Learning in Repeated Games with Minimal Information: The Effects of Learning Bias

Jacob W. Crandall and **Asad Ahmed**
Computing and Information Science Program
Masdar Institute of Science and Technology
Abu Dhabi, UAE
{jcrandall,aahmed}@masdar.ac.ae

Michael A. Goodrich
Computer Science Department
Brigham Young University
Provo, UT 84602, USA
mike@cs.byu.edu

Abstract

Automated agents for electricity markets, social networks, and other distributed networks must repeatedly interact with other intelligent agents, often without observing associates' actions or payoffs (i.e., minimal information). Given this reality, our goal is to create algorithms that learn effectively in repeated games played with minimal information. As in other applications of machine learning, the success of a learning algorithm in repeated games depends on its learning bias. To better understand what learning biases are most successful, we analyze the learning biases of previously published multi-agent learning (MAL) algorithms. We then describe a new algorithm that adapts a successful learning bias from the literature to minimal information environments. Finally, we compare the performance of this algorithm with ten other algorithms in repeated games played with minimal information.

Introduction

Large-scale distributed networks, such as new-age electricity markets and social networking sites, are becoming increasingly reliant on automated agents. To be successful, these automated agents must repeatedly interact with other intelligent agents. Such situations can be modeled as repeated games played with minimal information, wherein the actions and payoffs of associates are unobservable. Given the uncertainty present in these environments, it is important that multi-agent learning (MAL) algorithms be developed that can learn effectively in these games.

MAL in repeated games has been the subject of many research papers over the past couple of decades (Shoham, Powers, and Grenager 2007; Hoen et al. 2006). Many of these papers articulate a particular goal (such as best response, Nash equilibrium, and no regret) for MAL, and then derive an algorithm to begin to achieve that goal. Rather than articulate a new vision for MAL, we focus instead on the means that have been used by algorithm designers to realize their goals. That is, we focus on the set of assumptions, representations, and rules that algorithms encode to determine a strategy from experience. These assumptions, representations, and rules form an algorithm's *learning bias.* Thus, in this paper, we seek to determine what learning biases are most likely to produce behavior that yields high payoffs in repeated matrix games played with minimal information.

To determine what learning biases are most successful in repeated games played with minimal information, we dissect the learning biases encoded by a set of previously published MAL algorithms. We then create a new algorithm by adapting a successful learning bias to minimal information environments. We then compare the performance of this algorithm with ten other algorithms in a large number of repeated matrix games played with minimal information.

Learning Bias in Multi-agent Learning

The learning bias of an MAL algorithm consists of the set of assumptions, representations, and rules it uses to define its strategy space, model representations, and strategy selection rules. In this section, we identify and discuss the design decisions made in some existing MAL algorithms.

Strategy Space

In a repeated game, a strategy consists of a *probability distribution* over the agent's action set for each *state of the world*. Thus, to define the algorithm's strategy space, an MAL algorithm designer must define a set of possible probability distributions and the states of the world.

Probability distributions Essentially three different kinds of probability distributions have been encoded by designers of MAL algorithms. First, some algorithms choose actions from the set of all pure and mixed strategies. Second, some algorithms encode only pure strategies. This simplifying decision precludes the algorithm from fulfilling certain goals (such as Nash equilibrium), but can help the agent react quickly to changes in the environment. Third, some algorithms encode a few mixed strategies in addition to all pure strategies.

States of the world A strategy defines an agent's behavior over the world states defined by the algorithm designer. Since agents play the same game in each round of a repeated matrix game, many designers of MAL algorithms have assumed a single world state. We refer to this state representation as *stateless*. This limiting assumption means that the agent's strategy consists of a single probability distribution over its actions, which can preclude the agent from reasoning about how its current actions affect its future payoffs.

Table 1: Assumptions, representations, and rules encoded by algorithms in repeated matrix games.

Algorithm	Probability Distributions			State Representation		Priors		Belief Models			Reward Representation		Strategy Selection Rules				
	Mixed + Pure	Pure only	Pure + Some Mixed	Stateless	Recurrent State	Optimistic	Random	Opponent Strat.	Own Strat.	Utility	Imm. Reward	Imm. + Future Reward	Gradient Ascent	Best Response	Maximin	Satisficing	Teaching
*WPL (Abdallah and Lesser 2008)	✓			✓			✓		✓	✓	✓		✓				
Hyper-Q (Tesauro 2004)			✓	✓			✓	✓		✓		✓		✓			
FP (Fudenberg and Levine 1998)		✓		✓		?	?	✓			✓			✓			
sFP (Fudenberg and Levine 1998)	✓			✓		?	?	✓			✓			✓			
Nash-Q (Hu and Wellman 1998)	✓			✓			✓			✓		✓		✓			
**Q-learning (Sandholm and Crites 1996)		✓			✓		✓			✓		✓		✓			
*WoLF-PHC (Bowling and Veloso 2002)	✓			✓			✓		✓	✓		✓	✓				
*GIGA-WoLF (Bowling 2005)	✓			✓			✓		✓	✓	✓		✓				
*Exp3 (Auer et al. 1995)	✓			✓			✓			✓	✓		✓				
Minimax-Q (Littman 1994)	✓			✓			✓			✓		✓			✓		
M-Qubed (Crandall and Goodrich 2011)			✓		✓	✓				✓		✓		✓	✓		
*Satisficing (Stimpson and Goodrich 2003)			✓		✓	✓			✓	✓	✓					✓	
Tit-for-tat (Axelrod 1984)		✓			✓	✓											✓
Godfather (Littman and Stone 2001)			✓		✓	✓						✓					✓
Bully (Littman and Stone 2001)		✓		✓		✓					✓						✓

