Learning to Teach and Follow in Repeated Games

Jacob W. Crandall and **Michael A. Goodrich**
Computer Science Department
Brigham Young University
Provo, UT 84602
crandall, mike@cs.byu.edu

Abstract

The goal of a learning agent playing a repeated game is to maximize its payoffs over time. In repeated games with other learning agents, this often requires that an agent must learn to offer and accept profitable compromises. To do so, past research suggests that agents must implement both *teaching* and *following* strategies. However, few algorithms successfully employ both kinds of strategies simultaneously. In this paper, we present an algorithm (called SPaM) that employs both kinds of strategies simultaneously in 2-player matrix games when the complete game matrix is observable. We show (empirically) that SPaM learns quickly and effectively when associating with a large class of agents, including self, best response learners, and (perhaps) humans.

Introduction

Learning in repeated matrix games has been studied extensively in the literature. Despite this research, "state-of-the-art" learning algorithms continue to learn strategies that produce low payoffs in many simple matrix games, particularly games that model important social dilemmas (e.g. the iterated prisoner's dilemma). This is largely because these learning algorithms are unable to both offer and accept compromises in these games. To be successful in many games, an agent must learn to offer and accept profitable compromises when associating with many different agents.

Past algorithms designed for repeated games generally fall into one of two categories. The first category of algorithms, which we will call *follower* algorithms, adapt their strategies with the goal of finding a best response to the play of their associate(s). This approach allows an agent to accept good compromises offered by associates. However, follower agents are often unable to offer acceptable compromises to others. Consequently, follower agents often earn low average payoffs over time in many important games. The second category of algorithms, which we will call *teacher* algorithms, seek to "teach" their associate(s) to play acceptable (for the teacher) strategies. These agents are often capable of offering acceptable compromises, but are generally unable to accept profitable compromises suggested by others.

In this paper, we present a technique for combining teacher and follower strategies into a single agent/algorithm. We call this algorithm SPaM (*S*ocial and *Pa*yoff *M*aximization).We show that combining follower and teacher attributes as does SPaM results in an agent receiving high average payoffs over time when associating with a large variety of learning agents (including self, best-response learners, and, perhaps, humans) in many important 2-agent matrix games. Prior preliminary research also suggests that SPaM can be implemented effectively in multi-state repeated games (Crandall & Goodrich 2004).

Background and Related Work

As we mentioned in the introduction, past work in multiagent learning for repeated general-sum games can be separated into two categories: follower algorithms and teacher algorithms. In this section, we review some of this work.

Follower Algorithms

Development of the theory of follower algorithms (or adaptive learners (Camerer, Ho, & Chong 2002)) has occupied the energy of most research in multiagent learning. Follower approaches include fictitious play (Fudenberg & Levine 1998), Q-learning (Watkins 1989; Sandholm & Crites 1995; Littman & Stone 2001), Q-learning variants (e.g., (Littman 1994; 2001)), policy gradient methods (e.g., (Bowling & Veloso 2002)), and no-regret algorithms (e.g., (Jafari *et al.* 2001; Bowling 2004)).

The goal of follower algorithms is typically to learn a best response to the assumed stationary strategies of other agents. When all agents play a best response to the strategies of the other agents, the results is a Nash equilibrium (NE). However, achieving this goal has been challenging. Despite some successes, follower algorithms have been mostly unsuccessful at learning non-myopic equilibrium in repeated games.

Teacher Algorithms

Follower algorithms frequently have the characteristic of ignoring the payoffs of other agents in the game. This characteristic is actually necessary in many important contexts since the payoffs (and actions) of other agents may be completely unknown. However, in many other contexts, at least some knowledge about the payoffs of other agents is available. Evidence suggests that humans do use information about the payoffs of other agents when making decisions

(Camerer, Ho, & Chong 2002). Teacher algorithms use this information to coax other agents to play differently.

Perhaps the most famous teacher is tit-for-tat (TFT) (Axelrod 1984), which was designed for the iterated prisoner's dilemma. TFT plays the action played by its associate in the previous time period. The desired result is that TFT's associate learns that cooperation yields higher average payoffs over time than does defection. Thus, TFT receives higher payoffs than it would otherwise.

