Department of Computer Science
George Mason University
Technical Report Series

4400 University Drive MS#4A5
Fairfax, VA 22030-4444 USA
http://cs.gmu.edu/
703-993-1530

Can Good Learners Always Compensate for Poor Learners?

Keith Sullivan ksulliv@cs.gmu.edu
Liviu Panait lpanait@cs.gmu.edu
Gabriel Balan gbalan@cs.gmu.edu
Sean Luke sean@cs.gmu.edu

Technical Report GMU-CS-TR-2005-3

Abstract

Can a good learner compensate for a poor learner when paired in a coordination game? Previous work has given an example where a special learning algorithm (FMQ) is capable of doing just that when paired with a specific less capable algorithm even in games which stump the poorer algorithm when paired with itself. In this paper, we argue that this result is not general. We give a straightforward extension to the coordination game in which FMQ cannot compensate for the lesser algorithm. We also provide other problematic pairings, and argue that another high-quality algorithm cannot do so either.

1 Introduction

Concurrent learning is a subset of cooperative multi-agent learning where the overall problem is divided into simpler subcomponents such that each agent explores its space of actions with little or no control over its teammate's actions [16]. While concurrent learners may converge to suboptimal solutions due to co-adaptation and other pathologies, many algorithms have attempted to solve these problems without a clear answer [3, 10, 12, 17, 21].

Concurrent learning introduces a new wrinkle to multiagent machine learning: what if learners used entirely different algorithms? This is not implausible: for example, an agent (on the web, say) may not have control over the other learning agents. But while recent research on learning in competitive games has addressed heterogeneous learners [2], the state of the art in *cooperative* scenarios still involves homogeneous learners.

We are aware of only one paper that focuses on concurrent heterogeneous cooperative learners: [11] analyze different combinations of a traditional reinforcement learning algorithm and an extension called FMQ. The authors report that a two agent team using the traditional algorithm cannot consistently learn the globally optimal solution; however, the optimal solution is achieved when both agents use FMQ. Moreover, an agent using FMQ can help another agent using the traditional algorithm to learn the global optimum in several coordination games. The authors thus conclude that a "smart" learner (FMQ) can still be successful even when it must work with less smart learners (the traditional algorithm).

We will argue this is not so. We extend the research in [11] by including a difficult coordination game, as well as two new algorithms. One of these algorithms is a modified evolutionary algorithm: this extends our analysis to combinations with agents that may use *entirely* different learning techniques, not just variations of the same learning method. The results of the experiments indicate the opposite conclusion from that in [11]: in difficult domains, good results are usually obtained only if both agents are "smart" enough. That is because it takes both agents to converge to an optimal solution, and poor learners may force "smart" ones to converge to suboptima.

This rest of this paper is organized as follows: Section 2 presents related work, followed by an introduction to single-stage coordination games in Section 3. Section 4 introduces the learning algorithms used for our analysis. The experiments and their results are detailed in Section 5. Finally, Section 6 offers some conclusions and directions for future research.

2 Related Work

Claus and Boutilier [3] show that a straightforward application of reinforcement learning (RL) to concurrent learning is not guaranteed to find the optimal solution, even in the case when agents are able to observe the other agents' actions. The authors then suggest the search could be improved by using more optimistic exploration actions. This direction is further explored in [12], who update the utilities of actions based in part on the maximum reward previously received when performing that action. Kapetanakis and Kudenko [10] observe that such biasing of utility computation may not work in domains where the joint reward information is noisy. They propose an improved multiagent reinforcement learning algorithm called FMQ that uses the maximum reward received per action to bias the probability of choosing that action. This improved algorithm shows advantages in domains with limited amounts of noise, but crucially its performance is poor when there is a lot of noise. In [11], they apply FMQ to a concurrent environment, showing that FMQ can compensate for a simple Q-learner in environments where the Q-learner performs poorly. Finally, Verbeeck et al [21, 22] propose coordinated restarts of suboptimal learning algorithms in combination with action exclusions (similar to tabu search) to guarantee convergence to the globally optimal solution. But the restarts may require a significant amount of time, and the convergence to optima is guaranteed only if all Nash equilibria are visited infinitely often.