* The algorithm can be used in games played with minimal information.

** The cited version of the algorithm cannot be used in games played with minimal information, but other versions of the algorithm can.

? Specification of priors is left to the implementation.

Alternately, an agent can use information from previous rounds to define its state. This has been called *recurrent state* (Moody et al. 2004). For example, Tit-for-tat defines world state by its associate's previous action (Axelrod 1984). Similarly, learning algorithms have used the previous actions of all agents (called joint actions) to define state (Sandholm and Crites 1996; Crandall and Goodrich 2011; Bouzy and Metivier 2010). However, it is not possible to define state with associates' previous actions in games played with minimal information.

Table 1 (cols. 2-6) summarizes the strategy spaces encoded by a representative set of algorithms in matrix games. Many of these algorithms encode all mixed and pure strategies while using a stateless representation of the world.

Model Representation

An MAL algorithm designer must also specify the algorithm's *belief models*, *reward representation*, and initial parameter settings (i.e., *priors*).

Belief models Most learning algorithms utilize mathematical structures, updated after each time step, to form the agent's beliefs about the world. These models typically fit into three categories: *opponent strategy models*, *own strategy models*, and *utility models*. Opponent models refer to beliefs about the behaviors of associates, which the agent uses to compute expected rewards. For example, FP models the empirical distribution of opponents strategies. From this model, it estimates the expected reward for taking each action. Most algorithms that use opponent models cannot be used in games played with minimal information since the behavior of associates is not directly observable.

Some algorithms use mathematical structures that explicitly model their own strategies. Such models allow the agent to reflect on the success of its current strategy. For example, to determine its strategy at time $t + 1$, GIGA-WoLF remembers and compares its current mixed strategy x_t with the mixed strategy z_t, which changes more slowly than x_t.

Utility models, the most popular form of belief model for the algorithms shown in Table 1, estimate the reward the agent will received for executing a particular strategy in a given state. Perhaps the most prominent example of an algorithm that uses a utility model is Q-learning, which seeks to learn the Q-value of each state-action pair.

Reward representation MAL algorithms usually consider two forms of rewards: *immediate rewards* and *future rewards*. Algorithms that encode only immediate rewards update the agent's model estimates using only the immediate reward received in that time period. Algorithms that encode future rewards combine the immediate reward with some estimate of the payoffs the agent will receive in the future.

Priors While initial parameter settings (or priors) are sometimes unimportant in single-agent learning (e.g., (Watkins and Dayan 1992)), they can be very important in multi-agent domains. Despite their importance, many descriptions of MAL algorithms give little thought to priors. Rather, information about the priors is often (1) not specified, (2) set to some fixed value (such as 0), or (3) initialized randomly. We refer to such priors as *random*.

However, some algorithms are specific about priors. These algorithms often set priors *optimistically* to promote behavior that is likely to be profitable (such as cooperation). For example, tit-for-tat sets the initial state of the world to cooperate, which essentially gives associates the benefit of the doubt. Similarly, M-Qubed and the satisficing learning algorithm rely on high initial Q-values and aspirations (Crandall and Goodrich 2011; Stimpson and Goodrich 2003). This biases these algorithms toward solutions that produce higher long-term payoffs.

Table 1 (cols. 7–13) categorizes the priors, belief models, and reward representations of the representative algorithms. Most of these algorithms use random priors with immediate rewards. Interestingly, this combination often corresponds to the use of mixed strategies and a stateless representation of the world. Only two algorithms (Godfather and M-Qubed) encode both optimistic priors and future rewards.

