Lenient Learners in Cooperative Multiagent Systems

Liviu Panait
lpanait@cs.gmu.edu

Keith Sullivan
ksulliv@cs.gmu.edu

Sean Luke
sean@cs.gmu.edu

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

ABSTRACT

In concurrent learning algorithms, an agent's perception of the joint search space depends on the actions currently chosen by the other agents. These perceptions change as each agent's action selection is influenced by its learning. We observe that agents that show lenience to their teammates achieve more accurate perceptions of the overall learning task. Additionally, lenience appears more beneficial at early stages of learning, when the agent's teammates are merely exploring their actions, and less helpful as the agents start to converge. We propose two multiagent learning algorithms where agents exhibit a variable degree of lenience, and we demonstrate their advantages in several coordination problems.

Categories and Subject Descriptors

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

General Terms

Experimentation

Keywords

Multiagent Learning, Cooperation, Coordination

1. INTRODUCTION

This paper focuses on applications where multiple agents concurrently learn how to better interact with one another. Imagine a simple scenario where two agents learn to coordinate. The task is for the agents to each independently choose one action with the goal of maximizing the joint reward that they receive. Figure 1 illustrates a search space of joint rewards for a simple two-agent domain. The figure shows two peaks of different sizes. The lower peak represents a globally-suboptimal solution, and the wide coverage of that peak implies that solution quality changes only a little when either agent chooses a slightly different action. The higher peak represents the global optimum; its smaller coverage implies that the solution quality may change rapidly if either agent chooses a slightly different action.

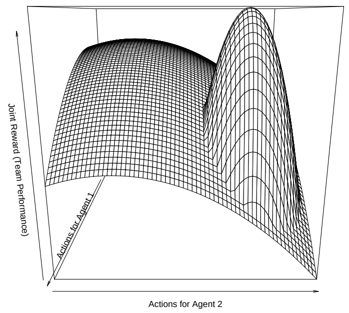

Figure 1: A bimodal search space for the possible rewards received by a two-agent team. Wider peaks may attract many search trajectories, even though such peaks may be globally suboptimal.

In concurrent multiagent learning, each agent is usually afforded only a partial glimpse of the search space. Specifically, each agent can only detect the rewards it receives for its own actions. However, these rewards also depend on the actions chosen by its teammate, and as a consequence, an agent's perception of the search space depends on what actions its teammate is currently choosing.

An agent's perception of the joint space is additionally affected by the approach it uses to aggregate the rewards obtained for each of its actions (when combined with different actions for the teammate). For example, Figure 2 shows different perceptions of the joint search space in Figure 1 for two aggregation methods. We observe that if an agent estimates the quality of its action as the *average* reward obtained when combined with multiple actions chosen by its teammate, the agent fails to perceive the potential of actions corresponding with the globally optimal peak. On the other hand, an agent that shows *lenience* towards its teammate by assessing the quality of its actions as the *maximum* reward received over multiple actions chosen by its teammate (thus ignoring the lower rewards usually associated with miscoordinations) perceives the higher potential of actions corresponding to the global optimum. If the goal is to discover globally optimal solutions, the agents should be able to tell that actions associated with such solutions are superior to others. In this paper, we focus on lenient learning as an approach to doing this.

We are particularly interested in *varying* the degree of lenience as time passes. As a consequence, we propose two algorithms that decrease the learners' degree of lenience with time. There are several reasons to vary the degree of lenience an agent exhibits. While an extremely high degree of lenience may be useful at early stages of learning to identify promising actions, ignoring many rewards may also be wasteful (in terms of evaluations) at later stages of learning once the agents start to converge. Additionally, ignoring rewards

AAMAS'06 May 8–12 2006, Hakodate, Hokkaido, Japan.

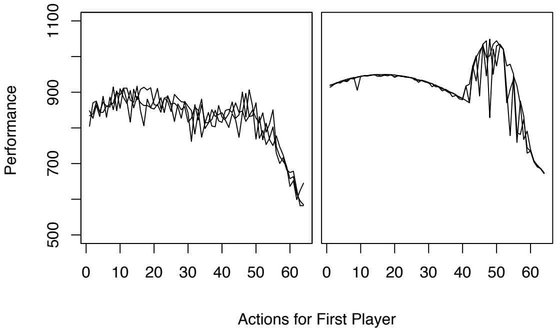

Figure 2: Two possible perceptions of the domain in Figure 1, for random actions of the teammate. (left) The average of ten joint rewards. (right) The maximum of ten joint rewards.

as the agents start to converge may be harmful to estimating the actual expected value in domains where agents receive noisy reward information. Finally, lenience may interfere with the learners' attempts to choose one of multiple equally-good solutions, if several exist.