Evolutionary computation (EC) has also been applied to multiagent learning in domains such as RoboCup [14], iterated prisoner's dilemma [1], single stage cooperative games [19] and predator prey pursuit [5]. Cooperative coevolutionary algorithms (CCEAs) [7, 18] are a common approach to concurrent evolutionary learning agents. Weigand analyzed the conditions under which coevolutionary systems gravitate towards suboptimal solutions [23]. Panait et al [17] show that biasing the CCEA to search for maximal rewards improves performance on three problem domains, including two games: Climbing and Penalty, which we discuss in the next section.

3 Single-Stage Cooperative Games

Markov decision processes (MDPs) are widely used in multiagent reinforcement learning to account for the presence of other agents in the environment [6, 13]. Single-stage cooperative games[1] are a variation of MDPs where all agents receive the same reward [3]. More specifically, each agent independently chooses an action from its action set, and the actions from all the agents are combined into a *joint action*. All agents receive the same reward or penalty depending on the joint action. This process is repeated until (hopefully) agents learn to select better actions due to past interactions. We assume that agents do not explicitly communicate or observe teammates' actions; the only feedback mechanism is the reward received for the agent's action.

[1]Sometimes referred to as common interest games, cooperative games, or coordination games.

Table 1 shows four single-stage cooperative games: the Climbing game and the Penalty game introduced in [3], and partially stochastic and the fully stochastic variations of the Climbing game as proposed in [10]. Researchers have used these games extensively to highlight the advantages of certain multiagent learning algorithms (for example, [3, 10, 12, 17]). In the regular, partially stochastic, and fully stochastic Climbing games, the optimal joint action is (a,a), while the Penalty game has two optimal joint actions, (a,a) and (c,c); we do not distinguish between these two global optima in the Penalty game. Note that in the partially stochastic and fully stochastic games, the average reward for a joint action is the same as the plain reward for that same action in the regular Climbing game.

Each of these games is challenging due to miscoordination penalties. The Climbing game has a severe penalty for choosing action *a* when the other agent chooses action *b*. However, there are no major miscoordination penalties associated with action *c*, potentially making it tempting for the agents. The Penalty game introduces another miscoordination issue due to the presence of multiple optimal joint actions. Simply choosing the optimal action is no guarantee that the other agent will choose the same optimal action. If agents decide on a third equilibrium, (b,b), they avoid low rewards associated with miscoordination.

The stochastic variations of the Climbing game add more complications due to the noisy reward function. As agents cannot perceive their teammates' actions, the different rewards they observe for the same action may be due to either (1) the other agent experimenting with multiple actions, or (2) the noisy reward function. While the highest reward of 14 is sometimes achieved when both agents choose action *b*, they need to learn to separate the effects of (1) and (2), and to realize that the average reward for the joint action (b,b) is lower than that of (a,a).

4 Learning Algorithms

We experiment with four learning algorithms: three variations of reinforcement learning, and a genetic algorithm with a novel evaluation procedure. Two of these algo-

		Agent 2		
		a	b	c
Agent 1	a	11	-30	0
	b	-30	7	6
	c	0	0	5

(a)

		Agent 2		
		a	b	c
Agent 1	a	10	0	*k*
	b	0	2	0
	c	*k*	0	10

(b)

		Agent 2		
		a	b	c
Agent 1	a	11	-30	0
	b	-30	14/0	6
	c	0	0	5

(c)

		Agent 2		
		a	b	c
Agent 1	a	10/12	5/-65	8/-8
	b	5/-65	14/0	12/0
	c	5/-5	5/-5	10/0

(d)

Table 1: Joint reward matrices for the Climbing Game (a), Penalty Game (b), Partially Stochastic Game (c), and the Fully Stochastic Game (d). In the stochastic games, the first reward is returned with probability p, and the second reward is returned with probability $1-p$.

rithms are not new: reinforcement learning and FMQ have been previously analyzed, in particular in [10]. We chose reinforcement learning as a standard benchmark, while FMQ appears to be one of the stronger algorithms to date. We devised the other two algorithms based on the notion of lenience of an agent towards its teammates.