Strategy Selection

An algorithm typically uses its model estimates to determine which strategy to play from its strategy space. Table 1 lists the five types of strategy selection rules:

- The *gradient ascent* strategy rule selects a new strategy by nudging its previous strategy up the reward gradient. The algorithms that encode this strategy rule in Table 1 all define their strategy spaces with mixed strategies defined in a single world state.
- The *best response* strategy rule selects the strategy from its strategy space with the highest expected utility with respect to current belief models. Note that this does not guarantee that the agent actually plays a best response to associates' strategies since belief models may be in error.
- The *maximin* strategy rule consists of playing the maximin strategy, the strategy that maximizes its minimum possible expected payoff. This strategy selection rule is difficult to implement in minimal information environments since computing the maximin strategy requires knowledge of the agent's payoff matrix, which the agent can only constructe if associates' actions are observable.
- The *satisficing* strategy rule repeats the strategy used in the previous round when the previous payoff meets some aspiration level. While perhaps the least popular strategy rule, it is effective in some repeated matrix games (Karandikar et al. 1998; Stimpson and Goodrich 2003).
- *Teaching* strategy rules are designed to elicit desired behavior from associates. For example, tit-for-tat reciprocates defection and cooperation in the prisoners' dilemma to make cooperation more profitable to its associate than defection. To implement teaching strategies, an agent typically must observe its associates' payoffs and actions. This limits the use of teaching strategies in games played with minimal information.

Most of the algorithms in Table 1 use either best response or gradient ascent strategy rules. An algorithm can potentially use multiple strategy rules. For example, M-Qubed encodes both best response and maximin strategy rules.

Summary of Learning Biases

We make two observations about the learning biases encoded by our representative set of algorithms (Table 1). First, few of the algorithms can be used in minimal information environments. Many of the algorithms that do qualify (1) select strategies from among all probability distributions (pure and mixed), (2) encode a stateless representation of the world, (3) use random priors, and (4) use either a best response or gradient ascent strategy selection rule. We call this learning bias the *rational learning bias*, since these algorithms aspire to game-theoretic ideals.

Second, recent empirical studies have highlighted the robustness of M-Qubed (Bouzy and Metivier 2010; Crandall and Goodrich 2011). However, M-Qubed requires knowledge of associates' actions, so it cannot be used in games played with minimal information. Given the success of M-Qubed, it is desirable to find an algorithm that encodes a similar learning bias, but that can be used in minimal information environments. We now present such an algorithm.

A New Algorithm

M-Qubed is a reinforcement learning algorithm that uses optimistic priors, encodes state from the previous joint action(s) played in the game, and balances best response and maximin strategy selection rules. However, because it requires knowledge of associates' actions to encode its state and implement its maximin strategy rule, it cannot be used in games played with minimal information.

R-lopars (*r*einforcement *l*earning with *o*ptimistic *p*riors and *a*ction-*r*eward *s*tate) utilizes a similar learning bias to M-Qubed, but it does not require knowledge of associates' actions. Specifically, R-lopars is identical to M-Qubed with two important differences. First, R-lopars does not include the maximin strategy in its strategy space. Second, rather than use the previous joint actions to encode recurrent state, R-lopars uses the combination of its previous action and its previous reward r_i^{t-1} to encode its state. Since r_i^{t-1} depends on the joint actions of the agents, this state representation is similar to the one encoded by M-Qubed, except that it does not encode the maximin strategy selection rule.

Formally, let A_i be the set of actions available to player i, $a_i^t \in A_i$ be the action played by player i at time t, and r_i^t be the payoff received by player i after playing its action in time t. Then, player i's state $s_i^t \in S$ (S is the set of states) at time t is $s_i^t = (a_i^{t-1}, r_i^{t-1})$. Given the state s_i^t, player i selects a_i^t from the probability distribution $\pi_i(s_i^t)$ given by

$$\pi_i(s_i^t) \leftarrow \begin{cases} \pi_i^*(s_i^t) & \text{if } s^* \in S^{\text{Prev}} \\ (1-\varepsilon_i^t)\pi_i^*(s_i^t) + \varepsilon_i^t \chi_i & \text{otherwise} \end{cases} \quad (1)$$

where S^{Prev} is the set of states visited in the last $|S|$ episodes (we assume S is finite), s^* is the state with the highest global Q-value given by $s^* = \arg\max_{s \in S} \max_{a \in A_i} Q_i(s,a)$), χ_i denotes the uniform probability distribution over the action set A_i, ε_i^t is the exploration rate, and $\pi_i^*(s_i^t)$ is the agent's best response strategy with respect to its Q-values:

$$\pi_i^*(s_i^t) \leftarrow \arg\max_{a \in A_i} Q_i(s_i^t, a). \quad (2)$$

Table 2: Learning biases of the learning algorithms used in our study. $\kappa^t(s)$ is the number of times state s has been visited before time t, and $\kappa^t(s,a)$ is the number of times action a has been taken in s before time t.

