Department of Computer Science
George Mason University Technical Reports

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

Lenience towards Teammates Helps in Cooperative Multiagent Learning

Liviu Panait liviu@google.com
Keith Sullivan ksulliv@cs.gmu.edu
Sean Luke sean@cs.gmu.edu

Technical Report GMU-CS-TR-2013-2

Abstract

Concurrent learning is a form of cooperative multiagent learning in which each agent has an independent learning process and little or no control over its teammates' actions. In such learning algorithms, an agent's perception of the joint search space depends on the reward received by both agents, which in turn depends on the actions currently chosen by the other agents. The agents will tend to converge towards certain areas of the space because of their learning processes. As a result, an agent's perception of the search space may benefit if computed over multiple rewards at early stages of learning, but additional rewards have little impact towards the end. We thus suggest that agents should be lenient with their teammates: ignore many of the low rewards initially, and fewer rewards as learning progresses. We demonstrate the benefit of lenience in a cooperative coevolution algorithm and in a new reinforcement learning algorithm.

1 Introduction

Recently there has been increased interest in decentralized approaches to solving complex real-world problems. Among such approaches, the area of multiagent systems (MAS) emphasizes the joint behaviors of agents with some degree of autonomy. Unfortunately, hard-coding agents to achieve desired behaviors for the entire team may be difficult: problem complexity increases tremendously with more agents or more complex agent behaviors and interactions. Additionally, agents might need to adapt to new conditions that were unforeseen when their behaviors were designed. Machine learning promises viable solutions to these difficulties.

Our research interest is in cooperative multiagent learning, where multiple agents (a team) learn to solve a joint task or to maximize utility. More specifically, this paper focuses on applications where multiple agents simultaneously learn how to better interact with one another; we refer to this as *concurrent learning* [9].

Early multiagent learning approaches attempted to directly apply machine learning techniques to multiagent systems: each agent behaves rationally and attempts to improve its own behavior. Recent work debates the use of rationality and Nash equilibria [6, 13], and suggest concepts such as reputation and mutual trust to improve learning [1, 8]. We continue this thrust of research with an argument for *lenience*: each agent should be lenient with its teammates at early stages of learning, when each is exploring the space of actions. Lenience may be reduced as learning progresses and agents have begun focusing on a solution. We demonstrate this notion in simple two-agent scenarios, although it is easily extended to multiple agents.

We will begin with an argument for lenience based on illustrations of a single agent's perspective of the joint search space. We then apply lenience to two widely used multiagent learning paradigms, cooperative coevolution and reinforcement learning, and we show how the learning processes benefit as a result. The paper ends with a set of conclusions and directions for future work.

2 Single-Agent Perspectives on the Search Space

Imagine a simple scenario where two agents learn to coordinate. If X is the set of actions that the first agent can choose from, and Y is the set of actions available

This is an extended version of a 2-page poster from AAMAS 2006: Liviu Panait, Keith Sullivan, and Sean Luke. 2006. Lenient Learners in Cooperative Multiagent Systems. In *Proceedings of the 2006 Conference on Autonomous Agents and Multi-Agent Systems (AAMAS).*

This version of the technical report (GMU-CS-TR-2013-2) is a slightly edited and reformatted version of the original, GMU-CS-TR-2005-4. Importantly, the LMRL algorithm pseudocode (Section 4) has been revised for clarity, and a fifth experiment (Rastrigin) has been added which had been accidentally omitted.

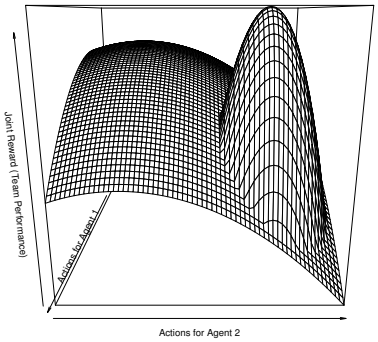

Figure 1: A bimodal search space for the possible rewards received by a two-agent team. Wider peaks may attract many search trajectories, even though such peaks may be globally suboptimal.

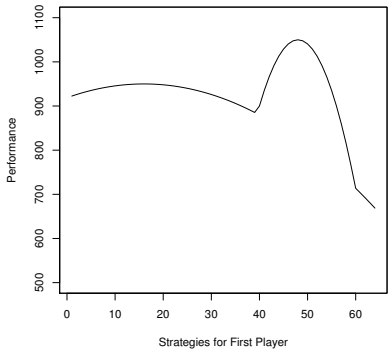

Figure 2: A desirable projection of solution quality for the first population provides enough information about the location of the optimal peadn may be computed at each point x as $\max_{y \in Y} f(x, y)$, where Y denotes the range of the second argument.

to the second agent, the task is for the agents to independently choose one action each (say x and y) such as to maximize the joint reward $f(x, y)$ that they receive. Figure 1 illustrates a search space of joint rewards for a simple coordination game.