4.1 Reinforcement learning

Reinforcement learning techniques are concerned with maximizing an agent's reward as it interacts with complex and possibly unknown environments [20]. RL takes inspiration from dynamic programming to define formulas to update the utility of performing actions while in various states. In the case of only *one* state (as in the Climbing and Penalty games), [11] reduces RL's update equation to:

$$U(\alpha_i) = \lambda * U(\alpha_i) + (1-\lambda) * R(\alpha_i) \qquad (1)$$

α_i is the agent's chosen action, $R(\alpha_i)$ the reward it receives, $U(\alpha_i)$ is the utility of action α_i, and $0 \leq \lambda \leq 1$ is the learning rate.

Agents employ Boltzman action selection to balance between exploration of alternatives and exploitation of actions with higher utility [8]. Each action α_i is chosen with a probability based on the estimated utility of that action:

$$P(\alpha_i) = \frac{e^{U(\alpha_i)/T}}{\sum_{\alpha'} e^{U(\alpha')/T}}$$

where T is a temperature parameter that starts high and decreases with time. High temperatures allow for more exploration as the contribution of the utility estimate is minimized, while very low temperatures result in the agent always selecting its better action.

4.2 FMQ Heuristic

The Frequency Maximum Q-Value (FMQ) heuristic is an extension of reinforcement learning that alters the probabilities of selecting actions [9]. The algorithm uses an optimistic estimation (EV) for the utility of an action, defined as

$$EV(\alpha) = U(\alpha) + c * freq(\max(R(\alpha))) * \max(R(\alpha))$$

where $max(R(\alpha))$ is the maximum reward received *so far* for choosing action α, $freq(\max(R(\alpha)))$ is the fraction of times that $max(R(\alpha))$ has been received over all the times that action α was executed, and c is a weight that controls the importance of the FMQ heuristic in the action selection. An agent learning using the FMQ heuristic chooses his actions according to the Boltzman action selection procedure updated to use the optimistic evaluations of utilities:

$$P(\alpha_i) = \frac{e^{EV(\alpha_i)/T}}{\sum_{\alpha'} e^{EV(\alpha')/T}}$$

If the two FMQ agents miscoordinate, not only will they receive a suboptimal reward, but they also decrease the frequency of observing the maximum reward for the actions they have chosen. This decreases the likelihood that that action will be selected in the future.

4.3 Lenient Multiagent Reinforcement Learning

In an accompanying paper, we propose another RL algorithm , the Lenient Multiagent Reinforcement Learning (LMRL), which selectively updates the utilities of actions based on *some* of the rewards. It is implemented as follows. We always update the utility of the action if the current reward exceeds the utility of the action. Otherwise, the utility is update with a probability based on a current per-action *temperature*. If the temperature associated with an action is high, then agent is "lenient" towards low-reward pairings and so it does not update its utility to reflect them. At a lower temperature, low-reward pairings are added to the utility with greater frequency.

The temperature of an action is decreased slightly every time that action is selected. As a consequence, actions that have been chosen more often have their utilities updated more often as well, while the utilities for actions that have been chosen rarely are mainly updated in response to higher rewards. This initially leads to an overoptimistic evaluation of the utility of an action. An agent may thus be temporarily fooled into choosing suboptimal actions. However, the utilities of such actions will decrease with time, and the agent is more likely to end up choosing the optimal action. There is also a small (0.01) probability of ignoring small rewards at all times: we found this to work in our experimental setup because the agents have non-zero probabilities of selecting an action at each time step.