Algorithm	Probability Distributions	State Representation	Strategy Selection	Prior	Reward Representation	Parameter Values and Explanations
S-alg(1)	Pure strategies + random	Recurrent state: $\alpha_i^t \leq r^{t-1}$?	Satisficing	Optimistic (high α_i^t)	Immediate reward	$\alpha_i^0 = r_i^{\max}$, $\lambda = 0.99$, $\omega = 1$, $E = (50, 100]$; α_i^t denotes the aspiration level at time t. State is a boolean value determined by the inequality $\alpha_i^t \leq r^t$
S-alg(2)	Pure strategies + random	Recurrent state: $\alpha_i^t \leq \mu_i^t$?	Satisficing	Optimistic (high α_i^t)	Immediate reward	$\alpha_i^0 = r_i^{\max}$, $\lambda = 0.997$, $\omega = 2$, $E = (50, 100]$; μ_i^t is player i's running average payoff. State is a boolean value determined by the inequality $\alpha_i^t \leq \mu_i^t$
QL(0)	Pure strategies	Stateless	Best response	Random (Q-values)	Immediate reward	$\forall s, a, Q(s,a) = \text{rand}\left(0, \frac{r_i^{\max}}{1-\gamma}\right)$, $\alpha = \frac{1}{10+\kappa^t(s,a)/100}$, $\gamma = 0.95$, ϵ-greedy exploration, $\epsilon = \frac{1}{10+\kappa^t(s,a)/1000}$
QL(1)	Pure strategies	Recurrent state: a_i^{t-1}	Best response	Random (Q-values)	Imm. + future rewards	$\forall s, a, Q(s,a) = \text{rand}\left(0, \frac{r_i^{\max}}{1-\gamma}\right)$, $\alpha = \frac{1}{10+\kappa^t(s,a)/100}$, $\gamma = 0.95$, ϵ-greedy exploration, $\epsilon = \frac{1}{10+\kappa^t(s,a)/1000}$
QL(2)	Pure strategies	Recurrent state: a_i^{t-1}, a_i^{t-2}	Best response	Random (Q-values)	Imm. + future rewards	$\forall s, a, Q(s,a) = \text{rand}\left(0, \frac{r_i^{\max}}{1-\gamma}\right)$, $\alpha = \frac{1}{10+\kappa^t(s,a)/100}$, $\gamma = 0.95$, ϵ-greedy exploration, $\epsilon = \frac{1}{10+\kappa^t(s,a)\ 1000}$
QL-OP	Pure strategies	Recurrent state: a_i^{t-1}	Best response	Optimistic (high Q-values)	Imm. + future rewards	$\forall s, a, Q(s,a) = \frac{r_i^{\max}}{1-\gamma}$, $\alpha = \frac{1}{10+\kappa^t(s,a)\ 100}$, $\gamma = 0.95$, ϵ-greedy exploration, $\epsilon = \frac{1}{10+\kappa^t(s,a)/1000}$
R-lopars	Pure strategies	Recurrent state: a_i^{t-1}, r_i^{t-1}	Best response	Optimistic (high Q-values)	Imm. + future rewards	$\forall s, a, Q(s,a) = \frac{r_i^{\max}}{1-\gamma}$, $\alpha = 0.1$, $\gamma = 0.95$, $\epsilon_i^t = \frac{1}{10+\kappa^t(s,a)/1000}$, $E = (50, 100]$
GIGA-WoLF	Mixed + pure strategies	Stateless	Gradient ascent	Random (mixed strategy)	Immediate reward	$\eta_i = 0.02$, $\alpha = 0.1$ (used to update r_t); we added 1% exploration, random mixed strategy for initial policy
WPL	Mixed + pure strategies	Stateless	Gradient ascent	Random (mixed strategy)	Immediate reward	$\eta_i = 0.02$, $\alpha = 0.1$; we added 1% exploration, random mixed strategy for initial policy
WoLF-PHC	Mixed + pure strategies	Stateless	Gradient ascent	Random (mixed strategy)	Immediate reward	$\alpha = \frac{1}{100+t/10000}$, $\delta = \delta_w = \frac{1}{20000+t}$, $\delta_l = 4\delta_w$, $\forall s, a$, $Q(s,a) = \text{rand}\left(0, \frac{r_i^{\max}}{1-\gamma}\right)$, ε-greedy with 5% exploration, random mixed strategy for initial policy
Exp3	Mixed + pure strategies	Stateless	Gradient ascent	Random (mixed strategy)	Immediate reward	$\eta_i = \frac{\lambda}{\lvert A_i \rvert}$ ($\lvert A_i \rvert$ is the number of actions for player i), $g_i = 300000$

Like M-Qubed, R-lopars uses an on-policy Q-update after each time step:

$$Q_i(s_i^t, a_i^t) \leftarrow (1-\alpha)Q_i(s_i^t, a_i^t) + \alpha[r_i^t + \gamma V_i(s_i^{t+1})] \quad (3)$$

where $\alpha \in (0,1)$ is the learning rate, $\gamma \in (0,1]$ is the discount factor, and

$$V_i(s) = \sum_{a \in A_i} \pi_i^t(s,a) Q(s,a) \quad (4)$$

R-lopars is summarized in Algorithm 1. R-lopars plays randomly for the first E episodes, during which time it estimates its highest payoff $r_i^{\max}$. It then initializes its Q-values optimistically to its highest possible value given $r_i^{\max}$, and then learns and acts using Eqs. (1)–(4).