The figure shows two peaks of different sizes. The lower peak represents a locally-optimal but globally-suboptimal solution, and the wide coverage of that peak implies that solution quality changes only a little when any agent chooses a somewhat different action. The higher peak represents the global optimum, and its smaller coverage implies that the solution quality may change rapidly if any agent chooses slightly different actions. Both peaks represent Nash equilibria — modifying either the x or the y value (but not both) would lead to a decrease in the function value. This implies that if both agents decide to choose actions corresponding to a Nash equilibrium, neither of them has a rational incentive to choose any other action on its own.

In concurrent multiagent learning, each agent is usually afforded only a partial glimpse of the search space. Specifically, each agent can only detect and respond to the difference in rewards it has received for different actions it has chosen in the past. Given the joint search space in Figure 1, an ideal search space for one of the learning agents is illustrated in Figure 2: for each action x, we plotted $g(x) = \max_{y \in Y} f(x, y)$. If both agents perceived the search space in this manner, they could both learn the actions corresponding to the the globally optimal team performance.

More realistic projections of the joint search space in Figure 1 are illustrated in Figure 3. Here, we compute the projection at point x as the average and the maximum of the rewards obtained by the agents when the first agent chooses action x and the second agent chooses 5 or 20 random actions. The graphs show that taking the average of all rewards results in noisy projections that lose the desired ranking of actions. On the other hand, ignoring all but the maximum reward results in better projections, especially for larger numbers of such rewards: the projection obtained for 20 rewards closely resembles the ideal one in Figure 2. This supports the empirical analysis and conclusions reported in [17].

The assumption that agents choose actions with uniform probability is usually reasonable at the beginning of a multiagent learning process if there is no a priori knowledge about the search space. However, as the agents learn, they tend more and more to choose those actions that resulted in higher rewards in the past. This departure from uniformity alters the projections of the search space, and thus the ranking of actions for each of the agents. Let us assume for simplicity that the second agent chooses its actions according to a normal distribution. How does this affect the projection of the joint search space as perceived by the first agent?

First, suppose that the second agent starts to prefer actions around the wider, suboptimal peak. Different projections of the search space as perceived by the first agent are illustrated in Figure 4. The projections resemble the one in Figure 2 when the standard deviation and the number of rewards are high. However, all information about the optimal solutions is lost in the projections that correspond to small standard deviations: coordinated actions of *both* agents are required to achieve higher payoffs on the optimal peak. On the other hand, suppose that the second agent tends to prefer actions corresponding to the higher, globally optimal peak (Figure 5). The contour of the optimal peak is clearly distinguishable on each of those projections, while the projection of the suboptimal peak becomes noisier at reduced standard deviation (given the higher peak, this is relatively inconsequential to the learning process).

We also observe that differences between pairs of graphs decrease with standard deviation: while in Figure 3(b) there is a clear differentiation between using 5 and 20 actions (this setting is equivalent to a normal distribution with standard deviation = ∞), such differences

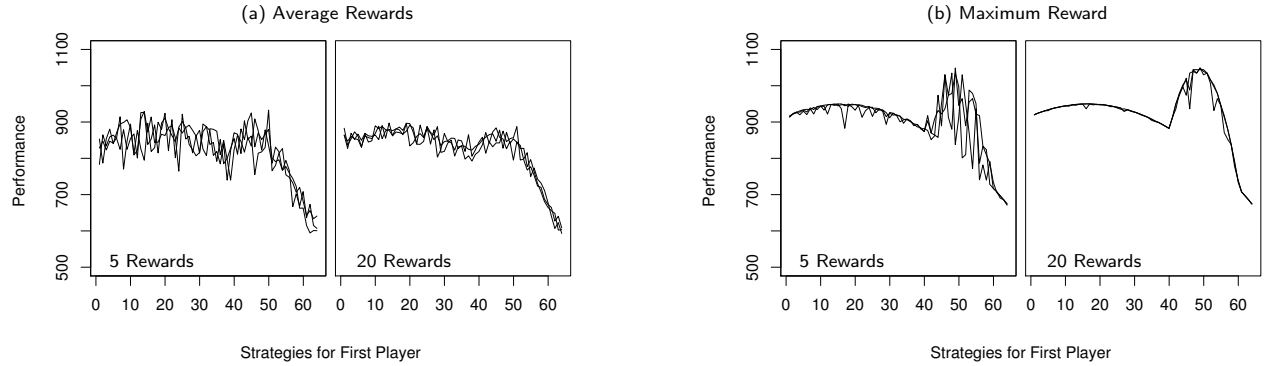

Figure 3: Projected search space for the first agent in the two-peaks domain, assuming the second agent chooses his action randomly with uniform probability. Due to noise, the process is repeated three times — there are three curves on each graph. (a) The projection at point x is computed by averaging 5 and respectively 20 joint rewards $f(x, y_i)$ for different actions y_i. (b) The projection at point x is computed as the maximum of 5 and respectively 20 joint rewards (the lower 4 and respectively 19 rewards are ignored).

are less obvious in Figures 4(a) and 5(b). This suggests that agents may benefit from additional rewards at early stages of learning to increase the accuracy of their projection of the joint search space. It also indicates that the impact of additional rewards may be minimal towards the end of the learning process: there is little increase in the accuracy of the projection when using different numbers of rewards.

