Strategic Foresighted Learning in Competitive Multi-Agent Games

P.J. 't Hoen and **S.M. Bohte** and **and J.A. La Poutré**[1]

Abstract. We describe a generalized Q-learning type algorithm for reinforcement learning in *competitive* multi-agent games. We make the observation that in a competitive setting with adaptive agents an agent's actions will (likely) result in changes in the opponents policies. In addition to accounting for the estimated policies of the opponents, our algorithm also adjusts these future opponent policies by incorporating estimates of how the opponents change their policy as a reaction to ones own actions. We present results showing that agents that learn with this algorithm can successfully achieve high reward in competitive multi-agent games where myopic self-interested behavior conflicts with the long term individual interests of the players. We show that this approach successfully scales for multi-agent games of various sizes, in particular to the social dilemma type problems: from the small iterated Prisoner's Dilemma, to larger settings akin to Harding's Tragedy of the Commons. Thus, our multi-agent reinforcement algorithm is foresighted enough to correctly anticipate future rewards in the important problem class of social dilemmas, without having to resort to negotiation-like protocols or precoded strategies.

1 Introduction

More and more opportunities are arising where smart software agents can perform useful tasks for their owners. With these agents representing individual stakeholders, they will naturally strive to act such as to maximize the extracted utility for the stakeholder.

It is well known however that in many cases, greedy, selfish behavior by individual agents may harm the reward gained by a collective of agents as a whole, and importantly, ultimately decrease the payoff of the individual agents. This class of problems is known as 'social dilemmas', with the classic iterated Prisoner's Dilemma [15] being an example in the smallest – two player – setting, and Harding's Tragedy of the Commons [6] embodying the same problem instantiated with many players. In both cases the most valuable immediate action that a selfish agent can take lowers the value extracted by all agents collectively. The selfish agent however reaps a higher immediate reward. When all agents take this immediately most rewarding action, the resulting collective joint action has an individual payoff that is (much) worse *for each individual agent* than some other joint actions. Thus, in repeated play of these games, myopic selfish behavior leads for all individuals to (very) poor outcomes. The challenge is to design adaptive algorithms for individual agents that are smart and foresighted enough to allow them to quickly learn how to extract the most individual utility in repeated play of these deceptive competitive multi-agent games.

Recent work in Multi-Agent Reinforcement Learning (MARL) has proposed a number of extensions from the single agent setting to the competitive multi-agent domain (see [3, 10, 13] for an overview). State-of-the-art MARL algorithms improve on single-agent RL approaches by incorporating models of the current opponent agents' behavior. This allows an agent to model the current "environment" including opponents, against which it has to optimize its own behavior by playing a best-response. Unfortunately, this approach encourages detrimental outcomes for social-dilemma type games.

We observe that an important aspect of competitive multi-agent games is largely ignored in MARL algorithms: the policy that maximizes payoff will typically depend on the *changing* actions of adversaries. Importantly, the adaptations of adversaries are likely to be reactions to *ones own* actions. Strategic reasoning about the expected adaptations of the adversaries can be paramount when an agent is repeatedly interacting with the same opponents, and earlier actions influence the future behavior of these adapting adversaries. As noted above, ignoring the consequences of ones actions on the development of the opponent policies can lead to detrimental outcomes, especially for settings with a social dilemma at its core.

In general, social dilemmas can be modeled by the generalized n-player iterated Prisoner's Dilemma, or nIPD [15]. In the nIPD game, every agent has a unilateral incentive to defect. However, if all agents follow this policy, then the individual reward received is lower than if all agents cooperated.

The two-player Prisoner's Dilemma (iPD) is illustrative in that in repeated play the optimal policy differs from playing the myopic best response, and it can be profitable to have a good estimation of the opponents' reactions to ones own play. In a single game, two competing agents can each choose from actions 'Cooperate' or 'Defect' (C or D), and the maximum *joint* payoff is achieved when both agents choose to Cooperate. However, an individual agent obtains a larger payoff by playing Defect, provided that the other agent plays Cooperate. From a game-theoretic perspective, playing $\{D, D\}$, the joint action with the lowest joint and individual payoff, is the dominant strategy and a Nash-Equilibrium in the single shot game.