The economics literature provides several important teacher algorithms (e.g. (Camerer, Ho, & Chong 2002)). Some of the earlier work involved reputation formation (Kreps & Wilson 1982; Milgrom & Roberts 1982). These works show that, in repeated market games, a firm may benefit by establishing a "tough" reputation in early rounds, even though establishing such a reputation may produce low immediate payoffs.

Additional teacher algorithms (called leader algorithms) were given by Littman and Stone (2001) for 2-player matrix games. These teacher algorithms (called *Bully* and *Godfather*) were each shown to induce desirable behavior from certain kinds of associates in many games. Using *Godfather*, Littman and Stone developed an algorithm (which we will call *Godfather++*) for calculating a repeated-play NE for 2-player matrix games (Littman & Stone 2005). We discuss this algorithm in more detail below.

Let player i be the agent in question, and let player $-i$ be its associate[1]. Let, $A_j(s)$ be the actions available to player j in state s, and let $a_j^t \in A_j(s)$ be player j's action at time t (from state s). The *joint-action* $a^t = (a_i^t, a_{-i}^t)$ is a vector of the actions taken by the agents at time t.

Godfather++ calculates a target solution (which may require that the agents alternate between joint-actions) which maximizes the product of the agents' (positive) advantages[2]. Let c_i^t and c_{-i}^t be the target actions (corresponding to the target solution) for players i and $-i$ at time t, respectively. As long as $a_{-i}^t = c_{-i}^t$, player i plays c_i^{t+1} at time $t+1$. However, if $a_{-i}^t \neq c_{-i}^t$, then player i *punishes* player $-i$ for the next n (calculated from the game matrix) rounds of the game by playing an attack strategy. This results in player $-i$ receiving an expected reward of no more than its minimax value each of the n punishment rounds. Thus, an agent's best response to *Godfather++* is to play c_{-i}^t at each time t.

Godfather++ has some very nice theoretical properties, although it does have several issues that should be addressed. Some of these issues include the following:

1 The length of the punishment phase of *Godfather++* (denoted by the variable n) may be unnecessarily large (as Littman and Stone acknowledge). Long punishment phases make it difficult for a learner to determine which of its action is being punished or rewarded (credit assignment problem). Thus, to increase teaching effectivness, the punishment phase should be as short as possible. Additionally, the majority of learning algorithms in the literature typically only condition utilities on very recent action histories, meaning that these agents are unable to "perceive" long punishment phases.

2 *Godfather++* does not specify how the agents should coordinate their actions when the target solution is a sequence of joint-actions. This means that agents may never play cooperatively in some games.

3 The strategy used by *Godfather++* in the punishment phase (i.e., the attack strategy) does not take into account the agent's own payoffs. Thus, vengeance may be overly costly. While this is not extremely problematic (in the limit) if the associate eventually learns to cooperate, there is no guarantee that the associate will learn to cooperate.

4 The target solution is not guaranteed to be unique. Thus, the agents may be unable to coordinate cooperative behavior (particularly in self play). Additionally, a similar (but different) solution may be proposed (and enforced) by an associate, which the algorithm cannot learn to accept.

5 The algorithm requires complete knowledge of the game matrix, as well as perfect knowledge of all agents' actions.

The algorithm we present next addresses points 1-3 (at least in part). We do not address points 4 and 5.

[1]We prefer the term *associate* to *opponent*, since the agents may not be competing.

[2]The advantages of the agents are the agents payoffs minus their minimax value.

An Algorithm for Teaching and Following

Littman and Stone (2001) concluded that neither teacher nor follower strategies alone are sufficient for successful multiagent learning. Rather, agents should employ mixtures of teacher and follower techniques. It is, however, unclear how teacher and follower strategies can be mixed successfully. One methodology is offered by Powers and Shoham (2004; 2005), in which an agent uses a teacher strategy (either *Godfather* or *Bully*) in early rounds of the game. After a deterministic number of iterations, the algorithm evaluates whether or not the associate is responding to the teacher strategy. If it is not, the algorithm switches to a follower strategy. However, their approach does not take into account both teacher and follower strategies simultaneously.

In this paper, we propose a methodology for combining follower and teacher strategies. Although incomplete, the methodology offers a first step towards employing follower and teacher strategies simultaneously. We call this algorithm SPaM (*S*ocial and *Pa*yoff *M*aximizing).

SPaM learns both a teacher and a follower (payoff maximizing) utility function. The teacher utility function estimates how well actions induce "good" behavior from associates. The follower utility function, as always, estimates the actual material payoffs that an action yields.