Aside from these enhancements, the algorithm follows a traditional RL approach, including the Boltzman action selection based on the utility of each actions. The pseudocode for the algorithm is as follows:

```
Lenient Multiagent RL
Parameters
  $MaxTemp$: maximum temperature
  $\alpha$: temperature multiplication coefficient
  $\beta$: exponent coefficient
  $\delta$: temperature decay coefficient
  $\lambda$: learning rate
  $N$: number of actions
Initial Settings
  For each action $i$
    $U_i = random() * 0.001$
    $Temp_i = MaxTemp$
Algorithm
Repeat

  // Action Selection
  $T = 10^{-6} + \min_{i=1}^{N} Temp_i$
  $W_i = \frac{e^{(U_i/T)}}{\sum_{j=1}^{N} e^{(U_j/T)}}$
  Use probability distribution $W_1, ..., W_N$
    to select action $i$
  $Temp_i = Temp_i * \delta$

  // Utility Update
  Perform action $i$ and observe reward $r$
  If $(U_i \leq r)$ or
    $\left(random() < 10^{-2} + \beta^{-\alpha * Temp_i}\right)$ Then
    $U_i = \lambda * U_i + (1-\lambda) * r$
```

4.4 Lenient Evolutionary Algorithm

Evolutionary algorithms (EAs) are stochastic search techniques inspired by natural evolution [4]. EAs maintain a set of samples in the search space (typically referred to as a "population" of "individuals"). Each such individual is assigned a "fitness" (the quality of the individual). EAs then form a new population by repeatedly selecting, copying and modifying the highly fit individuals in the current population. The new population replaces the old one, and the cycle of fitness assessment and breeding continues until a termination criterion is met. Each iteration of this cycle is known as a "generation." See [4, 15] for a more detailed discussion of evolutionary algorithms.

Cooperative Coevolutionary Algorithms (CCEAs) apply EAs to concurrent learning processes. A CCEA employs not one but multiple (for our purposes, two) populations, each evolving independently. CCEAs team up populations to evaluate them together. We propose a new EA that, like CCEAs, evaluates individuals (representing actions) in combination with actions not just the other populations, but from other learning algorithms (such as RL or another EA).

This EA also allows an agent (individual) to show varying degrees of lenience to its teammate. For this purpose, the entire population is cloned multiple times and shuffled, and each clone is evaluated with an action chosen by the teammates. As the teammate is also co-adapting, the particular action it chooses may change at any time. The extra clones of the population are required only for evaluations and may be discarded afterwards. After each clone of an action receives multiple rewards, these rewards are aggregated into a single fitness for the original action. This fitness is computed as the average of the better K rewards that were obtained. Varying degrees of lenience are implemented by using the average of fewer of the better rewards at early generations (thus ignoring more of the rewards observed for an action). Initially, the best reward is used. As learning progresses, we compute the fitness as the average of more and more "top" rewards, thus reducing the lenience towards the teammate.

```
Fitness Assessment
Parameters
  $C$ : number of clones
  $K$ : lenience parameter ($K \leq C$)
Clone population $C$ times and perform shuffling
For each cloned action
  Agent chooses that action and receives a reward
For each action $i$
  $Fitness(i)$ = average of better $K$ rewards
    received by $i$'s clones
```

5 Experimental Results

We ran each combination of learning algorithms in the four cooperative games. Experiments consisted of 30 trials of 1000 runs each. Each run lasted for 7500 joint action selections and their rewards. We set $k = -10$ in the Penalty game. For all the reinforcement learning algorithms, $T = T * 0.9995$ and $\lambda = 0.95$. For FMQ, we set $c = 10$; LMRL used $\alpha = 2$ and $\beta = 2.5$. For the lenient evolutionary algorithm, we used a population of size 10, and the evolution lasted for 150 generations; we also set $C = 5$ clones per population, and K started at 1 and increased every 15 generations until reaching the maximum value of 5. The lenient EA employed selection via binary tournament, and breeding randomized the action with probability 0.001.