In iterated play of the Prisoner's Dilemma game, there is no longer a clear dominant strategy, as both agents can achieve a higher aggregated *and* individual reward by cooperating and playing the joint action $\{C, C\}$, *provided* each agent has some strategic incentive to rarely unilaterally defect.

It is important to realize that this strategic incentive in Prisoner's Dilemma type games can consist of the predictable counter-moves of opponents: an intelligent adaptive adversary will react to ones own actions. The question thus becomes: which moves should one play to maximize the individual reward given the *dependent* adaptive behavior of the adversary? How will the opponents react to selfish or cooperative actions and what are good moves to play?

Strategies developed by people like Tit-For-Tat for iPD [1] enforce

[1] CWI, The Netherlands Centre for Mathematics and Computer Science, email: hoen@cwi.nl,sbohte@cwi.nl,hlp@cwi.nl

cooperative play with (smart-enough) adversaries by predictably "punishing" a defecting opponent. Defecting as a strategy is discouraged by the likely *negative future* reaction of the opponent (see also the Folk Theorem discussed in Section 2). For larger and/or less predictable games, such fixed strategies may not exist (or may be hard to determine). It may however still be beneficial for an agent to reason about the impact of its actions on the aggregated behavior of the other agents, and how this will affect the possible rewards that can be gained in future play.

We argue that current state-of-the-art MARL algorithms (reviewed in for example [13]) do not incorporate a sufficient notion of the longer term impact of their actions on the dynamics of the environment. This environment includes other adaptive agents that react to moves of their opponents as well. For example, current MARL algorithms that play the two-player iPD will universally converge to the worst possible outcome of $\{D, D\}$ when an agent plays against an opponent that uses the same MARL algorithm (self-play) [3].

The one exception we are aware of is the M-Qubed algorithm [3], which successfully learns to play $\{C, C\}$ in self play for the two-player iPD. M-Qubed adaptively switches between playing a best-response strategy and the fixed precoded strategy of playing one action consistently. The problem with using such precoded strategies is that they can be exploited [14], and in general such an approach can be expected to be unsuitable for more complex multi-agent games.

In fact, we are not aware of any state-of-the-art MARL algorithms that scale well to larger, n-player IPD games. Myopic best-response type strategies become woefully inadequate, and no current MARL algorithm has sufficient foresightedness to detect how it's own behavior changes future payoffs due to other agents' adaptive behavior.

In this paper, we make a number of important contributions: first, we refine the foresighted StrOPM MARL framework developed in [14]. This StrOPM framework builds on multi-agent Q-leaning type ideas and includes the crucial element of estimating ones opponents' reactions into the forward reward estimations. The StrOPM algorithm in [14] successfully solves two-player two-action matrix games, including the two-player iPD. Here, we generalize the StrOPM framework such that it can be applied to games with more than two players. We develop an instantiation of this refined and generalized framework that is capable of scaling to larger size nIPD games. To deal with nIPD games, we additionally develop a generalized description of aggregate opponent policies, which we use to circumvent the state-space explosion by enabling a description of the nIPD problem in a much reduced, linear state space. We show that this approach successfully scales the StrOPM framework for the nIPD, and our StrOPM algorithm is shown to achieve a high degree of cooperation (and thus reward) even for the 9-player IPD, which was the maximum feasible game-size that ran within reasonable time. We thus argue that our generalization of the StrOPM framework is foresighted enough to correctly anticipate future rewards in the important problem class of social dilemmas, without having to resort to negotiation-like protocols or precoded strategies.

This result is of particular importance, as the generalized n-player iterated Prisoners Dilemma (nIPD) is a matrix game of particular interest, as noted [6]. Although the familiar 2-player iPD is a special case of the nIPD, it has been argued , that the nIPD is 'qualitatively different' from 2 player iPD, and that '... certain strategies that work well for individuals in the [2-player] iPD fail in large groups" [2, 4, 5]. Even more than the 2-player iPD, the nIPD game is also notoriously hard for existing MARL approaches. Thus, we believe the successful generalization of the StrOPM framework to the nIPD matrix game is an important step forward for MARL algorithms.