Let $T(s, a_i)$ and $F(s, a_i)$ be (respectively) the teacher and follower utilities for playing action a from state s. SPaM uses $T(s, a_i)$ and $F(s, a_i)$ to select its actions using the constrained maximization technique shown in Table 1. In words, S is the set of actions that have non-negative teacher utility (if no action has non-negative teacher utility, then S contains only the action with the highest teacher utility). With high probability (i.e., probability $1 - \eta$), SPaM selects the action in the set S with the highest follower utility. With

1. $S = \{a_i : T(s, a_i) \geq 0\} \bigcup \{\text{argmax}_a \, T(s, a_i)\}$
2. Select action a_i^t

$$a_i^t = \begin{cases} \text{argmax}_{a_i \in S} \, F(s, a_i) & \text{with probability } 1 - \eta \\ \text{argmax}_{a_i} \, F(s, a_i) & \text{with probability } \rho\eta \\ \text{random} & \text{with probability } (1 - \rho)\eta \end{cases}$$

Table 1: Action selection in SPaM. In the figure, $T(\cdot)$ and $F(\cdot)$ represent teacher and follower utilities (respectively).

a small probability (i.e., probability $\rho\eta$) SPaM selects the action with the highest follower utility (whether or not it is in the set S). Otherwise, SPaM explores randomly.

We now describe how SPaM learns $T(s, a_i)$ and $F(s, a_i)$ in repeated matrix games.[3]

Learning a Teacher Utility Function

The goal of a *teacher* algorithm is to make player $-i$'s best response desirable for the teacher. This entails that an associate should be a) rewarded for playing "good" actions and b) "punished" (if necessary) for deviating from those "good" actions. For "good" actions to be player $-i$'s best response, the punishment must exceed the profit of deviating.

SPaM, like *Godfather++*, determines that "good" actions contribute to a sequence of joint-actions that maximize the product of the agents' (positive) advantages (as justified by Nash (1950)).[4] Let this sequence of joint-actions, called the *target solution*, be denoted by c and let $c^t = (c_j^t, c_{-j}^t)$ be the joint-action which c specifies at time t. Also, let $|c|$ denote the length of the sequence of c. In games in which $|c| > 1$, we must determine how agents cycle through the sequence of joint-actions. In our current implementation, $c^t = c^{t-1}$ if c^{t-1} is not played at time $t-1$. Otherwise, the current joint-strategy of the target solution updates as in *Godfather++*.[5]

SPaM uses the notion of guilt (denoted G_j^t for player j's guilt at time t) to determine whether an agent should be rewarded or punished. When $G_j^t > 0$, then player j has some level of guilt. When $G_j^t \leq 0$, then player j is *guiltless*. As a rule, a guilty agent should be punished while a guiltless agent should be rewarded. We describe how SPaM determines guilt below.

Let r_j^t be the payoff to agent j at time t. Also, let $r_j(c^t)$ be the *expected* payoff to agent j when the joint-action c^t is played, and let $r_j(c) = \frac{1}{|c|} \sum_{c^t \in c} r_j(c^t)$ be the *average expected* payoff to player j if both agents play the target solution.

[3]In (Crandall & Goodrich 2004), we described how SPaM can be used in two-player repeated multi-state games.

[4]The maximum possible product of the agents' (positive) advantages in competitive games is zero. In 2-player competitive games, a teacher algorithm has nothing to offer. Thus, $T(s, a) = 1$ for all pairs (s, a), meaning that the teacher utility function has no effect on the behavior of the agent (see Table 1). The rest of this section deals with the case in which the maximum product of the agents' advantages is nonzero.

[5]This implementation is sufficient for all games discussed in this paper, but should be modified for the general case.

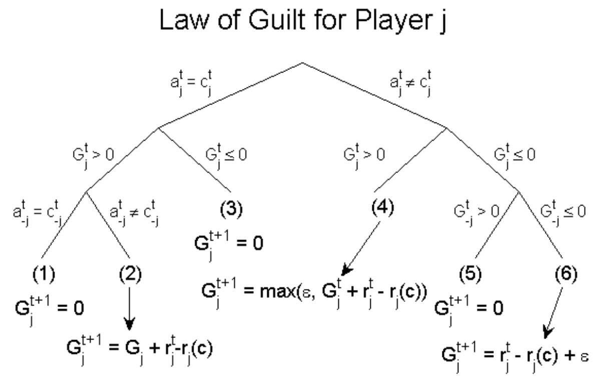

Figure 1: Tree showing how player j's guilt value (G_j) is updated after each round of the game.

