.g., neural networks, Radial Basis functions, k-nearest neighbors, support vector machines, etc. The *SolveModel* method treats the opponent model as an AIM opponent and solves it through the approach discussed in Section 4. In the *Learning /Exploration Phase 3*, i either chooses to exploit or continue learning the model. If the model has already been solved, it chooses to exploit (*step 31*); else it keeps learning (*step 33*). Learning in this case refers to continuing in the learning task for that model

Algorithm 1: MB-AIM-FSI

```
   begin
       input : episodes, runs
1      π_o^prev ← nil
2      π_i^prev ← nil
3      π_o^curr ← nil
4      Models ← nil
5      while episode++ < episodes do
6          o ← O
7          run ← 0
8          model ← nil
9          transitions ← nil
10         while run++< runs do
11             Phase1 (ExplorationPhase) :
12             if run > 1 and run <
               EXPLORATION_STEPS then
13                 transitions ← transitions ∪
14                 < π_i^prev, π_o^prev, π_o^curr >
15             Phase2 (ModelCheckingPhase) :
16             if run = EXPLORATION_STEPS then
17                 model ←
                   GetMLModel(Models, tarnsitions)
18                 if model = null then
19                     model ←
                       GenerateNewModel(transitions)
20                 else
21                     model ← UpdateModel(transitions)
22                     if IsSolvableModel(model) then
23                         model ← SolveModel(model)
24                 Models ← Models ∪ model
25             Phase3 (Learning/ExploitationPhase) :
26             π_o^prev ← π_o^curr
27             π_i^prev ← π_i^curr
28             π_o^curr ← OpponentAlgorithm(π_i^prev, π_o^prev)
29             if run > EXPLORATION_STEPS and
30              IsSolvedModel(model) then
31                 π_i^curr ←
                   GetOptimalAction(model, π_i^prev, π_o^prev)
32             else
33                 π_i^curr ←
                   PlayerAlgorithm(model, π_i^prev, π_o^prev)
   end
```

through the entire episode and gathering more information (transitions) about the model. This entire flow is captured in the *PlayerAlgorithm* and hence its specifics are the same as that of an AIM learner. Note the *Player Algorithm* is an on-line AIM learner leading to good initial Q-estimates for the off-line *SolveModel* method. When it is time to solve the model (*step 23*), the framework just loads all the learned information till date about the model to solve it comprehensively. The *PlayerAlgorithm* implementation captures the important issue of exploitation while exploration in the *Exploration* phase. After sufficient number of episodes when the framework has seen $\forall o \in \mathcal{O}$, random exploration over the $EXPLORATION\_STEPS$ number of steps would be wasteful. Thus the problem boils down to tracking the number of episodes after which a random exploration would be wasteful and then devising a mechanism that exploits well together with generating efficient transitions that help in recognizing the correct model. We tackle the first part of the problem through an update rule given by,

$$\begin{aligned} \lambda_1 &\leftarrow 1 \\ \lambda_{t+1} &\leftarrow \lambda_t + k \times (avgmodelsgen - \lambda_t) \end{aligned}$$

where λ_t is a probability estimate of observing a new model at episode t, $k \in (0:1)$ is a small positive constant, and $avgmodelsgen = \frac{numofmodelsgenerated}{t}$. As $avgmodelsgen$ approaches 0, λ_t tends to 0. The second part of the problem is addressed by choosing an action that is Bayes optimal as per all $m \in Models$. Hence the exploratory action a_τ^t at time step τ of episode t ($\tau < EXPLORATORY\_STEPS$) is chosen using the following rule,

$$\begin{aligned} a_\tau^t &\leftarrow \text{random action with probability } \lambda_t \\ &\leftarrow \text{choose } argmax_{a \in A_i} \frac{\sum_{m \in Models} Pr(m) Q_m^a(\cdot)}{\sum_{a \in A_i} \sum_{m \in Models} Pr(m) Q_m^a(\cdot)} \\ &\quad \text{with probability } (1 - \lambda_t). \end{aligned}$$

$Pr(m)$ is estimated as the number of visits to m until episode t. The inputs to the Q_m^a function for iteration τ are the past step joint strategy, i.e., $< \pi_i^{\tau-1}, \pi_o^{\tau-1} >$.