What do Figures 2–5 have to do with lenience? First, we observe that using the maximum of multiple rewards provides agents with valuable information about the search space. We thus suggest learning agents should be lenient with their teammates during early stages of concurrent learning processes: ignore many of the low joint rewards, and only take into consideration the higher ones. Second, the quality of projections increases almost insignificantly with more rewards once the learners start to converge. As a consequence, we suggest learning agents should become more critical with respect to joint rewards (ignore fewer of them) during advanced stages of learning.

3 Lenient Cooperative Coevolution

Cooperative coevolutionary systems [11] 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, where the joint problem is decomposed into several (likely intertwined) individual agent subproblems.

A standard approach to applying cooperative coevolutionary algorithms (CCEAs) to multiagent learning assigns each agents its own population of actions[1]. CCEAs do not evaluate actions for an agent in isolation, but they rather evaluate only the performance of a complete team with an action specified for each agent. The fitness (assessed quality) of an action is determined by testing it in combination with tuples containing actions for the other agents (as sampled from their current populations either randomly or based on their performance during the past evaluation phase). When combined with such tuples (which are called *collaborators* in coevolution parlance), an agent's candidate action receives a reward that is in fact the joint reward for the entire team of agents. The fitness of the candidate action is then computed by aggregating multiple such rewards (via taking the average or the maximum, similar to the process used for Figure 3). Aside from this collaborative assessment, each agent follows its own independent evolutionary process in parallel with other agents.

For example, suppose we are learning the optimal joint reward for a team with three agents. If the agents choose actions x, y, and z respectively, the joint reward is specified by $f(x, y, z)$. When applying CCEAs to this problem, one might assign the first population to represent the actions x for the first agent, the second population to represent the actions y for the second agent, and similarly the third population to represent the actions z of the third agent. Each population is evolved separately, except that when evaluating an action in some population (e.g., x), collaborating actions are chosen from the other populations (y and z) in order to obtain a joint reward $f(x, y, z)$. An action's fitness is computed as an aggregation of joint rewards from one or more evaluations with various collaborators: for example, the fitness of x could be computed as $\max\left(f(x, y_1, z_1), f(x, y_2, z_2)\right)$, where the sets $\{y_1, y_2\}$ and $\{z_1, z_2\}$ are selected from the second and from the third populations.

The effects of different settings of cooperative coevolutionary algorithms on search performance have been the focus of research studies such as [2, 17]. The results

[1] For more complex domains, the population could instead contain state-to-action mappings.

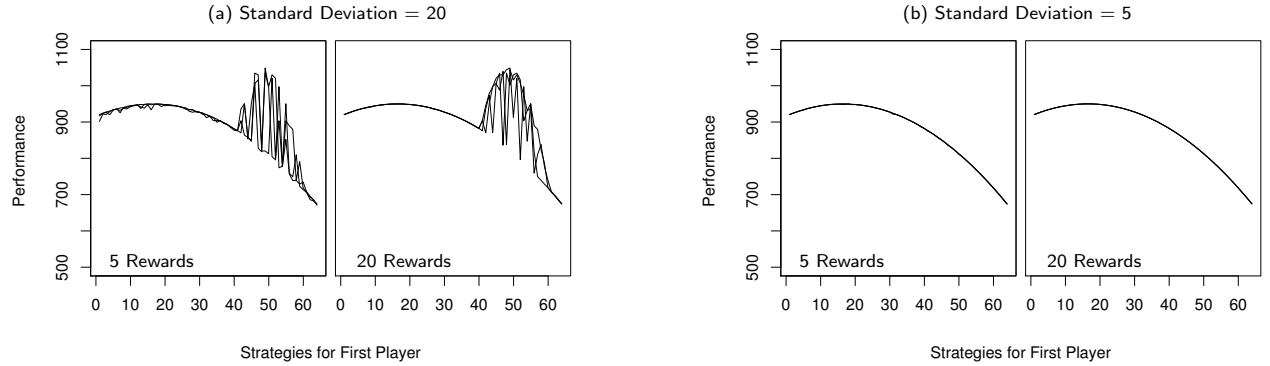

Figure 4: Projected search space for the first agent in the two-peaks domain, assuming the second agent chooses his actions according to a normal distribution centered on the suboptimal peak. The projection at point x is computed as the maximum of 5 and respectively 20 joint rewards $f(x, y_i)$ (the lower 4 and respectively 19 rewards are ignored). The y_i are randomly generated from normal distributions with mean 16 (centered on the suboptimal peak) and standard deviations 20 and 5, respectively. The process is repeated three times.