Initially, each agent is guiltless (i.e., $G_j^0 = 0$). Thereafter, the guilt of each player j is updated using the tree shown in Figure 1, which specifies six unique cases (labeled 1-6). Each of the six cases is represented by a unique path through the tree. For example, case 1 is the situation in which $a_j^t = c_j^t$, $G_j^t > 0$ and $a_{-j}^t = c_{-j}^t$. We now explain and justify how guilt is updated in each case.

1 The joint-action c^t is played when player j is guilty. Since player j is not being punished (while guilty) in this case, its guilt would normally be maintained. However, in this special case, to avoid the possibility that player j may attribute future punishment to the playing of the target solution, player j's guilt is absolved.[6]

2 When player j is guilty and plays cooperatively, its guilt is updated by $G_j^{t+1} = G_j^t + r_j^t - r_j(c)$. This means that player j's guilt is reduced if its payoff (r_j^t) is less than $r_j(c)$. Note that if $r_j(c)$ is at least G_j^t more than r_j^t, then player j's guilt is absolved.

3 Player j follows the target strategy while guiltless. Thus, it remains guiltless.

4 Player j deviates while guilty. Its guilt is updated as in case 2, except that its guilt cannot be completely absolved (since it deviated). Thus, its new guilt is the max of some small value $\varepsilon > 0$ and $G_j^t + r_j^t - r_j(c)$.

5 Player j deviates when its associate is guilty. In this case, player j is justified in deviating (so it remains guiltless).

6 Player j deviates when its associate is guiltless. In this case, player j acquires guilt to the degree that it profits from the deviation (the difference between r_j^t and $r_j(c)$, plus some small value $\varepsilon > 0$[7]). Thus, if deviating yields a profit less than or equal to -ε, an agent remains guiltless.

Note that SPaM measures its own guilt as well as its associate's guilt. Thus, SPaM's associate can hold SPaM ac-

[6]We note that this rule could possibly (in a small set of games) allow deviating to be a best response. We leave this to future work.

[7]ε is added to player j's guilt to require that player j be punished by *more than* it gains from deviating.

countable in the same way that SPaM holds its associate accountable. In this way, SPaM can (potentially) teach and follow simultaneously.

In short, player j's guilt is the (positive) profit it has obtained from deviating in the past. When guilt is positive, the agent has benefited from deviating and should be punished. Otherwise, the agent should be rewarded (with cooperative actions from its associate). We now specify how a teacher utility function is constructed from the guilt values to obtain the desired behavior.

The action selection methodology in Table 1 requires that teacher actions have positive utility only if they induce "good" behavior from the associate. Thus, if action a does not serve to induce player $-i$ to play c_{-i} in the future, then $T_t(s, a_i) < 0$. If action a does induce player $-i$ to play c_{-i} in the future, then $T_t(s, a_i) > 0$.

In the case in which player i's associate is guiltless, player i should conform to the target solution. Thus, in this case, the teacher utility $T_t(s, a_i)$ is given by

$$T_t(s, a_i) = \begin{cases} 1 & \text{if } (a_i = c_i^t) \\ -1 & \text{otherwise} \end{cases} \qquad (1)$$

In the context of Table 1, Eq. (1) assures that, with at least probability $1 - \eta$, SPaM will play c_i^t when $G_{-i}^t \leq 0$.

We now turn to the case in which $G_{-i}^t > 0$. As we have mentioned, in this case, player i should punish player $-i$. Thus, the teacher utility $T_t(s, a_i)$ is a function of how well action a punishes player $-i$. Thus,

$$T_t(s, a_i) = r_{-i}(c^t) - E[U_{-i}(s, a_i | G_{-i}^t > 0)] - E_p \qquad (2)$$

where $E[U_{-i}(s, a_i | G_{-i}^t > 0)]$ is the expected payoff to player $-i$ based on the empirical distribution of past actions taken by both agents (when player $-i$ is guilty) and E_p is the expected level of punishment.

