Learning by Demonstration in Repeated Stochastic Games

Jacob W. Crandall
Masdar Institute of Science and Technology
Abu Dhabi, UAE
jcrandall@masdar.ac.ae

Malek H. Altakrori
Masdar Institute of Science and Technology
Abu Dhabi, UAE
maltakrori@masdar.ac.ae

Yomna M. Hassan
Masdar Institute of Science and Technology
Abu Dhabi, UAE
yhassan@masdar.ac.ae

ABSTRACT

Despite high research emphasis over the last few decades, newly created multi-agent learning (MAL) algorithms continue to have one or more fatal weaknesses. These weaknesses include slow learning and convergence rates, failure to learn non-myopic solutions, and inability to learn effectively in domains with many actions, states, and associates. The continued presence of these prohibitive weaknesses in newly developed MAL algorithms suggests a need to identify and develop fundamentally different approaches to MAL. One possibility is to employ humans as teachers of these artificial learners. As a step toward determining the usefulness of this approach, we explore "learning by demonstration" (LbD) in repeated stochastic games, wherein the learning algorithm utilizes intermittent demonstrations from the human teacher to derive a behavioral policy. To do so, we compare and contrast two LbD algorithms in a rich formulation of the iterated prisoners' dilemma.

Categories and Subject Descriptors

H.4 [**Information Systems**]: Miscellaneous

General Terms

Algorithms, performance

Keywords

Multi-agent learning, game theory, learning by demonstration

1. INTRODUCTION

Due to its potential applicability to many real-world systems, multi-agent learning (MAL) in repeated games has become a popular research topic [19]. The goal of much of this research has been to develop algorithms that maximize an agent's payoffs over time. Unfortunately, current MAL algorithms still cannot successfully solve many of the real-world challenges that this research has targeted due to one or more debilitating weaknesses. First, most MAL algorithms learn too slowly to be useful in real-time systems. These algorithms often require thousands of iterations to converge, even in two-agent, two-action games. Second, most MAL algorithms do not "scale-up" to games with many agents, actions, and states [10]. Lastly, most MAL algorithms perform effectively in a restricted class of repeated games against a restricted set of associates, but often do not perform well when these limiting assumptions are not met.

While much work has attempted to overcome these weaknesses, we believe that repeated failures highlight the need for a new approach to MAL. Like a child learning a complex skill, agents learning in repeated games require a (potentially flawed) tutor to help them overcome the complexities of these dynamic environments (Figure 1). People with vested interest in the agent's success should potentially supply intermittent reward reinforcement, demonstrations of successful behavior, and intuition into what might be successful [5].

In this paper, we begin to explore how intermittent interactions between a human teacher and an artificial learning agent will affect the outcome of repeated stochastic games [18]. In particular, we focus on learning by demonstration (LbD) in repeated stochastic games, wherein the human teacher intermittently demonstrates the actions that he or she believes the agent should perform. The agent then uses these demonstrations to improve its behavior over time.

LbD has been studied and applied to many problems, particularly in the robotics domain [1]. Most of this research has pertained to situations in which the human teacher knows successful behavior. However, in repeated games, where information about learning associates, their tendencies, behaviors, and goals, and even the game itself is lacking, the teacher may or may not know how the agent should behave to be successful. Since the teacher will also likely learn throughout the repeated game, demonstrations provided by the human are likely to be noisy and to change with time.

As a step toward determining the potential of LbD in repeated stochastic games, we focus on two related sets of questions. First, what types of LbD algorithms will be successful in repeated stochastic games? These algorithms must quickly learn non-myopic solutions in games with many states and agents. Second, how good do demonstrations need to be for these algorithms to learn successfully? Can the algorithms utilize demonstrations from unformed novices, or do they require more-informed demonstrations? Can LbD algorithms learn effectively when human teachers are initially less-informed, but become more-informed over time?

In this paper, we begin to analyze these questions using a rich formulation of the iterated prisoners' dilemma. In particular, we compare and contrast two LbD algorithms in this game given different qualities of demonstrations. We begin by describing the prisoners' dilemma game.

2. MULTI-STAGE PRISONERS' DILEMMA

We consider the problem of learning in the repeated stochastic game shown in Figure 2(a) [4]. In this game, two players (represented by the circle and the square in the figure) be-

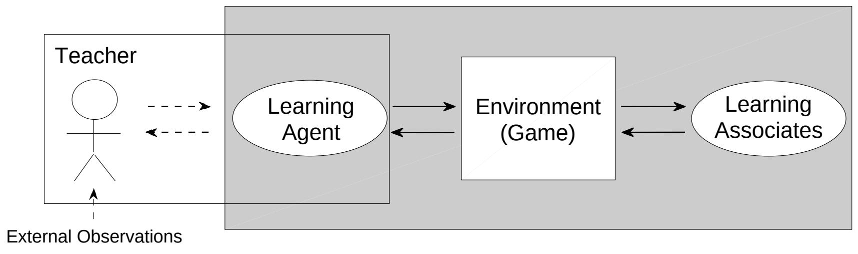
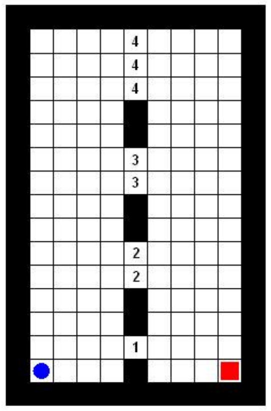


Figure 1: This paper discusses the situation in which a human teacher interacts with a learning agent in a repeated stochastic game. The grey rectangle represents the components of a traditional repeated game. The human teacher uses its knowledge, intuition, and external knowledge to teach the agent how to play the repeated game.