indicate that computing the fitness of actions as the maximum of multiple joint rewards performs significantly better than computing it as the minimum or as the average. Also, while using multiple collaborators might fare better in domains involving complex non-linear interactions among the components, it additionally increases the computational requirements. Wiegand [16] argues that the sampling of collaborators introduces a certain bias on the search: wider peaks (see Figure 1) are more likely to have collaborators sampled from them, and this may bias the search toward suboptimal solutions; this is termed *relative overgeneralization*. Relative overgeneralization may shift the focus of the search from optimality to robustness — depending on the problem, one criterion may be more important than the other. The impact of multiple collaborators on the learning process is graphically illustrated in [10].

Most cooperative coevolutionary algorithms assume that each action is evaluated as the maximum joint reward obtained with a fixed number of collaborators. Using the maximum to compute the fitness implies that all but one of the joint rewards are completely ignored. In terms of our earlier discussion, this translates into a constant level of lenience for each agent toward its teammates. We argue that a better setting involves a high level of lenience at the beginning of the learning process. In coevolution, this translates into a higher number of collaborators at early generations, followed by fewer collaborators at later stages of search.

There are many approaches to decreasing the number of collaborators over time, including methods that consider the diversity of the populations. Here, we apply a trivial ad-hoc setting: start with 10 collaborators for the first 5 evaluation phases, followed by using only 2 collaborators until exhausting the computational resources. In short notation, we refer to this collaboration setting as *10*5+2*rest*. We leave the analysis of other such schemes for future work.

3.1 Experiments and Results

We constructed several simple coordination games based on benchmark optimization problems with well-known properties as described in [12]. All domains were discretized into 1024×1024 intervals: an agent learns to select one of its 1024 actions so as to maximize the joint reward. Each agent maintained a population of 32 actions. Agents kept unmodified their best action from one learning stage (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.

The coevolutionary algorithm had a budget of 17600 evaluations of joint rewards. With these settings, CCEAs with 5 collaborators ran for around 110 generations. 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.

Our experiments compared the performance of methods using a fixed number of collaborators against our technique employing a variable-sized number of collaborators. We tested 10 different numbers of collaborators (1 to 10). Using a single collaborator allowed for many generations, while more collaborators promised more

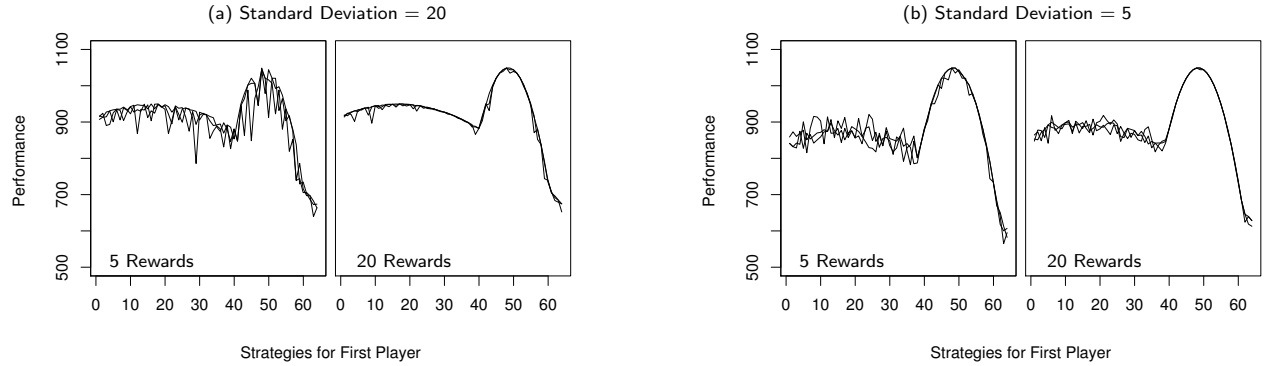

Figure 5: Projected search space for the first agent in the two-peaks domain, assuming the second agent chooses his actions according to a normal distribution centered on the optimal peak. The projection at point x is computed as the maximum of 5 and respectively 20 joint rewards $f(x, y_i)$ (the lower 4 and respectively 19 rewards are ignored). The y_i are randomly generated from normal distributions with mean 48 (centered on the optimal peak) and standard deviations 20 and 5, respectively. The process is repeated three times.