We make the following observations about Eq. 2. First, $E[U_{-i}(s, a_i | G_{-i}^t > 0)]$ is the value of the teacher utility function that is learned. Second, when the difference $r_{-i}(c^t) - E[U_{-i}(s, a_i | G_{-i}^t > 0)]$ is greater than E_p, then $T_t(s, a_i) > 0$. Third, when multiple actions produce satisfactory punishments, SPaM will select the least costly of those actions (see Table 1). Therefore, SPaM can lower its own loses while punishing its associates (see point 3 of the issues discussed in section 2).

We discussed in section 2 that the punishment phase should be brief (if possible). The variable E_p (the expected punishment level) dictates the duration of the punishment phase. For quick punishments, we use

$$E_p = \min(G_{-i}^t, r_{-i}(c^t) - m_{-i}), \qquad (3)$$

where m_{-i} is player $-i$'s minimax value. We use this value of E_p since a) player $-i$ need not be punished by more than its guilt value G_{-i}^t and b) since player $-i$ can guarantee itself a payoff of m_{-i} by playing its minimax strategy, an agent should not expect to punish its associate by more than $r_{-i}(c^t) - m_{-i}$.

Follower Utility Function

The follower algorithm used by SPaM can be any follower algorithm. However, the weaknesses of the algorithm chosen reflect on the performance of SPaM. In general, we favor a learning algorithm which learns quickly, thus, we use a variant of fictitious play (Fudenberg & Levine 1998) (which we call FP). FP varies from standard fictitious play in that it looks ahead an extra time period (rather than just the current time) in determining the action that maximize expected payoffs. Additionally, the previous joint-action taken by the agents is used for state s (as in the teacher utility function).

```
1) Observe the game matrix; compute $c$
2) $G_i = G_{-i} = 0$, $T(s, a_i) = 0$, $F(s, a_i) = 0$
3) Repeat,
   a) Take action $a_i$ according to Table 1
   b) Observe reward $r$ and actions $a_i^t$, $a_{-i}^t$:
        i)  Update $G_i$ and $G_{-i}$ using Figure 1
        ii)  Updated $U_{-i}(s, a_i | G_{-i} > 0)$
        iii) if $G_{-i} \leq 0$
                compute $T(s, a_i)$ using Eq 1
             else compute $T(s, a_i)$ using Eq 2
        iv) Compute $F(s, a_i)$
```

Table 2: The complete SPaM algorithm.

Algorithm Review

The complete algorithm is sketched in Table 2. As we mentioned in section 2, the algorithm empirically exhibits the following desirable characteristics. First, the length of the punishment phase is minimized provided that $E[U_{-i}(s, a_i | G_{-i}^t > 0)]$ is accurate. As an example, whereas *Godfather++* requires 3 rounds of punishment in the game *battle of the sexes*, SPaM typically requires no rounds of punishment, although one or two rounds of punishment are sometimes necessary. Second, SPaM is able to successfully coordinate profitable cooperative actions even in non-deterministic environments. Third, when punishing an associate, SPaM may select between multiple punishing actions (provided that multiple punishing actions have been shown to be effective). This allows SPaM to select the action that is most profitable to it. SPaM empirically exhibits these properties while maintaining the other positive characteristics of *Godfather++* (it appears).

Results

SPaM performs well when associating with a large variety of agents. We give some of these results below.

With Automated Agents

Figure 3 shows the mean average payoffs when SPaM, FP, and WoLF-PHC (Bowling & Veloso 2002) are matched with each other in the three repeated matrix games show in Figure 2. In each game, the intended actions of all the agents were executed with probability 0.95 and the opposite action was executed with probability 0.05.

Prisoner's Dilemma. Figures 3(a)-(c) show results from the iterated prisoner's dilemma (see Figure 2a). In this game, the target solution is for each agent to play a. Although action a is dominated by action b (for both agents), SPaM is

	a	b
a	3, 3	0, 5
b	5, 0	1, 1

(a)

	a	b
a	4, 4	2, 5
b	5, 2	0, 0

(b)

	a	b
a	0, 3	3, 2
b	1, 0	2, 1

(c)

Figure 2: Payoff matrices for (a) prisoner's dilemma, (b) chicken, and (c) tricky game (Bowling 2004). Payoffs to the row player are given first, followed by the payoffs to the column player.

able to teach its associates to play a, resulting in an average payoff per iteration of nearly 3. In this game, SPaM significantly outperforms FP and WoLF-PHC when matched with each of the three agents.