2 Agents and Matrix games

We define the assumptions and terminology for multi-agent Reinforcement Learning in iterated play of matrix games. We refer to [2] for some well-known terms from Game Theory (GT) such as a **Nash Equilibrium** (NE), a **dominant** strategy, **pareto optimal** or **deficient** strategies, and **best-response**.

Let S denote the set of states in the game and let A_i denote the set of actions that agent/player i may select in each state $s \in S$. Let $a = (a_1, a_2, \ldots, a_n)$, where $a_i \in A_i$ be a joint action for n agents, and let $A = A_1 \times \cdots \times A_n$ be the set of possible joint actions.

A **strategy (or policy)** for agent i is a probability distribution $\pi_i(\cdot)$ over its actions set A_i. Let $\pi_i(S)$ denote a strategy over all states $s \in S$ and let $\pi_i(s)$ (or π_i) denote a strategy in a single state s. The generalized strategy is a **mixed strategy**, and lists for each state the probability of selecting each available action. A **joint strategy** played by n agents is denoted by $\pi = (\pi_i, \ldots, \pi_n)$. Let a_{-i} and π_{-i} refer to the joint action and strategy of all agents except agent i.

We consider **matrix games**, where the payoff for each agent is defined by a set of matrices $R = \{R_1, \ldots, R_n\}$. Let $R(\pi) = (R_1(\pi), \ldots, R_n(\pi))$ be a vector of expected payoffs when the joint strategy π is played. Also, let $R_i(\pi_i, \pi_{-i})$ be the expected payoff for agent i when it plays strategy π_i and the other agents play π_{-i}. Let $R_i(\begin{bmatrix} a_i \\ a_{-i} \end{bmatrix})$ be the payoff for agent i playing action a_i while the other agents play action a_{-i}.

A **stage game** is a single iteration of a matrix game, and a **repeated game** is the indefinite repetition of the stage game between the same agents. While matrix games do not have state, agents can encode the previous w joint actions taken by the agents as state information, as for example illustrated in [12].

In this work, we assume that an agent can observe its own payoffs as well as the actions taken by all agents in each stage game, but only after the fact. All agents concurrently choose their actions. Adaptation of the agents' policy, i.e. learning as a result of observed opponent behavior, only takes effect in the next stage game. Repeated games are modeled as a series of stage games with the same opponent(s). Each agent then aims to maximize its reward from iterated play of the same matrix game.

2.1 The n-player iterated Prisoner's Dilemma

In the nIPD game, each player has a choice of two operations: either **cooperate** (C) with the other player or **defect** (D). A matrix game can be classified as a nIPD game if it has the following three properties: 1) Each player can choose between playing cooperation (C) and defection (D); 2) The D option is dominant for each player, i.e. each has a better payoff choosing D than C no matter how many of the other players choose C; 3) The dominant D strategies intersect in a deficient equilibrium. In particular, the outcome if all players choose their non-dominant C-strategies is preferable from every player's point of view to the one in which everyone chooses D, but no one is motivated to deviate unilaterally from D. The natural outcome of agents playing myopic Best-Response in nIPD is Defection by all agents, which is stable (a NE) but obviously Pareto-deficient.

The nIPD payoff matrix is shown in Table 1; C_i denotes the reward for cooperating with i cooperators and D_i the reward for defecting with i cooperators and $n - i - 1$ other defectors. The following conditions hold for the respective payoffs are [15]: (1) $D_i > C_i$ for $0 \leq i \leq n-1$; (2) $D_{i+1} > D_i$ and $C_{i+1} > C_i$ for $0 \leq i < n-1$; (3) $C_i > \frac{(D_i + C_{i-1})}{2}$ for $0 \leq i \leq n-1$ (the payoff matrix is symmetric for each player).

		Number of cooperators among the other n-1 players				
		0	1	2		$n-1$
player A	C	C_0	C_1	C_2	...	C_{n-1}
	D	D_0	D_1	D_2	...	D_{n-1}

Table 1. Structured payoffs for the (symmetrical) n-player PD

Most multi-agent learning algorithms to date have focused on an individual agent learning a (myopic) Best Response to the *current* strategies of the other agents. Play between such agents using this approach often converges, and has as a goal to converge, to a one-shot NE. However, a famous result from game theory (the **folk theorem**) suggests that the goal of reaching a one-shot NE may be inappropriate in repeated games.