Tables 2, 3, 4, and 5 show the results for the Climbing and the Penalty domains, as well as for the partially and fully stochastic versions of the Climbing domain, respectively. The tables are symmetric — the values under the main diagonal are left blank for increased clarity. At the end of each run we determined if the converged joint action was optimal. We then computed the number of runs (out of 1000) that converged to the joint action, and averaged over 30 trials.

In the Climbing game (Table 2), all but three learning teams discovered the optimal action more than 91% of the time. The reasons: RL-RL's use of all rewards caused it to be attracted occasionally to (b,c). RL-LMRL in turn was attracted to (c,c) and EA-EA to (b,b). Overall, FMQ and LMRL were the best learners in the Climbing domain, and they also did well when combined with one another. FMQ had a slight edge over LMRL as it also performed very well when teamed with RL.

All teams discovered the optimal strategy more than 98% of time in the Penalty game (Table 3). This improvement in performance was partly due to the fact that the Penalty game has twice the number of optimal solutions to which learning may converge.

The results for the partially stochastic Climbing game (Table 4) were similar (although more extreme) to the ones in the deterministic Climbing game from Table 2. FMQ and LMRL were again the "smarter" algorithms: they found the optimal joint action in almost all runs when paired with themselves or each other. FMQ was also able to help RL converge to the optimum, but LMRL could achieve that in almost any run. The EA algorithm deteriorated significantly across the board.

So far, FMQ has done well when partnered with most other algorithms. But Table 5 shows a different result. Only one pair (LMRL-LMRL) efficiently solved this problem. But more importantly, *every method but EA did best by far when paired with itself.* In general, heterogeneous parings performed very poorly! Notably,

	RL	FMQ	LMRL	EA
RL	462.7	999.8	468.9	913.5
FMQ	–	999.9	999.9	910.2
LMRL	–	–	991.6	913.4
EA	–	–	–	837.3

Table 2: Average number of iterations (out of 1000) that converged to the joint action for the Climbing game.

	RL	FMQ	LMRL	EA
RL	999.8	1000.0	999.5	998.2
FMQ	–	1000.0	1000.0	998.3
LMRL	–	–	1000.0	997.4
EA	–	–	–	985.3

Table 3: Average number of iterations (out of 1000) that converged to the joint action for the Penalty game.

while FMQ helped RL in the partially stochastic game, in the fully stochastic game FMQ *hinders* RL compared to RL-RL (and for that matter, FMQ-FMQ). That is to say, the claim in [11] that the "smarter" FMQ algorithm can help poorer RL find the optima must change to "FMQ hinders RL's attempts to converge to the optimum for this problem domain (and vice-versa)."

We finish with illustrations of miscoordination. Figure 1 shows an example run from the generally failed pairing of LMRL and RL in the partially stochastic game. RL incorporates all rewards into the utility estimates, which results in early convergence to low utilities for both the a and b actions, and a preference for c throughout the search. LMRL is initially optimistic about a and especially about b, but if cannot recover from RL's lock onto c and it eventually settles on c itself.

In Figure 2, LMRL in the fully stochastic game is similarly unable to recover from FMQ's premature convergence to low (optimistic) estimations for the utilities of both a and b. Compare this to the typical successful pairing of LMRL with itself in this game (Figure 3). Here, both agents explore the space long enough that, after recovering from initial over-optimism due to exaggerated lenience, they correctly identify the true utility of the optimal joint action (a,a).

