Can Good Learners Always Compensate for Poor Learners?

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

Department of Computer Science, George Mason University
4400 University Dr, MSN 4A5, Fairfax, VA 22030 USA

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.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent Systems

General Terms

Algorithms

Keywords

Reinforcement Learning, Multiagent Systems, Cooperative Games

1. INTRODUCTION

Concurrent learning is a subset of cooperative multiagent 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. 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 [1], 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: Kapetanakis and Kudenko analyze different combinations of a traditional reinforcement learning algorithm and an extension called FMQ [6]. 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 that this is not so. We extend the research in [6] by including a difficult coordination game, as well as two new algorithms. One of these algorithms is a modified evolutionary algorithm: this extends the analysis to combinations with agents that 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 [6]: 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.

2. 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 [4, 8]. Single-stage cooperative games[1] are a variation of MDPs where all agents receive the same reward [2]. 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. For example, using the Climbing game as shown in Table 1(a), if agent 1 chooses action b and agent 2 choose action a, then both agents receive a reward of -30. This process of choosing a joint action is repeated until (hopefully) the 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.

Table 1 shows four single-stage cooperative games: the Climbing game and the Penalty game introduced in [2], and the partially- and fully-stochastic variations of the Climbing game as proposed in [5]. Note that in the partially- and fully-stochastic games, when

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

		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 $1-p$.

$p = 0.5$, the average reward for a joint action is the same as the reward for the same joint action in the regular Climbing game. In the regular, partially- and fully-stochastic Climbing games, the optimal joint action is (a,a). In the Penalty game, the known penalty, $k \leq 0$, results in three optimal joint actions – (a,a), (b,b), (c,c) – of which (a,a) and (c,c) are preferred since they have a higher reward. Researchers have used these games extensively to highlight the advantages of certain multiagent learning algorithms (for example, [2, 5, 7]).

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 tempting the agents. The Penalty game introduces an additional miscoordination issue due to the presence of multiple optimal joint actions. While smaller values of *k* will make the individual actions *a* and *c* more attractive, there is still the problem of ensuring that the other agent will choose the same optimal action.

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).

3. 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 algorithms are not new: reinforcement learning and FMQ have been previously analyzed, in particular in [5]. 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.

3.1 Lenient Multiagent Reinforcement Learning

In an accompanying paper, we propose another RL algorithm, 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 included in 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 be important 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.

3.2 Lenient Evolutionary Algorithm

Evolutionary algorithms (EAs) are stochastic search techniques inspired by natural evolution [3]. 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 [3] 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.

	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 2: 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 3: Average number of iterations (out of 1000) that converged to the joint action for the stochastic game.

4. 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 the learning rate is 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. 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, 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. 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 2) were similar (although more extreme) to the ones in the deterministic Climbing game. 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 3 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, 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 [6] that the "smarter" FMQ algorithm can help poorer RL find the optima must change to "FMQ helps RL to converge to the optimum *for this problem domain*, but hinders it in others."

5. CONCLUSIONS

We extended the work in [6] 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 performance, and only when paired with itself. A good learner cannot compensate if its teammate converges to a suboptimal action. Contrary to the findings in [6], 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.

6. REFERENCES

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

[2] 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.

[3] K. De Jong. *Evolutionary Computation: A unified approach*. MIT Press, 2006.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.