Algorithm 1 R-lopars

$r_i^{\max} = -\infty$
$\forall s \in S,\ \pi_i(s) \leftarrow \chi_i$
for $t = 1$ **to** E **do**
  select a_i^t according to $\pi_i(s_i^t)$
  observe r_i^t
  $s_i^{t+1} \leftarrow (a_i^t, r_i^t)$
  $r_i^{\max} \leftarrow \max(r_i^{\max}, r_i^t)$
end for
set α and γ
$\forall s \in S, a \in A_i,\ Q_i(s, \mathbf{a}) \leftarrow \frac{r_i^{\max}}{1-\gamma}$
repeat
  $t \leftarrow t+1$
  Select a_i^t according to $\pi_i(s_i^t)$
  Observe r_i^t
  $s_i^{t+1} \leftarrow (a_i^t, r_i^t)$
  Update $Q_i(s_i^t, a_i^t)$ according to Eq. (3)
  Update $\pi_i(s_i^t)$ according to Eq. (1)
until Game Over

Experimental Setup

To evaluate the effectiveness of R-lopars and other learning biases in games played with minimal information, we conducted a large empirical study. We now discuss the algorithms, games, and measurement techniques used in this study. Results of the study are presented in the next section.

Algorithms

Table 2 summarizes the algorithms included in our study along with the learning biases they encode. We selected these algorithms using two rules. First, only algorithms that did not require knowledge of associate's actions and payoffs were selected. Second, we selected algorithms to represent a range of goals and learning biases, while seeking to adequately represent popular learning biases (such as the rational learning bias). Where possible, we tried to use the parameter values used in the published literature.

Table 3: Selected games, scaled to payoffs between 0 and 1.

(a) Common Interest

	c	d
a	1.00, 1.00	0.00, 0.00
b	0.00, 0.00	0.50, 0.50

(b) Coordination

	c	d
a	1.00, 0.50	0.00, 0.00
b	0.00, 0.00	0.50, 1.00

(c) Stag hunt

	c	d
a	1.00, 1.00	0.00, 0.75
b	0.75, 0.00	0.50, 0.50

(d) Tricky Game

	c	d
a	0.00, 1.00	1.00, 0.67
b	0.33, 0.00	0.67, 0.33

(e) Prisoners' Dilemma

	c	d
a	0.60, 0.60	0.00, 1.00
b	1.00, 0.00	0.20, 0.20

(f) Battle of the Sexes

	c	d
a	0.00, 0.00	0.67, 1.00
b	1.00, 0.67	0.33, 0.33

(g) Chicken

	c	d
a	0.84, 0.84	0.33, 1.00
b	1.00, 0.33	0.00, 0.00

(h) Security Game

	c	d
a	0.84, 0.33	0.84, 0.00
b	0.00, 1.00	1.00, 0.67

(i) Offset Game

	c	d
a	0.00, 0.00	0.00, 1.00
b	1.00, 0.00	0.00, 0.00

(j) Matching Pennies

	c	d
a	1.00, 0.00	0.00, 1.00
b	0.00, 1.00	1.00, 0.00

(k) Shapley's Game

	d	e	f
a	0.0, 0.0	1.0, 0.0	0.0, 1.0
b	0.0, 1.0	0.0, 0.0	1.0, 0.0
c	1.0, 0.0	0.0, 1.0	0.0, 0.0

(l) Rock, Paper, Scissors

	d	e	f
a	0.5, 0.5	1.0, 0.0	0.0, 1.0
b	0.0, 1.0	0.5, 0.5	1.0, 0.0
c	1.0, 0.0	0.0, 1.0	0.5, 0.5

In addition to R-lopars, the algorithms used in the study included four algorithms that encode the rational learning bias (GIGA-WoLF, Exp3, WPL, and WoLF-PHC), two versions of satisficing learning (S-alg(1) and S-alg(2)), and five distinct Q-learning algorithms that do not require knowledge of associates' payoffs and actions.

Games

We evaluated the algorithms in the twelve two-player matrix games shown in Table 3. These games include constant-sum, common-interest, and conflicting-interest games. Most of these games have been studied repeatedly in the literature, as they test algorithms' abilities to compete, share profits, and bound losses. We also evaluated the algorithms in three sets of two-player randomly generated matrix games with between two and five actions. These random games consisted of 200 constant-sum games, 200 common-interest games, and 200 conflicting-interest games, respectively.

Evaluation

We conducted two kinds of tournaments for each matrix game. First, we conducted *round-robin tournaments* in which each algorithm was paired with every other algorithm and itself. Each pairing consisted of a repeated game lasting 300,000 rounds. An algorithm's performance was measured as its average payoff per round when paired with each of the 11 algorithms (averaged over 50 trials each).

Table 4: Summary of results for the selected games.