5.2 Results

We experimented with a configuration where $\mathcal{O}$ consisted of one instantiation each of WoLF-IGA, WoLF-PHC, and ReDValeR. Note, $\mathcal{O}$ restricts the choice of a seller strategy in the example scenario discussed in Section 5.1. The learning rates of WoLF-IGA and WoLF-PHC (l_{max} and l_{min}) were set at 0.01 and 0.005 respectively. For ReDVaLeR, both η and σ was set at 0.1. The number of episodes of interactions were set at 8000 with each episode spanning 100 runs. At each episode, o is drawn randomly from $\mathcal{O}$. To implement the *SolveModel* method, a variation of the k-nearest neighbor [8] algorithm was used. The joint-strategy state space $\mathcal{S}$ was divided into eight zones (four zones each for $\pi_i^{prev} = 1$ and 0 respectively, and for each π_i^{prev}, π_o^{prev} varied in the intervals [0:0.25), [0.25, 0.50), [0.50, 0.75), [0.75, 1.00]). Each transition is mapped into one of the intervals and is stored if the density of the zone is below a fixed threshold and is ignored otherwise. A model is assumed to be fully specified and ready to be solved offline when all the zones for the model have the desired density d. An unknown transition can then be predicted by mapping it into one of the zones and taking the weighted majority using the L2 norm of all the points in the zone. This is a fast efficient representation of the learned model which has a polynomial space complexity in $\frac{1}{d}$ and the number of zones. The prediction has a run time complexity of $O(\frac{1}{d})$.

Henceforth we would refer to the components of $\mathcal{O}$ as domain level learners. Figure 7 shows a comparative study of the average reward received over all the 57 structurally distinct 2×2 games of the MB-AIM-FSI framework with respect to the domain level learners as the default strategy for player i. From the plot it is evident that *MB-AIM-FSI* framework generates higher average reward in most cases w.r.t the domain level learners. What is more interesting is that the *MB-AIM-FSI* framework has lower variation of average reward over all the 57 games while the domain level learners show a high fluctuation over the entire spectrum. This observation corroborates the claim that the MB-AIM-FSI framework is successful in modeling and identifying a

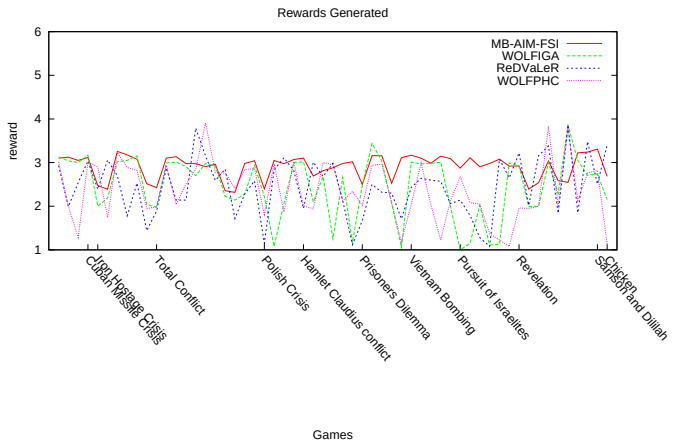

Figure 7: Result against domain-level opponents.

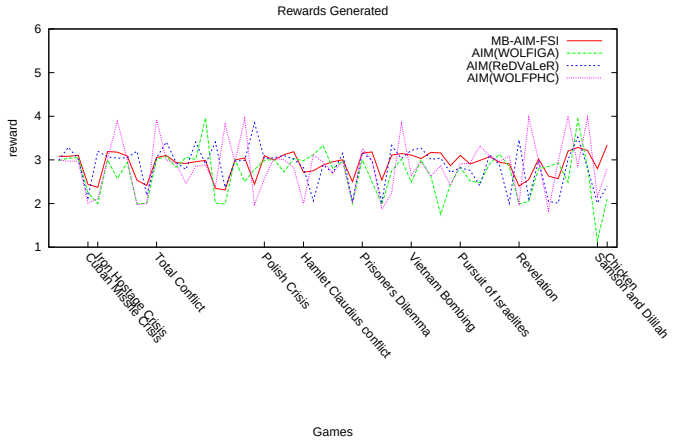

Figure 8: Result against AIM learners trained against domain level opponents.