6 Conclusions

We extended the work in [11] to include two new learning algorithms that exhibit lenience toward teammates, as well as a difficult coordination game characterized by stochastic rewards for every joint action. The experiments indicate that only the lenient multiagent reinforcement learning algorithm can achieve near-optimal perfor-

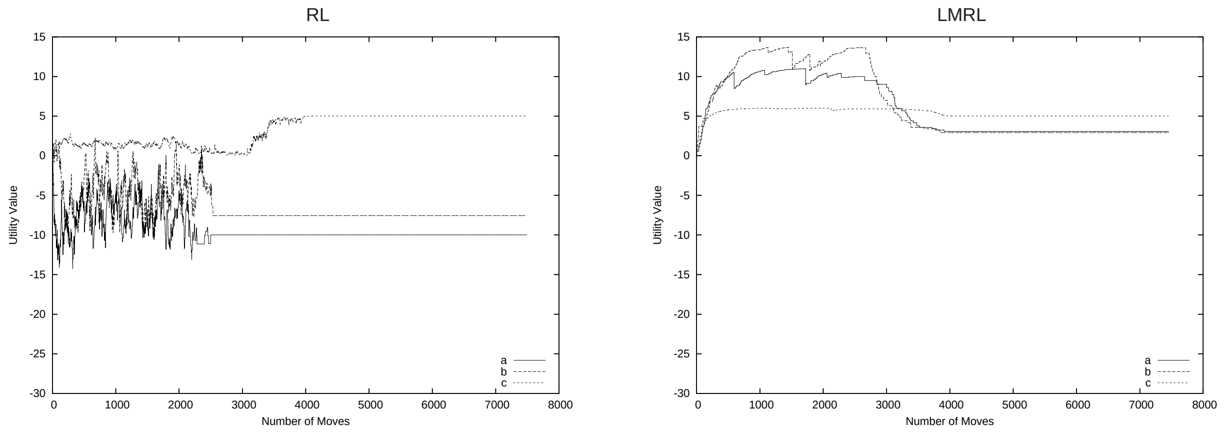

Figure 1: Utility of each agent's three actions for a pair of RL-LMRL learners in an instance of the partially stochastic version of the Climbing domain. The team of agents converges to the suboptimal joint action (c,c) which avoids miscoordination penalties; each agent's estimated utility for action c reflects the correct reward of 5 for this joint action.

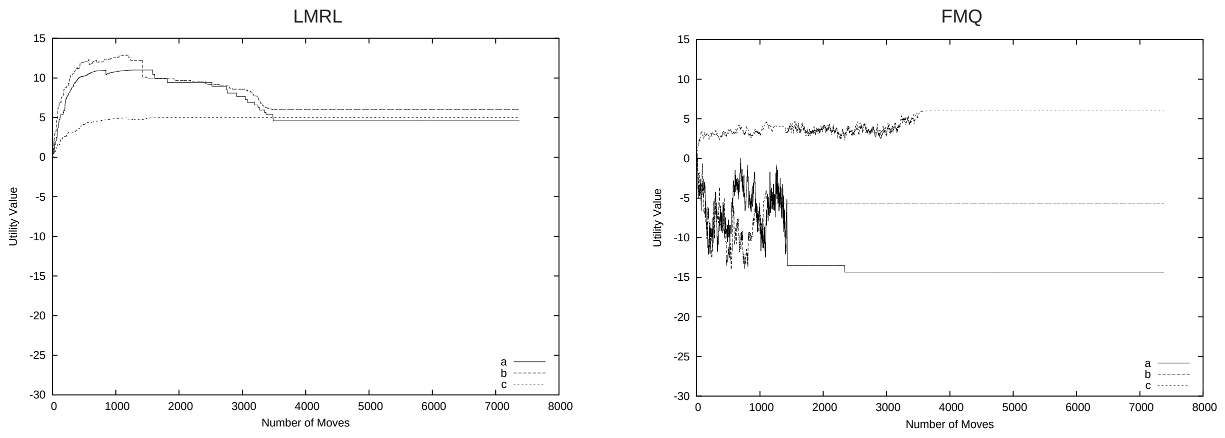

Figure 2: Utility of each agent's three actions for a pair of LMRL-LMRL learners in an instance of the partially stochastic version of the Climbing domain. LMRL starts with optimistic estimations for the utilities of the actions; as lenience toward the teammate is decreased, so do some of the utilities. Ultimately, the two learning algorithms converge to the (b,c) joint action. Both agents have an accurate estimate of 6 for the utility of this joint action.