Number of Collaborators	Mean Performance	Std. Dev. Performance
1	960.308064	30.622387
2	989.164076	48.826596
3	1004.111361	49.314857
4	1020.230865	44.603223
5	1031.049270	37.868720
6	1035.819193	32.235383
7	1036.992860	30.671499
8	1042.224061	22.536767
9	1042.303012	22.076877
10	1041.991019	20.991886
10*5+2*rest	**1047.110667**	**16.520011**

Table 1: Performance of different collaboration schemes in the discretized two-peak problem domain. 10*5+2*rest significantly outperforms all other settings.

accurate fitness assessment mechanisms at the expense of fewer generations.

The experiments were performed using the ECJ library [7], 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. Statistical significance was verified using t-tests assuming unequal variances at 95% confidence.

We must point out a few problems with our experiments. First, the same collaborators are used to evaluate an entire population at each generation, as opposed to drawing random collaborators for each action. This reduces the evaluation noise in the population, but it also limits the exploration of the search space. We are not aware of much literature comparing these two settings, but we do not expect the difference to be significant. Second, the collaborators are chosen based on a tournament selection of size 2, as opposed to choosing them randomly. This setting might diminish the benefit of multiple collaborators later on in the search (the extra selection pressure might make the population seem more converged than it actually is). In our defense, we point to the relationship between this selection pressure and the diversity in the population: the results of the selection process might be similar (in terms of expectation) if using random selection from a less diverse population, or using tournament selection with size 2 from more diverse populations. Thus, we argue that other collaboration schemes could work better than the fixed collaboration schemes even if random selection were used instead.

The first experiment compared the results of the settings described above in the Two-Peaks domain illustrated in Figure 1. Specifically, the joint reward for a team with two agents, where the first agent chooses action x and the second agent chooses action y, is computed as

$$f(x,y) = \max \begin{cases} 950 - 500 * \left(\left(\frac{x-16}{64}\right)^2 + \left(\frac{y-16}{64}\right)^2 \right) \\ 1050 - 9600 * \left(\left(\frac{x-48}{64}\right)^2 + \left(\frac{y-48}{64}\right)^2 \right) \end{cases}$$

where x and y range from 0 to 64, discretized into 1024 intervals. This problem domain is illustrated in Figure 1. The results are summarized in Table 1. The results for our decreasingly lenient setting, 10*5+2*rest, are significantly better than the ones for all other settings.

The second experiment tested the methods using a Rosenbrock-like function[2] two-dimensional search space with

$$f(x,y) = 1000 - \left(2 * \left(x^2 - y\right)^2 + (1-x)^2\right)$$

where x and y range between -5.12 and 5.12 discretized into 1024 intervals. The results are presented in Table 2.

[2]This function resembles the two-dimensional Rosenbrock, but with a diminished influence of the non-linear component of the fitness function.

Number of Collaborators	Mean Performance	Std. Dev. Performance
1	999.805368	0.702603
2	999.931824	0.165259
3	999.948679	0.151703
4	999.946774	0.133769
5	999.939639	0.142567
6	999.947729	0.110188
7	999.951085	0.082815
8	999.943306	0.107280
9	999.940341	0.117084
10	999.945124	0.088891
10*5+2*rest	**999.986614**	**0.054527**

Table 2: Performance of different collaboration schemes in the discretized Rosenbrock-like problem domain. 10*5+2*rest significantly outperforms all other settings.

Number of Collaborators	Mean Performance	Std. Dev. Performance
1	999.974879	0.087959
2	999.992303	0.017698
3	999.992725	0.014395
4	999.994459	0.003547
5	999.994457	0.003247
6	999.994630	0.003522
7	999.994800	0.003434
8	**999.995458**	**0.003786**
9	999.994743	0.004014
10	999.994870	0.004457
10*5+2*rest	**999.995700**	**0.003646**

Table 3: Performance of different collaboration schemes in the discretized Griewangk problem domain. 10*5+2*rest significantly outperforms all other settings, except for 8 collaborators.

The results for 10*5+2*rest are again significantly better than the ones for all fixed collaboration schemes.

A third experiment used the Griewangk function

$$f(x,y) = 1000 - \left(1 + \frac{x^2}{4000} + \frac{y^2}{4000} - \cos(x)\cos\left(\frac{y}{\sqrt{2}}\right)\right)$$

with x and y between -5.12 and 5.12, discretized again into 1024 intervals. This modified function has several suboptimal peaks, and an optimum of value 1000 at (0,0). The results of different collaboration methods on this problem domain are presented in Table 3. The results indicate that 10*5+2*rest is significantly better than all other settings except for 8 collaborators.