2. LENIENT COEVOLUTION

Cooperative coevolutionary systems [6] are variants of evolutionary computation (a stochastic optimization technique) which apply multiple parallel learners (optimization processes) to work jointly on different aspects of the problem. This makes them a good fit for multiagent learning by decomposing the joint problem into several, likely intertwined, individual agent subproblems.

A standard approach to applying cooperative coevolutionary algorithms (CCEAs) to multiagent learning assigns each agent its own set (*population*) of higher-rewarded actions. The quality (*fitness*) of an action is determined by testing it in combination with actions from the other agent (as sampled from its current population either randomly or based on their performance during the past evaluation phase). When combined with these actions (called *collaborators* in coevolution parlance), an agent's action receives a reward equal to the joint reward for the entire team of agents. The fitness of the candidate action is then computed by aggregating multiple such rewards (e.g., by taking the average or the maximum, similar to the process used for Figure 2). Aside from this collaborative assessment, each agent follows its own independent evolutionary process in parallel with other agents.

Most cooperative coevolutionary algorithms assume that each action is evaluated as the maximum joint reward obtained with a fixed number of collaborators [8]; of them, one is usually the best action at the previous learning stage (*generation*), and the others are chosen at random. In terms of our earlier discussion, using the maximum joint reward translates into a constant level of lenience: the quality of the action is assessed by ignoring lower rewards. Given the observed variance in the impact that lenience can have on an agent's perception, we argue that a varying degree of lenience might be beneficial for CCEAs, especially when given fixed computation capabilities.

To do this, we experiment with a trivial ad-hoc setting: the fitness of each individual is assessed as the maximum reward obtained with ten collaborators for the first five generations; after that, only the maximum reward with *two* collaborators is used. This implies that the degree of lenience varies with time: the worst nine out of ten rewards will be ignored for each action in an agent's population during the first five generations, while only one out of two rewards will be ignored thereafter. While we found this trivial setting to be useful for this proof of concept investigation, readers with further interest in time-dependent collaboration schemes for CCEAs are referred to [5].

Experiments. To test our hypothesis that a variable degree of lenience is helpful, we applied CCEAs to four simple coordination games based on benchmark optimization problems with well-known properties: Two-Peaks (as in Figure 1), Rosenbrock, Griewangk, and Booth [7]. All domains were discretized such that each of the two agents had a set of 1024 actions. Each agent maintained a population of 32 actions. Agents kept unmodified their best action from one generation to another, and the remaining population of actions was created by mutating actions chosen via tournament selection of size 2 (two random actions were picked with replacement from the population, and the fitter of the two was selected). Mutation worked as follows: a coin was repeatedly tossed, and the action (an integer number) was increased or decreased (the direction chosen at random beforehand) until the coin came up heads, making sure it did not go outside the allowed bounds. The coin was biased such that it came up heads with probability 0.05. One of the collaborators was always set to the best action from the other agent's population at the previous generation; the others were chosen by a tournament selection of size 2.

We fixed the budget to 17600 evaluations of joint rewards. When choosing this budget, we felt that too small of a value might prevent differentiations among the algorithms because they would not be allowed to search enough. Similarly, too large of a budget might diminish the differences between methods that waste evaluations and methods that use them effectively. The value we chose seemed to be a good compromise.

The experiments were performed using the ECJ library [4], and they involved 250 runs per method. Performance was computed as the average of the best performing pair of actions (one per each agent) at the last generation. The results indicated that the CCEA with a variable degree of lenience was significantly better in all four problem domains than CCEAs involving a fixed degree of lenience (statistical significance was verified using t-tests assuming unequal variances at 95% confidence).

3. LENIENT MULTIAGENT REINFORCEMENT LEARNING

Drawing inspiration from dynamic programming concepts, reinforcement learning (RL) methods update the estimates of utilities for performing actions in various states of the environment, or for being in those states themselves. These utilities are used for both the exploration of the space, and for the exploitation of the agent's knowledge about the environment. As the memory requirements of traditional RL grow exponentially with the number of agents, multiagent reinforcement learning reduces the memory consumption by decomposing the utilities of joint actions into per-agent utilities of actions. We assume for simplicity that the environment has a single state, and we only focus on computing the utility of choosing different actions; this is similar to the analysis of multiagent RL in [1, 2, 3].