The folk theorem implies that, in many games, there exists NEs for repeated games, repeated Nash-Equilibria (rNEs), that yield higher individual payoffs to all agents than do one-shot NEs, i.e. the rNE Pareto dominates the NE. Hence, in repeated games, a successful set of agents should learn to play profitable rNEs. However, since many repeated games have an infinite number of rNEs, the folk theorem does little to indicate which one the agents should play. [8] present an algorithm for computing rNEs that satisfies a set of desiderata, but how to learn these strategy online is unknown. Additionally, an agent may have preferences between rNEs and play one above the other, if allowed by its opponents.

3 The StrOPM Framework

Here, we refine the StrOPM framework of [14] and generalize it to apply to games with more than two players. In the StrOPM framework, an agent applies reinforcement learning to a state-based policy as described in Algorithm 1. At each epoch of learning, the agent adapts its policy along the gradient of increasing reward. The gradient of reward is calculated including the expected changing behavior of the opponent, as a reaction of the agent's own actions. The StrOPM algorithm tracks the changes in observed opponent policy, on a state by state basis, over time. It is assumed that these changes in the opponent behavior, at least in part, reflect reaction to actions chosen by the StrOPM algorithm . mThe policy of the StrOPM is then optimized with expected future reactions taken into account.

We describe the StrOPM algorithm from the perspectives of an agent i and its opponents, agents $-i$, we use this notation with subscripts to indicate policy, states, action, etc ...of the two types of agents. E.g., a_i and a_{-i} are actions of agents i and $-i$ respectively.

States. The set of states that an agent i can visit, S_i, fulfills the unichain assumption [11]: One set of "recurrent" class of states. Starting from any state in the class, the probability of visiting all the states in the class is 1. We introduce a **transition function** $T : S_i \times A_i \times A_{-i} \rightarrow S_i$ to return the next state of agent i upon playing a (joint) action from the current state.

Policies. For a state $s \in S_i$, $\pi_i(s)$ is the state-based policy of agent i, and $\pi_{-i}(s)$ is the estimate of the opponent $-i$ policies when agent i is in state s. The opponent policy is estimated online using Exponential Moving Average (EMA). The estimate of $\pi_{-i}(s)$ after observing action a_{-i} is adjusted according, for action a_i:

$$\pi_{-i,t+1}(s)(a_{-i}) = (1 - \alpha_{EMA1})\pi_{-i,t}(s)(a_{-i}) + \alpha_{EMA1}. \quad (1)$$

After this update, the policy $\pi_{-i,t+1}(s)$ is normalized to retain $\pi_{-i}(s)$ as a probability distribution.

We introduce **policy update actions**. A policy update action $pua_i(s)$ dictates whether the likelihood of an action a_i should be increased for state s. Additionally, we introduce the *null* policy update action $pua_{null}(s)$ to indicate that the policy should not be changed.

Let $\pi_i(s)^{pua_i}$ be the policy achieved by applying the policy update action $pua_i(s)$ to $\pi(s)_i$. The policy $\pi(s)_i$ of agent i given $pua_i(s)$ not equal to the null action is updated according to:

$$\pi_{i,t+1}(s)^{pua_i} = (1 - \alpha_{LEARN})\pi_{i,t}(s)(a_i) + \alpha_{LEARN}, \quad (2)$$

where α_{LEARN} is the learning rate. The probabilities $\pi_i(s)(\cdot)$ for actions $a_j \neq a_i$ are then normalized to retain $\pi_i(s)$ as a probability distribution.

We wish to select policy update action $pua_i(s)^*$ in (2) that maximizes a measure of expected future payoff of the changed policy. Below, we explain how to compute this.