(a)

	Defect (Gate 1)	**Cooperate** (Gates 2, 3, or 4)
Defect (Gate 1)	-25, -25	-10, -32
Cooperate (Gates 2, 3, or 4)	-32, -10	-16, -16

(b)

Figure 2: (a) A multi-stage prisoner's dilemma game. Each agent (blue circle and red square) must cross the middle barrier via one of four gates to reach the other agent's starting position in as few steps as possible. (b) High-level payoff matrix for the MSPD.

gin each round in opposite corners of the world, and seek to move across the world through one of four (initially open) gates to the other player's start position in as few moves as possible. The physics of the game are as follows:

1. From each cell, each player can attempt to move up, down, left, or right. Moves into walls or closed gates result in no change to the player's position. The players move simultaneously; moves are only effectuated after both players have chosen an action.

2. If both players attempt to move through gate 1 at the same time, gates 1 and 2 close and both players are forced to go through gate 3.

3. If player i attempt to enter gate 1 on a move that player $j \neq i$ does not, then player i is allowed passage through gate 1, and gates 1, 2, and 3 close so that player j can only reach its goal through gate 4.

4. If either player moves through gates 2, 3, or 4, then gate 1 closes for the remainder of the round.

5. Each player's score for a round is determined by the number of moves it takes for it to reach its goal. A round is automatically terminated after 40 moves if one of the players has not yet reached its goal.

6. After both players reach their respective goals, the gates are reset (to open) and each of the players is returned to its start position.

7. We assume that the locations of both agents and the current status of the four gates are known to both players at all times.

When a player attempts to move through gate 1, it is said to have *defected*. Otherwise, it is said to have *cooperated*. Viewed in this way, the *high-level* game is the prisoner's dilemma matrix game shown in Figure 2(b), where one player selects the row, and the other player selects the column. Each cell specifies the negative cost, based on the minimum number of moves its takes to reach the goal, of the row player (first number) and the column player (second payoff), respectively. We refer to this game as the multi-step prisoners' dilemma (MSPD).

The Nash equilibrium of a single round of the MSPD is for each agent to defect. However, in the repeated game, there are an infinite number of Nash equilibria of the repeated game [8]. Furthermore, the Nash bargaining solution of the game is for both agents to cooperate. Thus, without knowledge of the behavior of one's associate, it is unclear how this game should be played.

Due to the relatively large state-space of this game compared to many commonly studied repeated games in the literature, artificial learning algorithms without previous knowledge of the dynamics of the game must simultaneously solve two decision problems. First, the agent must make the *high-level* decision of determining which gate it should move through given the behavior of its associate. This high-level decision problem is the iterated prisoners' dilemma game shown in Figure 2(b) *if* both agents take a shortest path through their chosen gate, or the nearest open gate if the chosen gate closes. Thus, the second decision problem is the *low-level* control problem of determining the shortest path. Since the low-level control problem often takes several rounds for an artificial agent to learn, the *high-level* game changes over time, thus complicating the game.

Given these challenges, it is interesting to observe the behavior of existing artificial learning algorithms in this game.

3. BEHAVIOR OF EXISTING MAL ALGORITHMS IN THE MSPD

Existing MAL algorithms for repeated stochastic games fall into two categories: followers and leaders [15]. Follower algorithms typically use only their own payoffs to attempt to learn a best response to associates' strategies, while leader algorithms consider the payoffs of both players and attempt to coax or coerce associates to follow specific solutions. In this section, we evaluate the strengths and weaknesses of these two approaches to MAL in the MSPD using two representative algorithms. In subsequent sections, we begin to evaluate the extent to which LbD can be used to improve upon these algorithms.

3.1 Follower Algorithms in the MSPD

Most reinforcement learning [13] algorithms for stochastic games are follower algorithms. These algorithms experimentally acquire knowledge of their environment, and use this knowledge to derive a strategy π given the current state of the world (s). Example algorithms include Q-learning [20], Minimax-Q [14], Nash-Q [11], Correlated Q-learning [9], and WoLF-PHC [3].