Chicken. Figures 3(d)-(f) show results from the game chicken (see Figure 2b). In chicken, the target solution is for each agent to play a, despite the fact that each agent has an incentive to unilaterally deviate. Again, SPaM teaches each of the three associates to do so, resulting in an average payoff per iteration of nearly 4. Again, SPaM outperforms FP and WoLF-PHC in this game when playing with each of the three associates.

Tricky Game. Figures 3(g)-(l) show results when the agents are matched with each other in tricky game (see Figure 2c). In tricky game, the target solution is for the row player to play a and the column player to play b, which results in payoffs of 3 and 2 to the row and column players respectively. While the prisoner's dilemma and chicken both generally require only a single round of punishment after a deviation, tricky game frequently requires two rounds of punishment after a deviation. Since our implementations of both FP and WoLF-PHC encode state s as the previous joint-action of the agents, none of the agents are able to "perceive" some of SPaM's punishments. However, SPaM still performs better than both WoLF-PHC and FP when matched with all three associates, both as a row player and a column player. However, its payoffs are not as high as they otherwise would be had each of the agents used longer histories for state.

With Humans

We are working on obtaining official results pairing humans with the three algorithms in various repeated matrix games. Preliminary results suggests that SPaM performs very well when playing many of these games humans. We leave these results to future work.

Discussion and Future Work

We have described some of our work in progress on combining teaching and following strategies using SPaM. While as yet incomplete, results show that SPaM performs well in many 2-player matrix games when matched with a wide variety of agents, include itself, follower algorithms, and (in preliminary results not shown here) humans.

Although promising, SPaM suffers from a number of weaknesses. One of these weaknessses is that it does not perform as well as desired when matched with static agents, since static agents cannot be "taught." One possible method for overcoming this challenge is to increase the parameter η when the associate does not respond to teaching. However, we have not as yet found a way to do this effectively without decreasing average payoffs when matched with agents that learn slowly. Additionally, there remains much progress to be made before these these techniques can be used in practical environments (i.e., multi-state environments, environments with incomplete information, and environments with more than two agents).

References

Axelrod, R. 1984. *The Evolution of Cooperation*. Basic Books.

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

Bowling, M. 2004. Convergence and no-regret in multiagent learning. In *Advances in Neural Information Processing Systems*.

Camerer, C. F.; Ho, T.-H.; and Chong, J.-K. 2002. Sophisticated ewa learning and strategic teaching in repeated games. *Journal of Economic Theory* 104:137–188.

Crandall, J. W., and Goodrich, M. A. 2004. Establishing reputation using social commitment in repeated games. In *AAMAS Workshop on Learning and Evolution in Agent Based Systems*.

Fudenberg, D., and Levine, D. K. 1998. *The Theory of Learning in Games*. The MIT Press.

Jafari, A.; Greenwald, A.; Gondek, D.; and Ercal, G. 2001. On no-regret learning, fictitious play, and nash equilibrium. In *Proceedings of the 18th International Conference on Machine Learning*.

Kreps, D. M., and Wilson, R. 1982. Reputation and imperfect information. *Journal of Economic Theory* 27:253–279.

Littman, M. L., and Stone, P. 2001. Leading best-response strategies in repeated games. In *IJCAI Workshop on Economic Agents, Models, and Mechanisms*.

Littman, M. L., and Stone, P. 2005. A polynomial-time nash equilibrium algorithm for repeated games. *Decision Support Systems* 39:55–66.

Littman, M. L. 1994. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the 11th International Conference on Machine Learning*.

Littman, M. L. 2001. Friend-or-foe: Q-learning in general-sum games. In *Proceedings of the 18th International Conference on Machine Learning*.

Milgrom, P., and Roberts, J. 1982. Predation, reputation and entry deterrence. *Journal of Economic Theory* 27:280–312.

Nash, J. F. 1950. The bargaining problem. *Econometrica* 28:155–162.

Powers, R., and Shoham, Y. 2004. New criteria and a new algorithm for learning in multi-agent systems. In *NIPS*.

Powers, R., and Shoham, Y. 2005. Learning against opponents with bounded memory. In *IJCAI*.

Sandholm, T. W., and Crites, R. H. 1995. Multiagent Reinforcement Learning in the Iterated Prisoner's Dilemma. *Biosystems, Special Issue on the Prisoner's Dilemma*.