We introduce $\xi(\pi_{-i}(s), a_i)$ to estimate the impact of an agent i playing an action a_i on the development of the policies $\pi_{-i}(s)$ of the opponent. This function predicts the change in policy of the opponents upon playing action a_i; i.e.

$$\pi_{-i,t+1}(s) = \xi(\pi_{-i,t}(s), a_i). \quad (3)$$

As a first implementation of this function, we take the approach that the changes in the opponent policy are, at least in part, caused by an agent's own actions. Our StrOPM implementation estimates the change in policy as a continuation of the change in policy from estimation of the opponent policy N epochs in the past, i.e. how was the opponent policy at time $t - N$? This is done for each state: for state s reached after playing action joint action $\begin{bmatrix} a_i \\ a_{-i} \end{bmatrix}$ and $T(s', a_i, a_{-i}) = s$ transitions from the current state s' to s, the change in policy for this new state is estimated as a linear extrapolation of the change in the estimated opponent policy:

$$\xi(\pi_{-i,t}(s), a_i) = \frac{\pi_{-i,t}(s) - \pi_{-i,t-N}(s)}{N} + \pi_{-i,t}(s), \quad (4)$$

where we limit ξ to $[0, 1]$. As states can encode a history of play, we judge how an opponent changes its behavior based on our choice of actions.

Thus estimating the change in opponent policies, the question becomes how to compute the value of proposed policy updates. Here, we make use of the unichain assumption that says we only need to consider loops starting from and returning to the current state to compute this value. The algorithm then updates the policy for the possible loops to increase the expected average reward.

From a state s_0, a single loop $L(s_0)$ is defined as:

$$L(s_0) = s_0, \begin{bmatrix} a_{i,0} \\ a_{-i,0} \end{bmatrix}, s_1, \begin{bmatrix} a_{i,1} \\ a_{-i,1} \end{bmatrix}, s_2, \ldots, s_(n-1), \begin{bmatrix} a_{i,(n-1)} \\ a_{-i,(n-1)} \end{bmatrix}, s_0, \quad (5)$$

for a sequence of joint moves for agent i starting in s_0, going through $s1$, $s2$, ...to s_n and ending again in s_0. Each next state reached is through a joint move; $T(s_j, \begin{bmatrix} a_{i,j} \\ a_{-i,j} \end{bmatrix}) = s_{j+1}$. All intermediate states reached in the loop are unique, and not equal to s_0. For brevity, we denote the $j-th$ state s_j in the sequence by $L(s_0)^j$, the $j-th$ action pair $\begin{bmatrix} a_{i,j} \\ a_{-i,j} \end{bmatrix}$ by $L(s_0)_j$, and $L(s_0)_{j+}$ and $L(s_0)_{j-}$ the components of the $j-th$ action pair: $a_{i,j}$ and $a_{-i,j}$. The length of the sequence $L(s_0)$, $|L(s_0)|$, is equal to n; the number of joint actions played before the considered state is reached again.

Let $Bag(s, n) = \{L(s) | |L(s)| \leq n\}$, i.e. $Bag(s, n)$ are all the loops starting in state s with length of at most n . The probability of a particular loop $L(s)$ occurring, denoted by $Pr(L(s))$ is:

$$Pr(L(s)) = \prod_{0 \leq j < |L|} Pr(L(s)_{j+} | \pi_{i,j}(L(s)^j)) \times Pr(L(s)_{j-} | \pi_{-i,j}(L(s)^j)), \quad (6)$$

where $L(s)_j$ and $L(s)^j$ are the respective elements in the loop as defined above, $\pi_{i,j}$ and $\pi_{-i,j}$ are the respective policies at time j, evolving according to Equation (3), and $Pr(\cdot)$ denotes the probability of a specific transition along the sequence given the respective policies $\pi(s)$.

The expected reward over the possible loops can then be expressed as the weighted expected value of individual loops:

$$E(Bag(s,n)) = \sum_{L(s) \in Bag(s,n)} Pr(L(s)) \times E(L(s)), \quad (7)$$

where $E(L(s))$ is the expected average reward for each joint action of a loop starting in state s:

$$E(L(s)) = \frac{\sum_{0 \leq j < |L(s)|} V(L(s)_j)}{|L(s)|}, \quad (8)$$

where $V(L(s)_j)$ denotes the estimated value of the $j-th$ joint action in the loop, $L(s)_j$, to agent i.