For simplicity, we consider a more basic follower algorithm to represent the learning capabilities of followers in the MSPD. This algorithm uses Monte Carlo reinforcement learning (MCRL) to learn $V(s,a)$, the value of taking action a from state s. The algorithm then acts according to the following strategy rule:

$$a \leftarrow \begin{cases} \arg\max_{a \in A(s)} V(s,a) & \text{with prob. } 1-\varepsilon \\ \text{random} & \text{otherwise} \end{cases} \qquad (1)$$

where $A(s)$ is the set of available actions from state s, and $\varepsilon \in [0,1]$ is agent's exploration rate. For simplicity, we use $\varepsilon = 0.1$ throughout this paper.

MCRL estimates $V(s,a)$ as the average reward it has received when it has taken a from state s in the past. Formally, let $R(s,a)$ be the set of rewards obtained by the agent for taking action a from state s. Then,

$$V(s,a) = \frac{1}{|R(s,a)|} \sum_{r \in R(s,a)} r \qquad (2)$$

where $|R(s,a)|$ denotes the size of the set $R(s,a)$.

In repeated stochastic games, it is unclear what the values in $R(s,a)$ represent. In our implementation of MCRL, each reward $r \in R(s,a)$ is based on the number of moves taken by the agent in the remainder of the current round (after action a was taken from state s), plus the number of moves taken by the agent in the subsequent round. This representation allows the agent to determine how its actions in the current round affects it payoffs in the next round.

In order to learn more quickly, the MCRL algorithm we consider in this paper uses k-nearest neighbor function approximation ($k = 20$) to determine $V(s,a)$. State and distance metrics used by the algorithm are given in the Appendix.

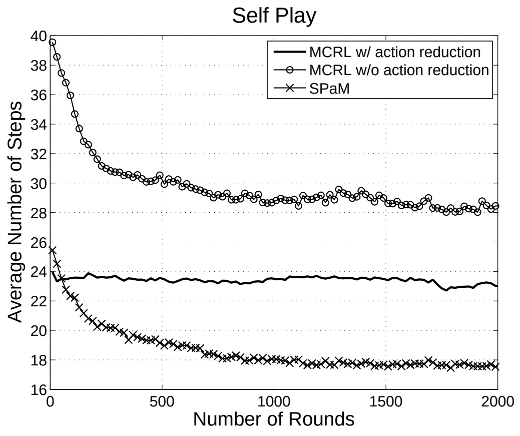


Figure 3: Average number of steps taken by MCRL (with and without action reduction) and SPaM in the MSPD in self play. Displayed values are a sliding-window average from 25 trials.

Figure 3 plots the average performance of two versions of MCRL in the MSPD when it associates with a copy of itself (self play). The first version of MCRL, labeled *MCRL w/o action reduction*, selects its actions from among the four compass directions. Since MCRL must play randomly until it stumbles upon its goal, it learns quite slowly in this game since random behavior is unlikely to take it too its goal within 40 moves (when the round is automatically terminated). Thus, MCRL without action reduction has such a hard time solving the low-level control problem, its performance is worse than mutual defection, in which each agent must move 25 steps to get to its goal. Rather the MCRL agents typically learned to alternate between going through gate 1 (10 steps) and playing randomly while the other player goes through gate 1 (40 steps). Continued random exploration ($\varepsilon = 0.1$) keeps MCRL's average number of steps higher than 25.

The second version of MCRL shown in Figure 3, labeled *MCRL w/ action reduction*, applies an action-reduction technique to reduce the number of actions the agent needs to consider taking. Such techniques, which require knowledge about the transition characteristics of the game, eliminate actions that are unlikely to move the agent closer to an open gate or its goal. In this way, the low-level control problem is learned more quickly, which allows MCRL to focus on the high-level decision problem of determining which gate it should move through. As such, Figure 3 shows that *MCRL w/ action reduction* performs much better than *MCRL w/o action reduction*, though it still learns the myopic solution of mutual defection in self play. Ironically, continued random exploration causes its average number of moves per round to be little less than mutual-defection.

Thus, given domain-specific knowledge, action-reduction algorithms can be used to help solve the low-level control problem. However, even given such assistance, follower algorithms such as MCRL still tend to learn myopic, less successful solutions in the MSPD in self play.

3.2 Leaders Algorithms in the MSPD

So-called leader algorithms [16] have been devised to coax follower algorithm to learn less-myopic strategies. Leader algorithms for repeated games include the famous tit-for-tat strategy for the prisoners' dilemma [2], and the *Bully* strategy for the chicken matrix game [15]. While most leader algorithms have been developed for matrix games, some leader algorithms exist for repeated stochastic games [6, 4].

In this paper, we represent the performance leader algorithms in the MSPD with SPaM [4], which learns to cooperate in self play in the MSPD. As such it outperforms MCRL in self play (Figure 3). SPaM computes both a follower utility function (such as MCRL) and a "social" utility function. The social utility function assigns high utility to actions that give its associate a high payoff for doing the "right" thing or give its associate a low payoff for doing the "wrong" thing. Additionally, the social utility function gives low utility to actions that either give its associate a high payoff for doing the "wrong" thing or give its associate a low payoff for doing the "right" thing. Once computed, SPaM combines the social and follower utility functions by computing a set of socially acceptable actions (based on the social utility function), and selecting the action from that set with the highest follower utility. SPaM uses the same action-reduction technique as MCRL.