The proposed lenient multiagent reinforcement learning algorithm (LMRL) is based on the following idea: if at early stages of learning an agent receives rewards $r_1, r_2, ..., r_k$ when choosing action a_1 at various times, the agent only updates the utility of a_1 based on the maximum of $r_1, r_2, ..., r_k$, ignoring the others. The reason for this is that those rewards were obtained while the other learning agent selected some actions $b_1, ..., b_k$, most of which we poor choices to begin with and will be abandoned by the other agent in the future due to low utility. But later, after each agent has largely converged to one or a few actions, it becomes more important to achieve accurate utility estimates. The agents therefore will decrease their degree of lenience to one another by ignoring fewer of the lower utilities.

To implement this idea, an agent always updates the utility of the action if the current reward exceeds the utility of that action. Agents associate a temperature with each action, and the level of lenience is inversely proportional to the temperature: the temperature is initially high and agents are thus more likely to ignore lower rewards. The temperatures of actions decrease as those actions are selected, and as a consequence the agents become more likely to incorporate the lower rewards into the utility estimates. As agents have non-zero probabilities of selecting an action at each time step, there is also a small (0.01) probability of ignoring small rewards at all times. Agents choose actions via Boltzman selection.

```
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 value between 0 and 0.001
    $Temp_i = MaxTemp$
Algorithm
Repeat
  // Action Selection
  $MinTemp = 10^{-6} + \min_{i=1}^{N} Temp_i$
  $W_i = \frac{e^{\frac{U_i}{MinTemp}}}{\sum_{j=1}^{N} e^{\frac{U_j}{MinTemp}}}$
  Use probability distribution $W_1$, ..., $W_N$ to select action $i$
  $Temp_i = Temp_i * \delta$
  // Utility Update
  Perform action $i$ and observe reward $r$
  $RandVal =$ random value between 0 and 1
  If $(U_i \leq r)$ or $\left(RandVal < 10^{-2} + \beta^{-\alpha * Temp_i}\right)$ Then
    $U_i = \lambda * U_i + (1 - \lambda) * r$
```

Experiments. We tested LMRL in the Climb and Penalty domains introduced in [1], and in the two stochastic variations of the Climb domain discussed in [2]. We performed a preliminary sensitivity study for the parameters: as a result, we set $MaxTemp = 500$, $\alpha = 2$, $\beta = 2.5$, $\delta = 0.995$, and $\lambda = 0.95$. The agents spent 7500 moves learning. We compared LMRL's performance with that of FMQ [2] and with a straightforward application of RL to concurrent learning (also in [2]). We ran each algorithm 10000 times in each of the four problem domains.

The lenient multiagent RL algorithm consistently converged to the global optimum in the Climb, Penalty, and Partially-Stochastic Climb domains. According to [2], this is equivalent to the performance of FMQ in these problem domains, and it is also significantly better than the performance of traditional Q-learning approaches, as well as to an algorithm previously proposed in [3]. However, the lenience also helped the agents learn the global joint action in the Fully-Stochastic Climb domain in more than 93.5% of the runs. Contrast this to FMQ's poor performance in this domain as mentioned in [2]. We likewise found that FMQ converged to the global optimum solution in only around 40% of runs in this difficult domain, despite an extensive sensitivity study for parameter values.

4. CONCLUSIONS

This paper argues that multiple agents that learn concurrently can benefit from showing lenience to each other, especially during early interactions. We illustrated this concept with perceptions of the joint search space that each agent would perceive during different stages of learning. We then extended two popular multiagent learning algorithms, namely cooperative coevolution and multiagent reinforcement learning, to include lenience in the agents' decision processes, and we showed the superiority of these extensions in several coordination games.

5. REFERENCES

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

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

[3] 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, 2000.

[4] S. Luke. ECJ 10: An Evolutionary Computation research system in Java. Available at http://www.cs.umd.edu/projects/plus/ec/ecj/, 2003.

[5] L. Panait and S. Luke. Selecting informative actions improves cooperative multiagent learning. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi Agent Systems – AAMAS-2006*, 2006.

[6] M. Potter and K. De Jong. A cooperative coevolutionary approach to function optimization. In Y. Davidor and H.-P. Schwefel, editors, *Proceedings of the Third International Conference on Parallel Problem Solving from Nature (PPSN III)*, pages 249–257. Springer-Verlag, 1994.

[7] H. Schwefel. *Evolution and Optimum Seeking*. John Wiley and Sons, New York, 1995.

[8] R. P. Wiegand, W. Liles, and K. De Jong. An empirical analysis of collaboration methods in cooperative coevolutionary algorithms. In E. Cantu-Paz *et al*, editor, *Proceedings of the 2001 Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 1235–1242, 2001.