The values $V_i : A_i \times A_{-i} \rightarrow \Re$ learns the value of a joint action to agent i; $V_i(\begin{bmatrix} a_i \\ a_{-i} \end{bmatrix})$ learns $R_i(\begin{bmatrix} a_i \\ a_{-i} \end{bmatrix})$. The function V_i is updated using EMA similar to Equation 1.

Strategic Updating. Let the opponent changes in policy be estimated by $\xi(\pi_{-i}(s), a_i)$, we can then compute the policy update with highest expected payoff:

$$pua_i(s)^* = \max_i E(Bag(s,n))^{\xi}_{pua_i(s)}, \quad (9)$$

where $E(Bag(s,n))^{\xi}_{pua_i(s)}$ denotes the expected average reward for the possible loops of at most length n, starting in state s, after effecting policy update action $pua_i(s)$ and taking $\xi(\pi_{-i}(s), a_i)$ as the estimated opponent adaptation. Our StrOPM algorithm thus obtained is outlined in Algorithm 1.

Algorithm 1 StrOPM

```
Initialize π_i(s), π_{-i}(s) for all s ∈ S_i and V
for all all joint actions. Set the initial
state.
Do in each epoch sequentially for each
agent i in state s:
loop
  calculate the highest valued pua_i(s)*
  using E(Bag(s,n))^ξ_{pua_i(s)} as value for the
  individual policy update actions.
  Update policy π_i(s) using the highest
  valued policy update pua_i(s)* action.
  Play action a_i ∈ A_i based on the chosen
  policy update action pua_i(s)*. StrOPM
  plays action a_i for chosen policy update
  action pua_i(s) = pua(s)* if pua* is not the
  null policy update action. Otherwise
  choose the action according to π_i(s).
  Receive reward R_i([a_i a_{-i}]) for the joint
  action determined by agents −i.
  Update the estimate of the reward for the
  joint action V([a_i a_{-i}]) and the opponent
  policy π_{-i}.
  set the current state to T(s, [a_i; a_{-i}]).
end loop
```

4 Experiments

We demonstrate StrOPM for nIPD games of different sizes: from the standard two-player iPD to nIPD games with up to nine players.

For the two-player iPD, the payoff matrix is shown in Figure 1a, inset. We let each agent encode as states the last joint action played; $\{C, C\}$, $\{C, D\}$ $\{D, C\}$, $\{D, D\}$.

This type of state representation, along with the detailed modeling of each individual opponent policy, cannot be scaled indiscriminately for an increasing number of agents (for n players the number of states per joint action scales as 2^{n-1}). We observe however that for nIPD with more than two players, we can reduce the state space representation: even though the opponent payoff contributions C_{-i} and D_{-i} in Table 1 can be different for individual opponents $j \in -i$, their contribution to the reward gradient is summed, and each agent playing this game only experiences this summed gradient.

Thus, we can choose a simpler state representation where we model the behavior of the aggregated opponents. We choose policy update actions using the estimates of the aggregated opponent behavior, and determine the reward gradient through the expected global change in behaviors of the opponents as follows: For each agent, we encode as states the number of other agents that have cooperated in the last epoch; i.e. for n players there are $n-1$ states that capture whether 0, 1, to $n-1$ other agents cooperated in the last epoch. For each state s, separately for action C and D, each agent maintains an estimate of aggregate opponent policy $\pi_{-i}(s)$. The function ξ in Equation (4) then effectively captures the likelihood of either i) staying in the same state, ii) going to a state with more cooperators, or iii) moving to a state with less cooperators as a consequence of playing either action.

The function V learns the payoff Table 1. As for the two agent case, $\pi_i(s)$ encodes for each state the probability of agent i playing C in that state. A learning rate of 0.01 was used for all the EMA equations of Section 3. The StrOPM algorithm looks back $N = 10$ epochs in Equation 4. Additionally, an agent using StrOPM had a $\epsilon = 0.01$ probability of taking an exploration move in each epoch to ensure all states are sufficiently sampled in play.