Watkins, C. 1989. *Learning from delayed rewards*. Ph.D. Dissertation, University of Cambridge, England.

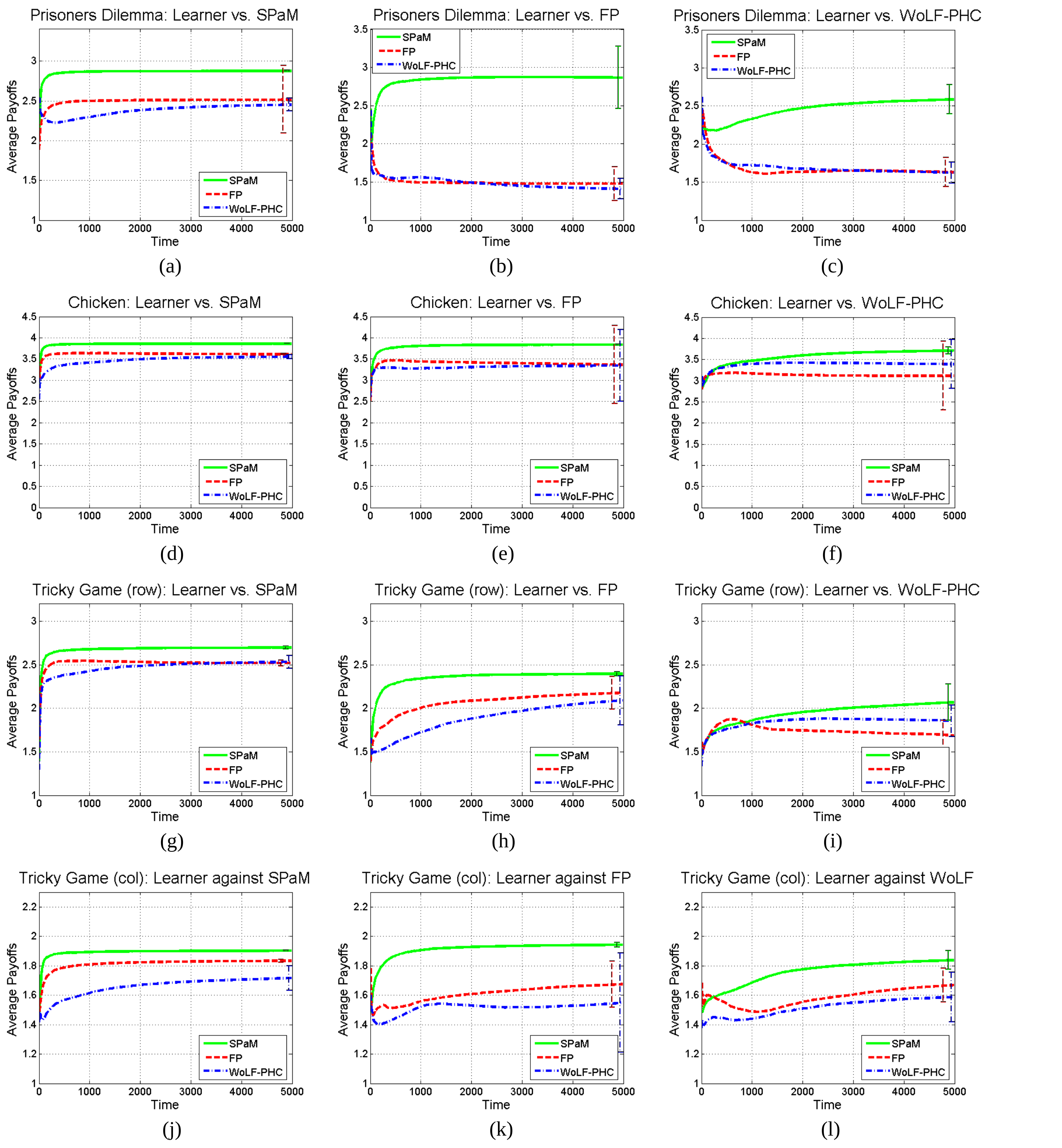

Figure 3: Payoffs given different pairings of artificial agents. The figures plot the mean (of 50 trials) average payoff per iteration against time. One standard deviation is shown after 5000 iterations. (g)-(i) show the payoffs to the row player in tricky game and (j)-(l) show payoffs to the column player. For SPaM, $\eta = 0.1$ and $\rho = 0.8$.