In the remainder of this section, we further analyze SPaM and MCRL in the MSPD to better determine their strengths and weaknesses.

3.3 Leaders and Followers: Beyond Self Play

We are interested in general-purpose MAL algorithms for repeated stochastic games that learn quickly and effectively when associating with a wide range of associates. Effective MAL algorithms should learn effectively when associating with leader and follower algorithms, as well as with agents that do not learn. Thus, we now compare and contrast the performance of MCRL (with action reduction) and SPaM when they associate with each other and with Random, a hand-coded algorithm that randomly selects between gates 1 and 2 in each round, and then moves directly toward that gate.

Figure 4 shows the asymptotic performance of MCRL and SPaM when playing the MSPD with these three associates. The figure shows that SPaM performs effectively when associating with both itself and MCRL. In addition to learning mutual cooperation in self play, it teaches MCRL to cooperate, thus resulting in a low average number of steps per round in both cases. On the other hand, MCRL performs effectively when it associates with SPaM, but does not learn effective solutions in self play.

However, MCRL scores better when associating with Random than does SPaM. Since Random does not react to its associates behavior, there is no incentive for an agent to cooperate with it. Thus, the best thing to do against Random is to always defect, which MCRL learns to do (with some continued exploration since $\varepsilon = 1$). SPaM, on the other hand, continues to try to teach Random to cooperate, which causes it to cooperate when it believes that Random will cooperate, and to defect when it believes that Random will defect.

These results demonstrate the strengths and weaknesses of typical leader and follower MAL algorithms. Followers tend to perform well against teachers and against static associates, but they learn (less-effective) myopic solutions when associating with other followers. Leaders tend to perform well when associating with followers and (sometimes) other

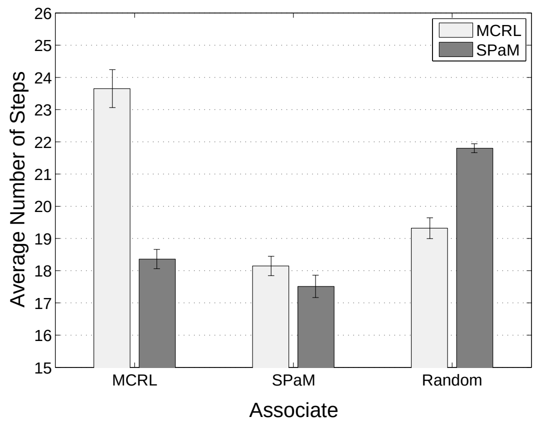


Figure 4: Asymptotic number of steps taken by MCRL and SPaM in the MSPD when associating with three algorithms. Results are an average of 25 trials. Error bars show a 95% confidence interval on the mean.

leaders, but they often do not perform well when associating with algorithms that do not learn.

In addition to not learning effectively against all associates, both MCRL and SPaM require domain-specific knowledge to learn effectively in the MSPD. For example, both algorithms required action reduction to solve the low-level control problem in order to effectively focus on the high-level control problem. Furthermore, SPaM requires knowledge of the payoffs of its associates, as well as high-level knowledge about what it means to cooperate and defect. These domain-specific needs limit the generalizability of these learning algorithms.

In the remainder of this paper, we consider the potential of LbD techniques to achieve general-purpose learning algorithms for repeated stochastic games. In the next section, we discuss two potential LbD algorithms. We then evaluate the potential of these algorithms to simultaneously and effectively learn low- and high-level behaviors in self play, and when associating with followers, leaders, and static algorithms.

4. TWO LBD ALGORITHMS

The two LbD algorithms we describe in this section use two different genre of machine learning. The first algorithm, called Imitator, seeks to imitate the teacher's demonstrations using a simple classification technique. Thus, we anticipate that this algorithm will be effective given (1) effective demonstrations from a human teacher and (2) good state features. However, when human input is less-effective, we anticipate that this algorithm will fail. Thus, the second LbD algorithm we consider uses reinforcement learning to distinguish between effective and ineffective demonstrations. This algorithm, which we call MCRL-LbD, utilizes the concept of policy reuse [7].

Before describing these algorithms in detail, we note that neither algorithm uses fundamentally new LbD techniques to those already proposed in the literature. In fact, both LbD using reinforcement learning and imitation learning

```
(1) D = {}
(2) Repeat while game continues
(3)    Observe $s$
(4)    $demoRound \leftarrow$ Schedule()
(5)    if ($demoRound$)
(6)       Observe action a and add ($s$, a) to D
(7)    Otherwise
(8)       Select action a according to Eq. 3
```

Table 1: The Imitator algorithm.

have been used repeatedly in the literature [1]. However, we are not aware of any prior work in which these or similar LbD algorithms have been analyzed in repeated stochastic games.