StrOPM in the 2-player iPD. We present results of the StrOPM algorithm for self-play in the two-player iPD (Figure 1a, solid line), and for more myopic agents in self-play (Figure 1a, dashed line). The myopic agents used a restricted version of StrOPM, MOPM, that was constructed by using StrOPM and setting ξ to $\xi(\pi_{-i}, a_i) = \pi_{-i}$ (i.e., the agent assumes that its adversaries do not adapt in response to its own actions). MOPM thus mimics the best-response type strategies most state-of-the-art MARL algorithms employ.

As can be seen in Figure 1a, the StrOPM learner in self-play converges to playing the (optimal) Cooperate-Cooperate ($\{C, C\}$) strategy with reward 0.35 (solid line). The full cooperation equilibrium is reached as the StrOPM learner estimates that unilaterally defecting leads to states where more and more defection is expected. Additionally, the full cooperation state is expected to lead to more cooperation. In contrast, agents using MOPM in self-play quickly "learn" that cooperation is risky and move to full defection (dashed line).

A closer study of the state-based policy of the StrOPM players reveals that the agents in self-play exhibit a learned Tit-For-Tat strat-

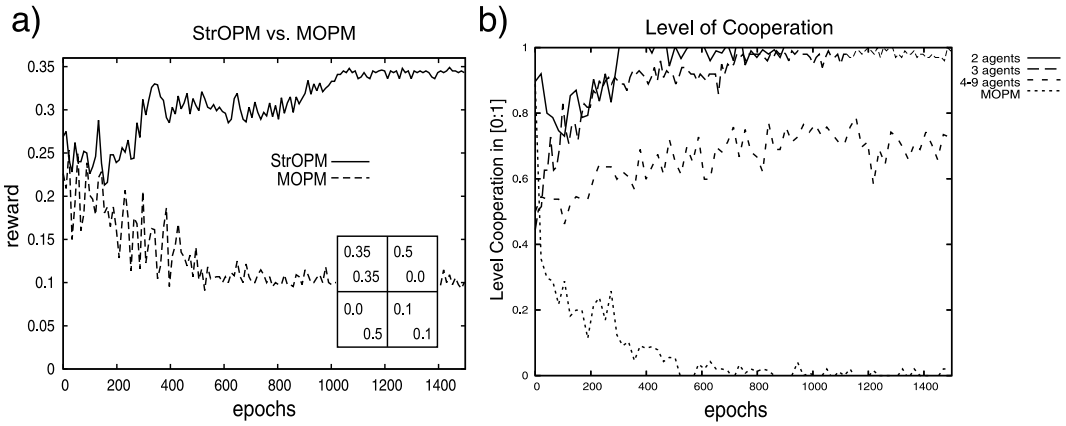

Figure 1. a) Inset: payoff matrix for a two player iPD game. Graph: average payoff for two StrOPM agents and two MOPM agents playing the iPD. b) Cooperation, as % of all agents cooperating, for 3 to 9 players in nIPD using StrOPM.

egy [1]. The Tit-For-Tat algorithm plays C until the opponent defects upon which a D is played followed by again C until the next defection to encourage cooperation. With states encoded as outlined above, the policy is to play C in state $\{C, C\}$, play D from state $\{C, D\}$, and C from $\{D, D\}$. The StrOPM agents learn this strategy and play C to cooperate and reach a good equilibrium, except for occasional exploratory moves. At the same time the StrOPM players also evolve a "threat" state where defection is retaliated in order to guard against exploitation by the opponent. Note that StrOPM also learns to play optimally in less difficult 2-player matrix games, like chicken, rock-paper-scissors, and matching pennies (not shown).

StrOPM playing niPD. The real challenge for playing strategic games like nIPD, is to successfully learn to cooperate with many players. In Figure 1b, we show results for three and more agents playing the nIPD. The x-axis shows the number of epochs learned, while the y-axis shows the average number of cooperators. For three agents, the StrOPM algorithm learns full cooperation. For four up to nine agents, the nIPD is no longer solved in full. The agents as a collective learn to cooperate approximately 70% of the time. Nine-player IPD was the maximum game-size for which we found it feasible to still compute the loops as defined in Equation 5 for the Bags used in the algorithm.