The fourth experiment used the Booth problem domain. In this domain, the two-argument optimization function equals

$$f(x,y) = 1000 - \left((x + 2*y - 7)^2 + (2*x + y - 5)^2\right)$$

(again transformed to a maximization problem), with x and y between -5.12 and 5.12, discretized into 1024 intervals. This is a challenging problem for coevolutionary search because of non-linearities among the variables; the optimum of the function is at (1,3) and it has a value of 1000. The results of the methods in this domain are presented in Table 4. The 10*5+2*rest method has a significantly better performance than all fixed settings.

The fifth and last experiment used the Rastrigin function (converted for maximization):

$$f(x,y) = 1000 - \left(6 + \left(x^2 - 3\cos(2\pi x) + \left(y^2 - 3\cos(2\pi y)\right)\right)\right)$$

with x and y taking values between -5.12 and 5.12, discretized into 1024 intervals. A peculiarity of this function is the large number of suboptimal peaks surrounding the global optimum. Due to large variances among the results in this problem domain, we performed 1000 runs for each of the settings, but we still were not able able to distinguish among any of the methods. The results are presented in Table 5. In this domain, the t-tests failed to indicate the superiority of any fixed collaboration scheme over the 10*5+2*rest setting.

4 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 [14]. These utilities are used for both the exploration of the space, as well as 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 joint utility tables into simpler utility tables, one per agent. This in essence projects the joint utility tables into per-agent tables which discount other agents. 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 [3, 5, 4].

Previous research in multiagent RL has focused on straightforward applications of traditional techniques to multiagent problems. This led to algorithms where agents update their utility estimates based on each and every reward they observe, similar to applications of RL to single-agent environments. From a higher-level perspective, these algorithms approach the multiagent learning problem using agents with no lenience for one

Number of Collaborators	Mean Performance	Std. Dev. Performance
1	999.944621	0.258064
2	999.945253	0.278818
3	999.896547	0.624545
4	999.927854	0.182885
5	999.940833	0.168899
6	999.911407	0.215897
7	999.912132	0.194326
8	999.893961	0.227951
9	999.903006	0.208813
10	999.877023	0.315515
10*5+2*rest	**999.996837**	**0.022781**

Table 4: Performance of different collaboration schemes in the discretized Booth problem domain. 10*5+2*rest significantly outperforms all other settings.

Number of Collaborators	Mean Performance	Std. Dev. Performance
1	999.295637	0.686536
2	999.282862	0.696519
3	999.306431	0.707764
4	999.255339	0.752893
5	999.253378	0.715312
6	999.286966	0.730213
7	999.270756	0.696554
8	999.251628	0.716294
9	999.266060	0.741226
10	999.215434	0.717056
10*5+2*rest	**999.258291**	**0.747410**

Table 5: Performance of different collaboration schemes in the discretized Rastrigin problem domain. Although 5*10+2*rest has lower mean performance than other settings, it is not statistically significantly better or worse because of the large variance in performance.

another (this resembles a cooperative coevolutionary setting using a single collaborator). We continue with a brief discussion of previous multiagent reinforcement learning literature, followed by lenient multiagent reinforcement learning algorithm that selectively updates the utilities of actions based only on *some* of the rewards. The results of preliminary experiments suggest that agents learning via RL techniques significantly benefit when showing lenience toward one another.

Claus and Boutilier [3] show that straightforward applications of RL to concurrent learning are not guaranteed to find the optimal solution for these games, even in the case when agents are able to observe the other agents' actions. The authors suggest that the search could be improved by using more optimistic exploration actions. This direction is further explored in [5], who update the utilities of actions based in part on the maximum reward previously received when performing that action. Kapetanakis and Kudenko [4] 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, which 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 its performance is poor when there is a lot of noise. Finally, Verbeeck et al [15] 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.

In contrast, our proposed lenient algorithm improves the performance of each learning trial without requiring any *a priori* coordination. It 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 ignores most of these rewards and only updates the utility of a_1 based on the maximum of $r_1, r_2, ..., r_k$. 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 might be ignored in the future due to lower utilities. As both agents become more selective at choosing their actions, it is expected that they will each tend to primarily select a single action (the best one) after some time. At this point, we would prefer that each agent updates the utility of that action based on every reward he observes. This will lower the optimistic estimation of the utility for that action until it equals the mean reward obtained by the agents for that joint reward. If this mean reward becomes lower than the estimated utility of other actions, the agent will tend to prefer other actions instead. This is desirable for games with stochastic rewards, such as the ones in Table 7.

We implement the algorithm as follows. We always update the utility of the action if the current reward exceeds the utility of the action. Otherwise, we use a probabilistic approach: if the agent has not explored that action sufficiently, it should show lenience to its teammates and not update its policy; but if the agent has explored that action many times in the past, it should tend to be more critical and use the reward to lower the utility of that action. The algorithm associates a temperature with each action, as opposed to a single temperature for the entire learning process. If the temperature associated with an action is high, the agent is more lenient and ignores low rewards it receives for choosing that action. 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 with 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 experiments because agents have non-zero probabilities of selecting an action at each time step.