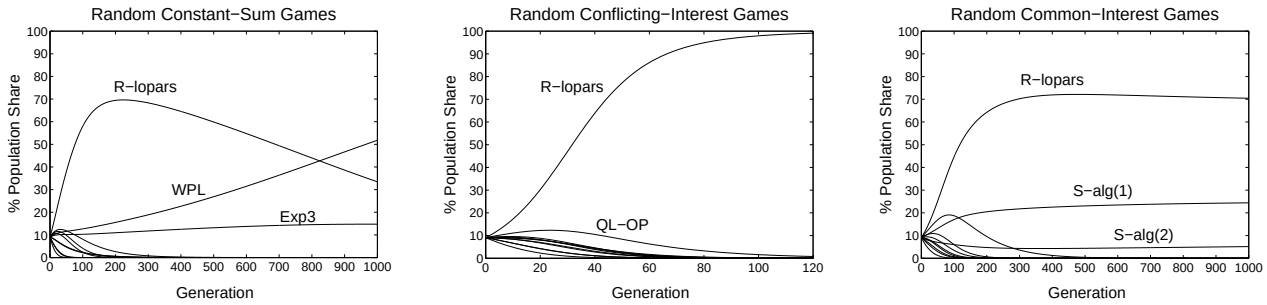
Game	Average Round-Robin Payoffs		Population after 1000 Generations	
Common Interest Game	1. Exp3	0.981	1. S-alg(2)	63.2%
	2. S-alg(2)	0.972	2. S-alg(1)	36.8%
	3. QL-OP	0.971		
Coordination Game	1. GIGA-WoLF	0.866	1. QL(0)	90.5%
	2. QL(0)	0.805	2. GIGA-WoLF	9.0%
	3. QL(1)	0.797	3. QL(2)	0.5%
Stag hunt	1. QL-OP	0.782	1. S-alg(2)	61.1%
	2. S-alg(1)	0.772	2. S-alg(1)	38.9%
	3. S-alg(2)	0.770		
Security Game	1. R-lopars	0.623		
	2. S-alg(1)	0.617	1. R-lopars	100%
	3. S-alg(2)	0.607		
Tricky Game	1. R-lopars	0.760	1. R-lopars	59.6%
	2. QL(2)	0.671	2. S-alg(1)	20.9%
	3. S-alg(2)	0.669	3. S-alg(2)	19.6%
Offset Game	1. R-lopars	0.244		
	2. GIGA-WoLF	0.193	1. GIGA-WoLF	100%
	3. QL(0)	0.158		
Prisoners' Dilemma	1. S-alg(2)	0.361		
	2. R-lopars	0.360	1. R-lopars	100%
	3. S-alg(1)	0.358		
Battle of the Sexes	1. GIGA-WoLF	0.918	1. GIGA-WoLF	58.6%
	2. QL(0)	0.880	2. QL(0)	41.2%
	3. QL-OP	0.863	3. R-lopars	0.2%
Chicken	1. S-alg(2)	0.830	1. S-alg(2)	63.4%
	2. Exp3	0.810	2. S-alg(1)	36.6%
	3. S-alg(1)	0.766		
Shapley's Game	1. R-lopars	0.517		
	2. QL-OP	0.462	1. R-lopars	100%
	3. QL(2)	0.460		
Matching Pennies	1. S-alg(2)	0.587	1. WPL*	71.4%
	2. R-lopars	0.547	2. WoLF-PHC*	11.1%
	3. QL(2)	0.526	3. QL(2)*	9.9%
Rock, Paper, Scissors	1. R-lopars	0.584	1. WPL*	54.7%
	2. QL-OP	0.529	2. WoLF-PHC*	37.8%
	3. QL(1)	0.529	3. R-lopars*	7.0%
Average	1. R-lopars	0.633	1. R-lopars	31.0%
	2. S-alg(2)	0.598	2. S-alg(2)	17.4%
	3. QL-OP	0.597	3. GIGA-WoLF	14.0%
	4. QL(2)	0.590	4. S-alg(1)	11.1%
	5. QL(1)	0.583	5. QL(0)	10.9%
	6. S-alg(1)	0.563	6. WPL	10.5%
	7. WPL	0.554	7. WoLF-PHC	4.1%
	8. GIGA-WoLF	0.552	8. QL(2)	0.9%
	9. QL(0)	0.546	9. Exp3	0.2%
	10. Exp3	0.540	T10. QL(1)	0.0%
	11. WoLF-PHC	0.531	T10. QL-OP	0.0%

* No convergence – average share over 50,000 generations given.

We also conducted *evolutionary tournaments* in which a large population of agents, each employing one of the 11 algorithms, was evolved over a series of generations according to the algorithms' fitnesses. Fitness was determined by randomly pairing each agent in the population with another agent in a 300,000-round repeated game. Initially, each algorithm was equally represented in the population. The proportion of the population employing each algorithm was then altered in each subsequent generation using the replicator dynamic (Taylor and Jonker 1978). Performance in these evolutionary tournaments was measured as the percentage of the population employing that algorithm after 1000 generations.