4.1 Imitator

An LbD algorithm utilizes human demonstrations to derive a strategy given the current state s. This strategy, which we denote $\pi(s)$, specifies a probability distribution over the actions $a \in A(s)$, where $A(s)$ is the set of available actions in state s. Let $\pi(s, a)$ be the probability assigned to action a by $\pi(s)$.

To derive a strategy that imitates the previously observed behavior of the human teacher, Imitator identifies those demonstrations which were given in similar states to the current state s. Formally, let $d = (d_s, d_a)$ be a demonstration from the human teacher, where d_s was the state of the world when the demonstration was observed, and d_a was the observed demonstration. Let D be the set of demonstrations observed up to the current round. Then, given D and the current state s, Imitator finds the k samples $d \in D$ such that the distance between s and d_s, defined by $dist(s, d_s)$, is the smallest. Let $N(s)$ denote this set of samples.

Given the set $N(s)$, Imitator computes $\pi(s, a)$ for each $a \in A(s)$ as follows:

$$\pi(s,a) = \frac{\sum_{d \in N(s)} I(a, d_a) \frac{1}{1+dist(s,d_s)^2}}{\sum_{n \in N(s)} \frac{1}{1+dist(s,d_s)^2}}, \tag{3}$$

where $I(a, d_a)$ is the indicator function such that

$$I(a, d_a) = \begin{cases} 1, & \text{if } a = d_a \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

In words, the probability $\pi(s, a)$ depends on (1) the number of samples $d \in N(s)$ for which the demonstrated action d_a matches a, and (2) the similarity between the sample state d_s and the current state s.

The Imitator algorithm is summarized in Table 1. Line (4) of the algorithm makes a call to the function $Schedule()$. This function returns true when the human controls the behavior of the agent (i.e., provides a demonstration) in the current round, or false when the agent must act on its own (according to Eq. 3).

In addition to this schedule, Imitator also requires definitions of state and the distance metric $dist(s_i, s_j)$. For the MSPD, we use the same state definition and distance metric used by MCRL, which is given in the Appendix. We note the obvious reliance of the algorithm on a good set of state features and a good distance metric. LbD algorithms with less reliance on these pre-defined specifications are an important topic left to future work.

```
(1) D = {}
(2) t = 1
(3) Repeat while the game continues
(4)    Repeat while the current round continues
(5)       Observe $s$
(6)       $demoRound \leftarrow$ Schedule()
(7)       if ($demoRound$)
(8)          Observe action a and add ($s$, a) to D
(9)       Otherwise
(10)         Compute $\delta(s)$
(11)         Select action a according to Eq. 5
(12)   Update $V(s,a)$ for each $(s,a)$ visited
             in round t - 1
(13)   t = t + 1
```

Table 2: The MCRL-LbD algorithm.

4.2 MCRL-LbD

Unlike Imitator, which assumes that human demonstrations are effective, MCRL-LbD makes its own assessment of the effectiveness of human demonstrations. To do this, it mimics human demonstrations (like Imitator) in early rounds of the game, and uses these experiences to estimate $V(s, a)$ (like MCRL). As it accumulates experiences, it slowly shifts its strategy to maximize its payoffs using its utility estimates $V(s)$ rather than following the human teacher's behavior. Thus, in later rounds, new demonstrations serve only to dictate the agent's exploration of its action space.

Formally, when the human does not control the agent via demonstrations, MCRL-LbD uses the following strategy rule to choose its actions:

$$\pi(s,a) \leftarrow \begin{cases} I(a, a^*) & \text{with prob. } 1 - \delta(s) \\ \text{Eq. 3} & \text{otherwise} \end{cases} \tag{5}$$

where $a^* = \arg\max_{b \in A(s)} V(s, b)$ and $\delta(s)$ is the probability that controls whether MCRL-LbD imitates human demonstrations or follows its utility estimates. As the number of human demonstrations from states similar to s increases, the agent places more confidence in its estimates of $V(s, a)$. This notion is reflected in the following equation, which we use to compute $\delta(s)$ in the MSPD:

$$\delta(s) = 0.02 + \left(1 - \max_{a \in A} \phi(s,a)\right)^3 \tag{6}$$

Here, $\phi(s, a) \in [0, 1]$ is known as the support for the pair (s, a), and is given by

$$\phi(s,a) = \frac{1}{k} \sum_{d \in N(s,a)} \frac{1}{1 + dist(s,d_s)^2}. \tag{7}$$

In this latter equation, $N(s, a)$ is the set of k demonstrations $d \in D$ such that $d_a = a$ and $dist(d_s, s)$ is minimized.

The MCRL-LbD algorithm is summarized in Table 2.

5. RESULTS

Recall that the goal of this paper is to begin to address the potential of LbD in repeated stochastic games by addressing two related questions. These questions are: What kinds of LbD algorithms, if any, are likely to be successful in repeated stochastic games? And, how effective do demonstrations need to be for LbD algorithms to facilitate successful learning in these games.

In this section, we begin to answer these questions using simulation studies in which Imitator and MCRL-LbD, given various degrees of demonstration effectiveness, are paired with other learning algorithms in the MSPD.