Aside from these enhancements, the algorithm follows a traditional RL approach: the action selection uses the Boltzman distribution, and the utility is updated based in part on the reward currently received for an action. On top of the memory required to store the utility table, the proposed algorithm needs storage to encode a temperature for every action. This is about the same as the memory requirement of the algorithm proposed in [5], and half that of FMQ [4]. The pseudocode for the algorithm is as follows:

Lenient Multiagent Reinforcement Learning

Parameters

$MaxTemp \leftarrow$ maximum temperature
$\alpha \leftarrow$ temperature multiplication coefficient
$\beta \leftarrow$ exponent coefficient
$\delta \leftarrow$ temperature decay coefficient
$\lambda \leftarrow$ learning rate
$N \leftarrow$ number of actions

Initialization

$U_1, ..., U_N \leftarrow$ per-action utilities (all 0)
$Temp_1, ..., Temp_N \leftarrow$ per-action temperatures (all 0)
for each action i **do**
    $U_i =$ random value between 0 and 0.001
    $Temp_i = MaxTemp$

Each Iteration

▷ Select an Action

$P_1, ..., P_N \leftarrow$ per-action probability distribution (all 0)
$MinTemp \leftarrow 10^{-6} + \min_{i=1}^{N} Temp_i$
for each action i **do**
    $P_i \leftarrow \frac{e^{\frac{U_i}{MinTemp}}}{\sum_{j=1}^{N} e^{\frac{U_j}{MinTemp}}}$
Select action $i^\star \in \{1, ..., N\}$ at random using probability distribution P_1,...,P_N

▷ Perform Action

Perform action $i^\star$ and observe reward r
$Temp_{i^\star} \leftarrow Temp_{i^\star} \times \delta$

▷ Update Utilities

$RandVal \leftarrow$ random value between 0 and 1 inclusive
if $(U_{i^\star} \leq r)$ or $\left(RandVal < 10^{-2} + \beta^{-\alpha \times Temp_{i^\star}}\right)$ **then**
    $U_{i^\star} \leftarrow \lambda * U_{i^\star} + (1 - \lambda) * r$

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

		Agent 2		
Agent 1		a	b	c
	a	10	0	-10
	b	0	2	0
	c	-10	0	10

Table 6: Joint reward matrixes for the Climb (left) and Penalty (right) domains.

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

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

Table 7: Joint reward matrixes for the Partially-Stochastic (left) and the Fully-Stochastic (right) versions of the Climb domain. The reward is stochastic: if both agents choose action b, the reward 14/0 implies that both agents receive either a reward of 14 or a reward of 0 with probability 50% each.

For additional clarity, Figure 6 illustrates the utilities associated with each action during a concurrent learning process in the Fully-Stochastic Climb domain. Both agents start with low utilities for all actions, and they slowly start to believe that action b has a utility value of 14, higher than all other actions. As both agents choose action b, they decrease the temperature associated with this action, and soon they start to incorporate lower rewards (remember that the joint action (b, b) has an average utility of 7). As a consequence, the utility of action b decreases, and the agents start to explore other actions as well. This generates extra miscoordination penalties which lower the estimates for the utilities of all actions. Given the higher temperature for action a, its utility is affected by high rewards, and both agents start to prefer it over the other actions. This leads to a decrease in miscoordination penalties, and both agents end up with precise utilities associated with that action.

4.1 Experiments and Results

We experiment with learning in repeated coordination games, in which agents repeatedly and independently choose one action each, and they are rewarded equally based on the joint action they have selected. The goal is for the agents to learn a joint action that has a maximum reward (or maximum average reward for the stochastic games). We employ four problem coordination games