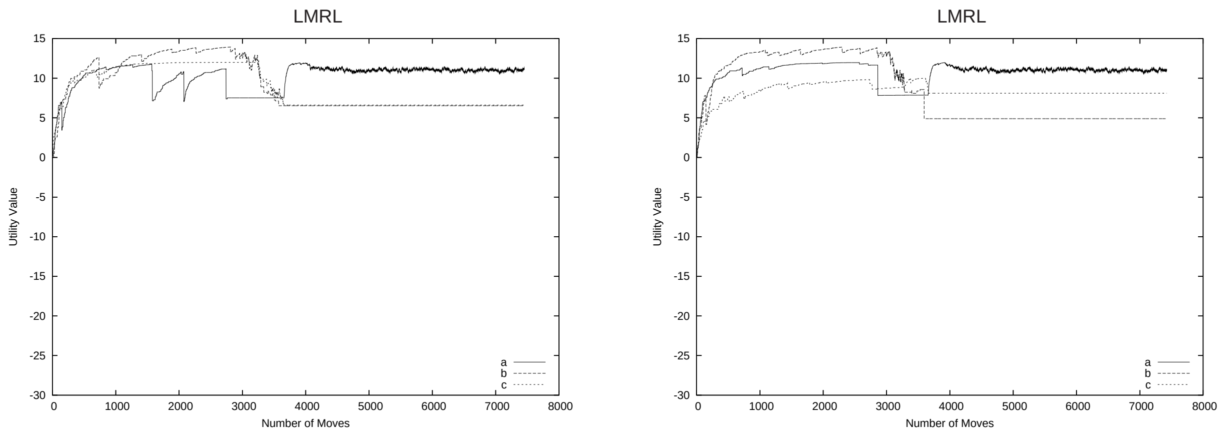

Figure 3: Utility of each agent's three actions for a pair of LMRL-LMRL learners in an instance of the fully stochastic version of the Climbing domain. The team of agents converges to the optimal joint action (a,a); each agent's estimated utility for action a reflects the correct reward of 11 for this joint action.

	RL	FMQ	LMRL	EA
RL	468.5	998.4	0.8	218.5
FMQ	–	999.8	998.1	419.9
LMRL	–	–	998.0	214.7
EA	–	–	–	303.8

Table 4: Average number of iterations (out of 1000) that converged to the joint action for the partially stochastic game.

	RL	FMQ	LMRL	EA
RL	464.4	3.3	2.6	230.9
FMQ	–	141.5	4.3	108.0
LMRL	–	–	928.5	235.1
EA	–	–	–	197.0

Table 5: Average number of iterations (out of 1000) that converged to the joint action for the stochastic game.

mance, and only when paired with itself. A good learner cannot compensate if its teammate converges to a suboptimal action. Contrary to the findings in [11], we find that a pair of traditional RL algorithms performs better than both an FMQ-FMQ pair, as well as a combination of an FMQ and an RL learner. It is *not* the case that FMQ, or even the often-better LMRL, can compensate for a mismatched teammate algorithm.

The issue that remains is: are there well-defined classes of problems and subsets of learning algorithms for the teammate with which a given learning algorithms works well? For example, exactly where and why does FMQ pair well with various learners? This is not an easy question to answer, but investigation along these lines may reveal more clues about the relationship classes of algorithms have with one another, and why some features of algorithms may be at odds with features of other algorithms. Until then we must recommend a homogeneous approach to multiagent learning, if given the choice.