5.1 Experimental Setup

We paired both Imitator and MCRL-LbD with themselves, MCRL (with action reduction), SPaM, and Random in the MSPD in 5,000 round games. In these simulations, simulated human demonstrations in the form of hand-coded behaviors were provided to each LbD algorithm for between three and six consecutive rounds. The LbD algorithm then acted autonomously for 10 to 50 consecutive rounds, after which simulated demonstrations were again provided. Thus, on average, demonstrations were provided for five out of every 30 rounds for the first 4,000 rounds of the game. However, no demonstrations were provided to the LbD algorithms in the final 1,000 rounds of the game.

In order to evaluate the effect of demonstration quality on the performance of Imitator and MCRL-LbD, we ran simulations using the following three hand-coded demonstration behaviors:

1. Tit-for-tat (TFT) – This behavior moves directly toward gate 1 if the associate defected in the previous round, and gate 2 if the associate cooperated in the previous round. Given its robustness in the iterated prisoners' dilemma [2], TFT was chosen to represent demonstrations from an informed teacher.

2. Random – At the beginning each round, this behavior randomly selects gate 1 or 2 and then moves directly toward that gate. These demonstrations represent demonstrations from less-informed human teachers.

3. Learner – This behavior changes over time in attempt to mimic a human teacher that begins the game as a less-informed teacher, but then becomes more-informed as the number of rounds increases. Specifically, Learner defects with probability one on its first demonstration, after which it slowly transitions to Random behavior over the next 25 rounds of demonstrations. It then slowly evolves into TFT over its next 65 rounds of demonstrations.

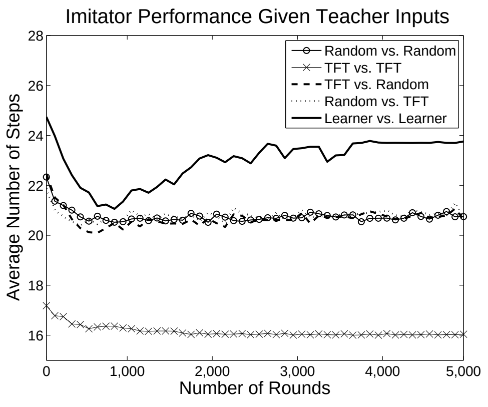
The two LbD algorithms (Imitator and MCRL-Lbd) combined with the three demonstration behaviors (TFT, Random, and Learner) combine to form six different learning agents. We refer to these learning agents as Imitator-TFT, Imitator-Random, Imitator-Learner, MCRL-TFT, MCRL-Random, and MCRL-Learner. We now describe the performance of each of these agents in the MSPD.

5.2 Findings

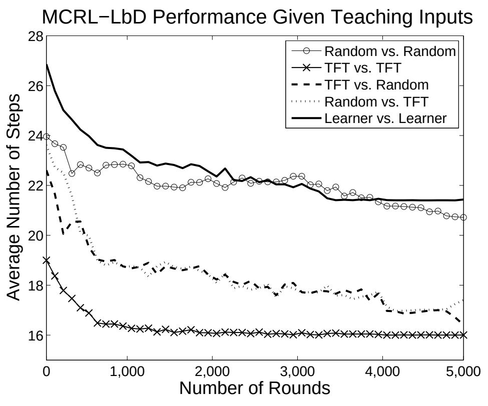
We first describe the behavior of the LbD agents in the MSPD in self play. We then discuss their performance when paired with MCRL, SPaM, and Random.

5.2.1 *Self play.*

The average performances of Imitator and MCRL-LbD in self play given the various forms of human demonstrations are shown in Figure 5. We make several observations



(a)



(b)

Figure 5: Average number of steps required by (a) Imitator and (b) MCRL-LbD, respectively, in self play given various demonstration behaviors. Results are a moving-window average of 10 different simulations.

about these results. First, while Imitator-TFT's payoffs are slightly better (less number of steps) in early rounds than MCRL-TFT's, both of these agents learn to cooperate in self play. These results indicate that of these algorithms are able to effectively solve the high- and low-level decision problems simultaneously in stochastic games when both of the agent receive informed demonstrations from a teacher.

Second, when Imitator-TFT was paired with Imitator-Random, both agents effectively learn the low-level control problem. However, their high-level decisions (i.e., the gates they choose) are essentially random, as Imitator-Random randomly selects which gate it approaches, and Imitator-TFT then follows suit in the subsequent round. Thus, the learned behavior of the agents is as if they both played the matrix game shown in Figure 2(b) randomly, which results in an average number of steps per round of about 20.75 for both agents.

However, in most cases, MCRL-TFT and MCRL-Random learned to cooperate when paired together in the MSPD, as mutual cooperation emerged in nine of the ten simulations in which these agents were paired. In these simulations, MCRL-TFT learned to act as a leader to coax MCRL-Random to cooperate. Thus, after 5,000 rounds, on average, these agents only required between 16 and 17 moves per round to reach their goals.