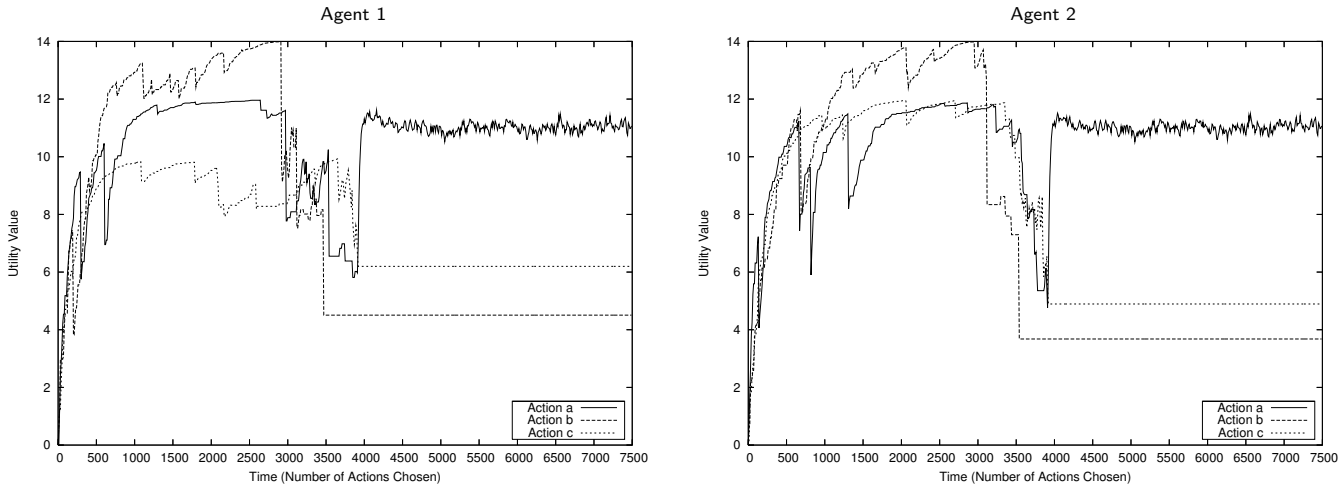

Figure 6: Utility of each agent's three actions.

(Tables 6–7): the Climb and Penalty domains introduced in [3], and two stochastic variations of the Climb domain as discussed in [4]. Climb is difficult because of the large penalties associated with miscoordination of actions corresponding to the two higher optima, while Penalty is challenging because the agents need to select among two joint actions with equal reward. Agents may also select suboptimal actions that avoid miscoordination penalties in each domain: action *c* in Climb, and action *b* in the Penalty domain. The stochastic variations of the Climb domain are challenging due to the additional noise in the rewards for joint actions. We experimented with the lenient multiagent RL algorithm in all four problem domains. 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 learned over 7500 moves. Table 8 reports the number of runs that converged to each of the nine joint actions (based on each agent's action with maximal utility at the end of the run). These numbers are averaged over 10 trials of 1000 runs each.

The lenient multiagent RL algorithm consistently converged to the global optimum in the Climb, Penalty, and Partially-Stochastic Climb domains. According to [4], this is equivalent to the performance of FMQ in these problem domains, and it is also significantly better than the performance of both traditional Q-learning approaches, as well as to the algorithm previously proposed in [5]). 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. This contrasts FMQ's poor performance in this domain as mentioned in [4]; we also 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 and a higher number of joint actions (7500 instead of 2000 in previous work)

Climb

	a	b	c
a	992.4	0	0
b	0	7.6	0
c	0	0	0

Penalty

	a	b	c
a	494.6	0	0
b	0	0	0
c	0	0	505.4

Partially-Stochastic Climb

	a	b	c
a	999.0	0	0
b	0	0.0	0.6
c	0	0	0.4

Fully-Stochastic Climb

	a	b	c
a	935.2	0	0.001
b	0	20.2	20.4
c	0	0	24.1

Table 8: Average number of runs (out of 1000) that converged to each of the joint actions for the four coordination games.

5 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 graphically based on projections 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.

Future work is required in multiple directions. What formal guarantees can be proved for these algorithms? How can agents automatically detect the appropriate level of lenience? Are these algorithms readily applicable to teams with tens, hundreds, or thousands of agents, or what further extensions are required for this purpose? We hope future research will help us find answers to these challenging questions.

References

[1] B. Banerjee, R. Mukherjee, and S. Sen. Learning mutual trust. In *Working Notes of AGENTS-00 Workshop on Deception, Fraud and Trust in Agent Societies*, pages 9–14, 2000.

[2] L. Bull. Evolutionary computing in multi-agent environments: Partners. In T. Back, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 370–377. Morgan Kaufmann, 1997.

[3] 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.

[4] 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.

[5] 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.

[6] M. I. Lichbach. *The cooperator's dilemma*. University of Michigan Press, 1996.

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

[8] R. Mukherjee and S. Sen. Towards a pareto-optimal solution in general-sum games. In *Agents-2001 Workshop on Learning Agents*, 2001.

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

[10] L. Panait, R. P. Wiegand, and S. Luke. A visual demonstration of convergence properties of cooperative coevolution. In *Parallel Problem Solving from Nature – PPSN-2004*. Springer, 2004.

[11] 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.

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

[13] Y. Shoham, R. Powers, and T. Grenager. On the agenda(s) of research on multi-agent learning. In *Proceedings of Artificial Multiagent Learning. Papers from the 2004 AAAI Fall Symposium. Technical Report FS-04-02*, 2004.

[14] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[15] K. Verbeeck, A. Nowe, and K. Tuyls. Coordinated exploration in stochastic common interest games. In *Proceedings of the Third Symposium on Adaptive Agents and Multi-Agent Systems (AAMAS-3)*, 2003.

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

[17] 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.