Results

The top performers in each of the games and tournaments are shown in Tables 4 and 5 and Figure 1. As expected, no one algorithm performed the best in all games. However, R-lopars performed the best on average. It had the highest average payoff per round in the round-robin tournaments in both the selected games and in each class of random games. Furthermore, it performed well in many individual games, finishing in the top two in seven of the twelve games. In evolutionary tournaments, it had the highest average popula-



(a) (b) (c)

Figure 1: Population share in evolutionary tournaments in random games.

Table 5: Average payoffs in random games.

Common Interest		Conflicting Interest		Constant Sum	
1. R-lopars	0.971	1. R-lopars	0.847	1. R-lopars	0.528
2. QL-OP	0.964	2. QL-OP	0.811	2. QL(2)	0.512
3. GIGA-WoLF	0.957	3. QL(1)	0.800	3. QL-OP	0.511
4. S-alg(1)	0.955	4. QL(0)	0.798	4. QL(1)	0.509
5. QL(0)	0.951	5. QL(2)	0.795	5. WPL	0.506
6. QL(1)	0.946	6. S-alg(2)	0.792	6. Exp3	0.504
7. Exp3	0.945	7. GIGA-WoLF	0.790	7. GIGA-WoLF	0.497
8. S-alg(2)	0.940	8. S-alg(1)	0.789	8. WoLF-PHC	0.497
9. WPL	0.938	9. WPL	0.772	9. S-alg(2)	0.485
10. QL(2)	0.933	10. Exp3	0.772	10. QL(0)	0.481
11. WoLF-PHC	0.917	11. WoLF-PHC	0.749	11. S-alg(1)	0.471

tion share after 1000 generations across the selected games, achieving an average population share of 31%. In random games, it obtained the highest population share in common- and conflicting-interest games, and the second highest population share in constant-sum games.

It is interesting to analyze what aspects of R-lopars's learning bias make it perform as it does. To do this, we compare its performance with that of other algorithms whose learning biases differ from it in some way. First, we compare it to QL-OP. The main difference between these two learning algorithms is their state representation. Both algorithms use recurrent state, but QL-OP's state is its previous action, while R-lopars uses its previous action and reward.

R-lopars's and QL-OP's average performance in the round-robin tournaments are both in the top three on average in both selected and random games (Tables 4 and 5). However, R-lopars substantially outperforms QL-OP in evolutionary tournaments due to its superior performance in self play (Figure 2). This result illustrates the importance of defining state with respect to the previous behaviors of all agents, which R-lopars encodes with its previous reward.

While defining state with something that reflects the past behavior of all agents is important, it is not the only aspect of R-lopars's learning bias that makes it successful. Both S-alg(1) and S-alg(2) also encode a recurrent state based on previous rewards. However, R-lopars's performance is much more robust than both S-alg(1) and S-alg(2). While these algorithms also perform very well in self play (Figure 2), they do not perform as well as R-lopars when associating with other types of algorithms (Figure 3). In this case, R-

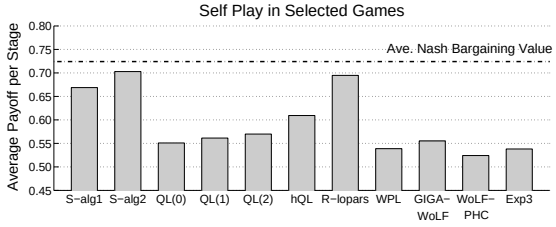


Figure 2: Average per round payoffs in self play.

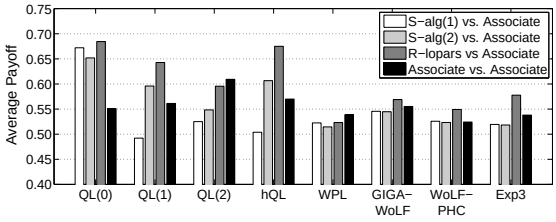


Figure 3: Average per round payoff in all selected games.

lopars's best response strategy rule and/or its use of future discounted rewards allow it to learn more effectively than the satisficing algorithms, which use a satisficing decision rule and only encode immediate rewards.

Tables 4 and 5 and Figure 1 also show the importance of setting optimistic priors in games played with minimal information. Except in random constant-sum games, the top two (and sometimes three) algorithms all set optimistic priors. A comparison of QL-OP and QL(1), who differ only in their priors, also illustrates this importance. In both selected and random games, QL-OP outperformed QL(1) on average.

Algorithms that encode the rational learning bias (GIGA-WoLF, WoLF-PHC, Exp3, and WPL) typically performed poorly in most of our tournaments. This suggests that a rational learning bias is not effective across a wide range of repeated matrix games played with minimal information. One exception to this latter observation is in constant-sum games. While R-lopars quickly gained a high population share in the random, constant-sum, evolutionary tournament (Figure 1(a)), WPL slowly gained an upper hand once weaker algorithms were eliminated from the population. Thus, in games played with minimal information, the rational learning bias appears to be somewhat successful in fully compet-

itive games, but not in other kinds of games.