These results suggest that MCRL-LbD is able to act as both an effective leader and an effective follower, depending on the demonstrations it receives from the human teacher. However, at least in self play, these results suggest that Imitator is less able to do so.

Third, Figure 5 shows that neither Imitator-Random nor MCRL-Random are able to consistently learn to cooperate in self play. While Imitator-Random learned to play randomly in each simulation (in conformance with the provided demonstrations), MCRL-Random learned mutual defection in five simulations, and mutual cooperation in the other five simulations. Thus, while MCRL-Random does not always learn mutual cooperation in self play, it is sometimes able to do so. This suggests that LbD algorithms that seek to distinguish between informed and less-informed human demonstrations could potentially learn effectively in repeated stochastic games, even when human demonstrations are imperfect.

Fourth, in self play, Imitator-Learner converged to mutual defection in nine simulations, and mutual cooperation in one simulation. We initially believed that Imitator-Learner would converge to mutual cooperation in self play since it eventually learns to play TFT. However, since TFT defects against defectors, the behavior produced by mimicking initial demonstrations caused both agents to continue to defect against each other in most cases. MCRL-Learner was also unable to consistently learn to cooperate in self play, though it did better than Imitator-Learner. MCRL-Learner converged to mutual defection in six simulations, while it converged to mutual cooperation in the other four simulations.

These latter results suggest that straight-forward implementations of LbD algorithms may not be suitable for learning non-myopic solutions in repeated stochastic games when human teachers learn throughout the course of the repeated game. However, when demonstrations are informed with respect to low-level control, the evidence of some mutual cooperation in MCRL-Learner agents suggests that well-formed variations of these algorithms could be successful. We leave further study of this interesting issue to future work.

5.2.2 *Associating with other algorithms.*

Figure 6 shows the average asymptotic performance of Imitator and MCRL-LbD given various demonstration behaviors against MCRL, SPaM, and Random. Figure 7 also shows the average percentage of the time the LbD algorithms cooperated in the final 1,000 rounds when associating with each agent.

The figures show that none of the LbD agents behaved ideally against all three associates. However, MCRL-TFT comes the closest. It successfully learned to cooperate with SPaM, while learning to defect against Random. Furthermore, it learned mutual cooperation with MCRL in six out of ten simulations. This latter result falls short of the performance of both Imitator-TFT and Imitator-Learner, which lead MCRL to always cooperate. However, neither Imitator-TFT nor Imitator-Learner learn to always defect against Random.

MCRL-Random and MCRL-Learner also learn to cooperate with SPaM and defect against Random, but, they do not lead MCRL to cooperate as often as does MCRL-TFT. This suggests that the quality of human demonstrations can

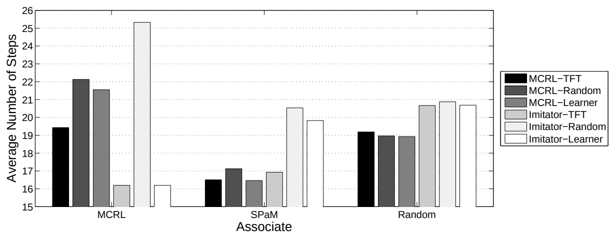


Figure 6: Average number of steps required by MCRL-LbD and Imitator given various human inputs to reach its goal when associating with MCRL, SPaM, and Random in the last 1,000 rounds. Results are an average of 10 trials.

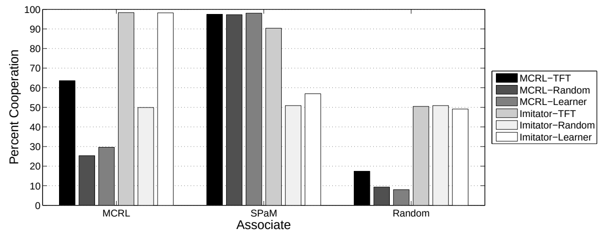


Figure 7: Percent cooperation in the last 1,000 rounds of MCRL-LbD and Imitator given various human inputs to reach its goal when associating with MCRL, SPaM, and Random. Results are an average of 10 trials.

be of great importance to LbD algorithms, though some evidence of mutual cooperation in simulations between MCRL-Random and MCRL provides hope that future LbD algorithms could learn to be leaders even when human demonstrations are not.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we described an investigation into the effectiveness of learning by demonstration (LbD) in repeated stochastic games. This investigation was designed to determine the potential of general-purpose LbD algorithms to help agents learn both high- and low-level skills capable of producing non-myopic equilibria. To do this, we described and analyzed the performance of two straight-forward LbD algorithms in a multi-stage iterated prisoners' dilemma given various forms of simulated human demonstrations. Results showed that LbD does help learning agents learn non-myopic equilibrium in repeated stochastic games when human demonstrations are well-informed. On the other hand, when human demonstrations are less informed, these agents do not always learn behavior that produces (less-successful) non-myopic equilibria. However, it appears that well-formed variations of LbD algorithms that distinguish between informed and uninformed demonstrations could learn non-myopic equilibrium.