variety of opponents and derive appropriate responses for a wide range of games. Figure 8 presents a comparative study of the average reward received over all the 57 structurally distinct 2×2 games by the *MB-AIM-FSI* framework in comparison to the strategies learnt by an AIM learner against a generative model of each of the domain level learners (3 benchmarks each being an AIM learned strategy against one of the three domain level learners). Once again we observe that the framework generates stable strategies over the entire spectrum. The AIM strategies show a high fluctuation, the reason being they are strategies adaptive to one of the components of $\mathcal{O}$ and hence have little guarantees of producing high average rewards consistently. However, in some cases they generate high average reward because fortuitously the approximation is an effective one. We believe for larger dimensional games, the chances of such accidental high reward would decrease significantly.

6. FUTURE WORK AND CONCLUSIONS

We developed a technique for playing optimally against a class of MAL algorithms that perform gradient ascent in their respective strategy spaces. We evaluated a number of well-known MAL algorithms and showed how each can be modeled as a Markov Decision Process in the joint strategy space. We then used reinforcement learning using a function approximation scheme (RBF in this case) to respond to such models assuming the availability of a generative model that models the underlying MDP. We then presented a framework that eliminates the need of a generative model and instead builds a model of the opponent online and which it uses later to compute an optimal strategy for future plays. We analyzed our approach on a market situation where sellers adopt one of several known MAL algorithms. Our proposed MB-AIM-FSI framework was able to learn models from interacting with such sellers, develop appropriate responses and later was able to use such responses gainfully by correctly identifying the seller strategies in future interactions. For our future work we would like to run our framework in larger dimensional games and with larger set of opponents. Also it would be interesting to map opponents as $< game, learner >$ pairs. A learner in one game may have similar behavior with another learner in another game. A strategic player which learns this commonality can transfer learning between the above games.

Acknowledgment: A DOD-ARO Grant #W911NF-05-1-0285 partially supported this work. We would also like to thank Peter Stone for his valuable feedback.

7. REFERENCES

[1] B. Banerjee and J. Peng. A unifying approach to performance and convergence in online multiagent learning. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 798–800, New York, NY, USA, 2006. ACM Press.

[2] M. Bowling. Convergence and no-regret in multiagent learning. In *Neural Information Processing Systems 17*. MIT Press, 2005.

[3] M. Bowling and M. Veloso. Convergence of gradient dynamics with a variable learning rate. In *Proc. 18th International Conf. on Machine Learning*, pages 27–34. Morgan Kaufmann, San Francisco, CA, 2001.

[4] M. H. Bowling and M. M. Veloso. Rational and convergent learning in stochastic games. In *IJCAI*, pages 1021–1026, 2001.

[5] S. J. Brams. *Theory of Moves*. Cambridge University Press, Cambridge: UK, 1994.

[6] Y. Chang and L. Kaelbling. Playing is believing: the role of beliefs in multi-agent learning. In *NIPS-2001*, 2001.

[7] V. Conitzer and T. Sandholm. Awesome: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. pages 83–90, 2003.

[8] T. Mitchell. *MachineLearning*. McGraw Hill, 1997.

[9] R. Powers and Y. Shoham. Learning against opponents with bounded memory. In *IJCAI*, pages 817–822, 2005.

[10] S. Singh, M. Kearns, and Y. Mansour. Nash convergence of gradient dynamics in general-sum games. pages 541–548.

[11] R. S. Sutton and A. G. Barto. In *Reinforcement Learning*. MIT Press, 1998.

[12] C. J. C. H. Watkins and P. D. Dayan. Q-learning. *Machine Learning*, 3:279 – 292, 1992.