References

[1] R. Axelrod. The evolution of strategies in the iterated prisoner's dilemma. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, pages 32 – 41. Morgan Kaufman, 1987.

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

[3] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of National Conference on Artificial IntelligenceAAAI/IAAI*, pages 746–752, 1998.

[4] K. De Jong. *Evolutionary Computation: A unified approach*. MIT Press, 2005.

[5] T. Haynes, S. Sen, D. Schoelnefeld, and R. Wainwright. Evolving multiagent coordination strategies with genetic programming. *Artificial Intelligence*, 1995.

[6] J. Hu and M. Wellman. Multiagent reinforcement learning: theoretical framework and an algorithm. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 242–250. Morgan Kaufmann, San Francisco, CA, 1998.

[7] P. Husbands and F. Mill. Simulated coevolution as the mechanism for emergent planning and scheduling. In R. Belew and L. Booker, editors, *Proceedings of the Fourch International Conference on Genetic Algorithms*, pages 264–270. Morgan Kaufmann, 1991.

[8] L. P. Kaelbling, M. L. Littman, and A. W. Morre. Reinforcement learning: A survey. *Journal of Artificial Intelligent Research*, 4, 1996.

[9] S. Kapetanakis and D. Kudenko. Improving on the reinforcement learning of coordination in cooperative multi-agent systems. In *Proceedings of the Second Symposium on Adaptive Agents and Multi-agent Systems*, 2002.

[10] S. Kapetanakis and D. Kudenko. Reinforcement learning of coordination in cooperative multi-agent systems. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI02)*, 2002.

[11] S. Kapetanakis and D. Kudenko. Reinforcement learning of coordination in heterogeneous cooperative multi-agent systems. In *Proceedings of the Third Autonomous Agents and Multi-Agent Systems Conference (AAMAS 2004)*, 2004.

[12] M. Lauer and M. Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 535–542. Morgan Kaufmann, San Francisco, CA, 2000.

[13] M. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the 11th International Conference on Machine Learning (ML-94)*, pages 157–163, New Brunswick, NJ, 1994. Morgan Kaufmann.

[14] S. Luke, C. Hohn, J. Farris, G. Jackson, and J. Hendler. Co-evolving soccer softbot team coordination with genetic programming. In *RoboCup-97: Robot Soccer World Cup I (Lecture Notes in AI No. 1395)*, pages 398–411, Berlin, 1998. Springer-Verlag.

[15] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 2001.

[16] L. Panait and S. Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3), 2005.

[17] L. Panait, R. P. Wiegand, and S. Luke. Improving coevolutionary search for optimal multiagent behaviors. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 653–658, Acapulco, Mexico, 2003. Morgan Kaufmann.

[18] M. Potter. *The Design and Analysis of a Computational Model of Cooperative CoEvolution*. PhD thesis, George Mason University, Fairfax, Virginia, 1997.

[19] T. Riechmann. Genetic algorithm learning and evolutionary games. *Journal of Economic Dynamics & Control*, 25:1019 – 1037, 2001.

[20] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

[21] K. Verbeeck, A. Nowe, M. Peeters, and K. Tuyls. Multi-agent reinforcement learning in stochastic single and multi-stage games. In D. Kudenko, D. Kazakov, and E. Alonso, editors, *Adaptive Agents and Multi-agent Systems III: Adaption and Multi-Agent Learning*, Lecture Notes in Computer Science, pages 275 – 294. Springer-Verlag, 2005.

[22] K. Verbeeck, A. Nowe, and K. Tuyls. Coordinated exploration in stochastic common interest games. In *Proceedings of Third Symposium on Adaptive Agents and Multi-agent Systems*, pages 97 – 102, 2003.

[23] R. P. Wiegand. *Analysis of Cooperative Coevolutionary Algorithms*. PhD thesis, Department of Computer Science, George Mason University, 2003.