Results also suggest that new algorithms and techniques need to be developed that can learn effectively when human teachers learn to improve their demonstrations throughout the course of the repeated game. While good initial behavior can be critical in some repeated games [2], future general-purpose LbD algorithms for repeated games should be able to better leverage ever-improving demonstrations in order to learn increasingly successful behavior.

Another area of future work involves creating an appropriate context for the human teacher that allows him or her to provide more-informed demonstrations. When humans play iterated prisoners' dilemma games, their performance is contingent on many factors [12, 17]. Thus, LbD algorithms could potentially provide information about the game and associates that would provide a context that facilitates better demonstrations.

A third area of important future work involves developing algorithms that derive state and distance metrics from human input. In this paper, we assumed that good state and distance metrics were known, but this is not likely to be the case in many real-time systems that can be modeled as repeated stochastic games. In such situations, in addition to providing demonstrations of desired behavior, the human can potentially provide information about the underlying representations that the algorithm should use [5].

7. REFERENCES

[1] B. D. Argall, S. Chernova, and M. Veloso B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

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

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

[4] J. W. Crandall and M. A. Goodrich. Establishing reputation using social commitment in repeated games. In *AAMAS workshop on Learning and Evolution in Agent Based Systems*, New York City, NY, 2004.

[5] J. W. Crandall, M. A. Goodrich, and L. Lin. Encoding intelligent agents for uncertain, unknown, and dynamic tasks: From programming to interactive artificial learning. In *AAAI Spring Symp. on Agents that Learn from Human Teachers*, 2009.

[6] E. M. de Cote and M. L. Littman. A polynomial-time nash equilibrium algorithm for repeated stochastic games. In *Conference on Uncertainty in Artificial Intelligence*, pages 419–426, 2008.

[7] F. Fernandez and M. Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the 5th International Joint Conference on Autonomous agents and multiagent systems*, pages 207–212, 2006.

[8] H. Gintis. *Game Theory Evolving: A Problem-Centered Introduction to Modeling Strategic Behavior.* Princeton University Press, 2000.

[9] A. Greenwald and K. Hall. Correlated Q-learning. In *Proceedings of the 20th International Conference on Machine Learning*, pages 242–249, 2003.

[10] P. J. Hoen, K. Tuyls, L. Panait, S. Luke, and J. A. L. Poutre. An overview of cooperative and competitive multiagent learning. In *LAMAS*, pages 1–46, 2005.

[11] J. Hu and M. P. Wellman. Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Proceedings of the 15th International Conference on Machine Learning*, pages 242–250, 1998.

[12] K. Iyori and S. H. Oda. Prisoner's dilemma network: its experiments and simulations. In *Proc. of the Seventh Experimental Economics Conference of Japan*, pages 150–161, 2003.

[13] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–277, 1996.

[14] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the 11th International Conference on Machine Learning*, pages 157–163, 1994.

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

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

[17] K. Miwa, H. Terai, and S. Hirose. Social responses to collaborator: dilemma game with human and computer agent. In *Proc. of the 30th annual meeting of the cognitive science society*, pages 2455–2460, 2008.

[18] L. S. Shapley. Stochastic games. *Proc. Nat. Acad. Sciences*, 39:1095–1100, 1953.

[19] Y. Shoham, R. Powers, and T. Grenager. If multi-agent learning is the answer, what is the question? *Artif. Intell.*, 171(7):365–377, 2007.

[20] C. J. C. H. Watkins and P. Dayan. Q-learning. *Mach Learning*, 8:279–292, 1992.

Appendix: State and Distance Metrics

The *state features* used by each of the algorithms described in this paper were:

1. $(x_i, y_i, x_{-i}, y_{-i})$ – the x, y coordinates of each player, respectively
2. (g_1, g_2, g_3, g_4) – the boolean status of each gate (0 for closed, 1 for open)
3. $(g_i^{\mathrm{t}-1}, g_{-i}^{\mathrm{t}-1})$ – the gate each player passed through in the previous round
4. $(d_{\mathrm{N}}, d_{\mathrm{S}}, d_{\mathrm{E}}, d_{\mathrm{W}})$ – the agent's proximity to walls or closed gates in each of the compass directions, where $d_j = 0$ if there was a wall next to the agent in direction j, and $d_j = 1$ otherwise

The *distance* between states v and z, denoted $dist(v, z)$, was defined as:

$$\begin{aligned} dist(v,z) \quad = \quad & |v.x_i - z.x_i| + |v.y_i - z.y_i| + \\ & |v.x_{-i} - z.x_{-i}| + |v.y_{-i} - z.y_{-i}| + \\ & |v.g_i^{\mathrm{t}-1} - z.g_i^{\mathrm{t}-1}| + |v.g_{-i}^{\mathrm{t}-1} - z.g_{-i}^{\mathrm{t}-1}| + \\ & \sum_{j=1}^{4} 2 \cdot |v.g_j - z.g_j| + \sum_{j \in A} 2 \cdot |v.d_j - z.d_j| \end{aligned}$$

where $A = \{N, S, E, W\}$.