In Figure 1b, bottom line, we show nIPD results for agents using myopic MOPM described above for three agents, to model agents that do not reflect on the impact of their actions; this short-sighted exploitation of the gradient leads to full defection of the agents. This illustrates that as for the two player iPD, classic MARL algorithms that ignore the impact of ones own actions on other agents will, for each state, increase the probability of playing defect, leading to the well know Tragedy of the Commons: universal defection of all agents.

5 Conclusions

We make a number of contributions in this paper: first, we refine the StrOPM MARL framework developed in [14], and generalize it so that is can be applied to matrix games with more than two players. We show that our implementation of this generalized framework successfully scales to larger size nIPD games. To this end, we develop a generalized description of aggregate opponent policies, which allows us to circumvent the state-space explosion and enables us to describe the nIPD problem in a much reduced, linear state space. We show that using this state-space description, our generalized StrOPM algorithm allows an agent to learn profitable strategies for long term behavior in generalized n-player IPD for games with up to nine players. The successful scaling of the framework is of particular interest because, as noted earlier, the n-player iterated Prisoners Dilemma is a game that can represent many important and hard real life problems.

One important novel component of this work is the concept of learning the *aggregate* changes in opponents' policies due to ones own actions. This is an idea that can in fact be incorporated into the quickly growing literature on MARL. One can foresee more and more complex nested opponent models [7] to extract every bit of reward from complex games like nIPD. Although perfectly learning about an opponent while at the same time perfectly learning to adjust oneself is problematic [9], there is still much scope to be "smarter" than your opponents.

REFERENCES

[1] R. Axelrod, *The evolution of cooperation*, Basic Books, New York, NY, 1984.

[2] A.M. Colman, *Game Theory and Experimental Games, The Study of Strategic Interaction*, volume 4 of *International Series in Experimental Psychology*, Pergamon Press, Oxford, 1982.

[3] J. W. Crandall and M. A. Goodrich, 'Learning to compete, compromise, and cooperate in repeated general-sum games', in *Proc. 22nd ICML*, (2005).

[4] N. S. Glance and B. A. Huberman, 'The outbreak of cooperation', *Journal of Mathematical Sociology*, **17**(4), 281–302, (1993).

[5] N. S. Glance and B. A. Huberman, 'Dynamics of social dilemmas', *Scientific American*, 76–81, (1994).

[6] Garrett Hardin, 'The tragedy of the commons', *Science*, **162**, 1243–1248, (1968).

[7] J. Hu and M.P. Wellman, 'Online learning about other agents in a dynamic multiagent system.', in *Proc ACM Conf. on Autonomous Agents*, pp. 239–246, (1998).

[8] Michael L. Littman and Peter Stone, 'A polynomial-time Nash equilibrium algorithm for repeated games', in *Proc. 4th ACM Conf. on Electronic Commerce*, pp. 48–54, (2003).

[9] J. H. Nachbar and W. R. Zame, 'Non-computable strategies and discounted repeated games', *Economic Theory*, **8**, 103– 122, (1996).

[10] R. Powers and Y. Shoham, 'New criteria and a new algorithm for learning in multi-agent systems', in *NIPS*, (2004).

[11] Martin L. Puterman, *Markov Decision Process*, John Wiley and Sons, Inc., New York, 1994.

[12] T. Sandholm and R. Crites, 'Multiagent reinforcement learning in the iterated prisoner's dilemma.', *Biosystems*, **37**, 147–166, (1995).

[13] Yoav Shoham, Robert Powers, and Trond Grenager, 'Multi-agent reinforcement learning: a critical survey', in *AAAI Fall Symposium on Artificial Multi-Agent Learning*, (2004).

[14] P.J. 't Hoen, S.M. Bohte, and J.A. La Poutré, 'Learning from induced changes in opponent (re)actions in multi-agent games', in *AAMAS'06*, (2006). to appear, available as technical rapport SEN-E0513.

[15] Xin Yao and Paul J. Darwen, 'An experimental study of n-person iterated prisoner's dilemma games', in *Evo Workshops*, pp. 90–108, (1994).