Finally, any time conclusions are made using empirical results, the results are somewhat contingent on the parameter values that are used. Parameter values constitute yet another aspect of an algorithm's learning bias. One particularly interesting question is whether an agent is better off using a faster learning rate. To investigate this, we re-ran our round-robin tournaments with two versions of each algorithm, one with a fast learning rate, and the other with a slow learning rate. Results showed that, while the learning rate had a slight impact on performance in some individual games, there was no net benefit to having a faster learning rate (or vice versa). Thus, in our study, learning rate was not an important component of an algorithm's learning bias.

Conclusions and Discussion

In this paper, we have analyzed the role of learning bias in repeated matrix games played with minimal information (i.e., games in which the actions and payoffs of associates are unobservable). Specifically, we have dissected the learning biases of multi-agent learning algorithms published in the literature by comparing and contrasting the strategy selection rules, priors, belief models, state and reward representations, and strategy spaces used by these algorithms. Based on this analysis, we created a new algorithm, called R-lopars, that adapted a successful learning bias to minimal information environments. We then evaluated this algorithm and ten other learning algorithms in a large empirical study involving a variety of repeated two-player matrix games played with minimal information. Our study showed that, on average, R-lopars outperformed the other algorithms.

We also found that, among a representative set of algorithms, the most common learning bias for repeated matrix games combines a best response or gradient ascent strategy selection rule with (1) a stateless world representation, (2) a strategy space defined over all pure and mixed strategies, and (3) a random or undefined prior. We refer to this learning bias as the *rational learning bias* due to the game theoretic goals of the algorithms that tend to encode it. Despite its popularity, algorithms implementing the rational learning bias performed poorly in our study.

While these results are limited in nature, we believe that they illustrate an important point. Rather than focus solely on learning "rational," game theoretic, concepts such as no-regret, best response, and Nash equilibrium, we believe that designers of a multi-agent learning algorithm should pay more attention to the wider learning biases their algorithm encodes via its assumptions, representations, and rules. The success of an algorithm in repeated games played with minimal information often hinges on the sometimes trivialized aspects of an algorithm, including priors and state and reward representation. As a result, R-lopars outperformed the other algorithms in our study.

References

Abdallah, S., and Lesser, V. 2008. A multi-agent learning algorithm with non-linear dynamics. *Journal of Artificial Intelligence Research* 33:521–549.

Auer, P.; Cesa-Bianchi, N.; Freund, Y.; and Schapire, R. E. 1995. Gambling in a rigged casino: the adversarial multi-armed bandit problem. In *Proc. of the 36th Symp. on the Foundations of CS*, 322–331. IEEE Computer Society Press.

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

Bouzy, B., and Metivier, M. 2010. Multi-agent learning experiments in repeated matrix games. In *Proc. of the 27^{th} Intl. Conf. on Machine Learning*.

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

Bowling, M. 2005. Convergence and no-regret in multiagent learning. In *Advances in Neural Information Processing Systems 17*, 209–216.

Crandall, J. W., and Goodrich, M. A. 2011. Learning to compete, coordinate, and cooperate in repeated games using reinforcement learning. *Machine Learning* 83(3):281–314.

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

Hoen, P. J.; Tuyls, K.; Panait, L.; Luke, S.; and Poutre, J. A. L. 2006. Learning and adaptation in multi-agent systems. *Springer Berlin / Heidelberg* 1–46.

Hu, J., and Wellman, M. P. 1998. Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Proc. of the 15^{th} Intl. Conf. on Machine Learning*, 242–250.

Karandikar, R.; Mookherjee, D.; Ray, D.; and Vega-Redondo, F. 1998. Evolving aspirations and cooperation. *Journal of Economic Theory* 80:292–331.

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. 1994. Markov games as a framework for multi-agent reinforcement learning. In *Proc. of the 11^{th} Intl. Conf. on Machine Learning*, 157–163.

Moody, J.; Liu, Y.; Saffell, M.; and Youn, K. 2004. Stochastic direct reinforcement. In *AAAI Spring Symposium on Artificial Multiagent Learning*.

Sandholm, T. W., and Crites, R. H. 1996. Multiagent reinforcement learning in the iterated prisoner's dilemma. *Biosystems* 37:147–166.

Shoham, Y.; Powers, R.; and Grenager, T. 2007. If multi-agent learning is the answer, what is the question? *Artificial Intelligence* 171(7):365–377.

Stimpson, J. R., and Goodrich, M. A. 2003. Learning to cooperate in a social dilemma: A satisficing approach to bargaining. In *Proc. of the 20^{th} Intl. Conf. on Machine Learning*, 728–735.

Taylor, P. D., and Jonker, L. 1978. Evolutionarily stable strategies and game dynamics. *Mathematical Biosciences* 40:145–156.

Tesauro, G. 2004. Extending Q-learning to general adaptive multi-agent systems. In *Advances in Neural Information Processing Systems 16*. MIT Press.

Watkins, C. J., and Dayan, P. 1992. Q-learning. *Machine Learning* 8:279–292.