Journal of Machine Learning Research 17 (2016) 1-42 Submitted 8/15; Revised 4/16; Published 5/16

Lenient Learning in Independent-Learner Stochastic Cooperative Games

Ermo Wei EWEI@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

Editor: Kevin Murphy

Abstract

We introduce the *Lenient Multiagent Reinforcement Learning 2* (LMRL2) algorithm for independent-learner stochastic cooperative games. LMRL2 is designed to overcome a pathology called *relative overgeneralization*, and to do so while still performing well in games with stochastic transitions, stochastic rewards, and miscoordination. We discuss the existing literature, then compare LMRL2 against other algorithms drawn from the literature which can be used for games of this kind: traditional ("Distributed") Q-learning, Hysteretic Q-learning, WoLF-PHC, SOoN, and (for repeated games only) FMQ. The results show that LMRL2 is very effective in both of our measures (complete and correct policies), and is found in the top rank more often than any other technique. LMRL2 is also easy to tune: though it has many available parameters, almost all of them stay at default settings. Generally the algorithm is optimally tuned with a single parameter, if any. We then examine and discuss a number of side-issues and options for LMRL2.

Keywords: multiagent learning, reinforcement learning, game theory, lenient learning, independent learner

1. Introduction

In a cooperative game, as we use the term here, some $N \geq 2$ players simultaneously choose an action from among those actions each is permitted to make, and then each receives the same (potentially stochastic) reward arising from their joint action. It is this equally-shared reward that makes the game cooperative: agents succeed or fail together. Such cooperative games are very common in multiagent systems: for example, a group of robots might work together to map out a room; a swarm of game agents might collaborate to defeat a hard-coded Bad Guy; or a team of wireless beacons might work together to form an optimal ad-hoc routing network.

We are interested in the situation where the agents in question do not know the reward function over their joint actions and must *learn* to collaboratively explore this space and ultimately adapt to the highest-reward joint action. Furthermore, the agents are *independent learners* in the sense that they do not know what actions the other agents are performing (although they may assume that other agents *exist*). In a primary alternative, where the

agents are told the others' action choices along with the rewards received, the agents are known as *joint-action learners*.

From a game theoretic perspective, there are two common types of games in which such agents may learn to optimize their actions. First, a game may be *repeated*, meaning that the agents play the same game over and over again, potentially choosing new actions each time. Second, there is the so-called *stochastic* game, where not only does a joint action affect the reward received, but it also changes the game played next time. In a stochastic game there is a set of possible sub-games (or *states*) for the agents to play, each with potentially different kinds of actions available to the agents, and each with their own reward function. Furthermore, each joint action in each state is associated with a transition function which maps joint actions to distributions over states: from the appropriate distribution the next state will be selected. There is a distinguished initial state for the game, and there may optionally be a special end state: the game simply terminates when it reaches this state. The agents know in which state they are playing, but not its reward function nor its transition functions. A repeated game is just a stochastic game with only one state.

There is significant literature in applying multiagent versions of reinforcement learning, policy search, or similar methods to repeated and stochastic games. Much of the literature has focused on general-sum or zero-sum learners, and the remainder has focused nearly entirely on the pathology of *miscoordination* in cooperative scenarios, as discussed later. But there is another critical pathology which arises specially in cooperative games, known as *relative overgeneralization*. Overcoming relative overgeneralization may require a more explorative approach than simple epsilon-greedy action selection: indeed as shown by Wiegand (2004), relative overgeneralization can cause players to not just hill-climb into local optima but get actively sucked into them despite action selection with a high degree of randomness. We discuss these pathologies in more detail in Section 3.

Perhaps because relative overgeneralization as a phenomenon necessitates at least three actions available per player, and ideally more, it has not shown up much in the small problems common to the multiagent reinforcement learning (MARL) literature, though it has been studied at some length in the related field of cooperative co-evolutionary algorithms (Panait, 2006; Panait et al., 2006a, 2008, 2004), which involve very large numbers of actions. We are interested in solving both relative overgeneralization and miscoordination. To this end we present and discuss a *lenient* learning algorithm meant for independent learning in stochastic cooperative games (and also repeated cooperative games) with any number $N \geq 2$ of agents. We call this algorithm *Lenient Multiagent Reinforcement Learning 2*, or LMRL2 for short. This is a heavy revision of the original LMRL algorithm, which was originally meant only for repeated games (Panait et al., 2006b).

As we will show, LMRL2 performs well across a spectrum of stochastic and repeated games when compared to other algorithms from the literature, and is able to deal robustly with overgeneralization, miscoordination, and high degrees of stochasticity in the reward and transition functions with relatively little parameter tuning. In this paper we will discuss issues in MARL which lenient learning is meant to address, then survey previous work and alternative approaches. We will then detail LMRL2, compare it to other methods, and discuss various side issues which arise.

2. Previous Work

Reinforcement Learning is a popular approach to solving multiagent learning problems (Busoniu et al., 2008), because many such problems may be readily cast into normal-form games, which themselves map straightforwardly into Markov Decision Processes (MDPs) for which reinforcement learning is well suited. Use of reinforcement learning in the multiagent context is, not suprisingly, known as *Multiagent Reinforcement Learning*, or MARL.

Many MARL methods may be organized according to three different characteristics. First, there is the *reward structure* involved: MARL algorithms may be designed for zero-sum (or constant-sum), cooperative, and general-sum games. Second, there is the *type of information* being provided to the algorithms. Here we distinguish between *joint action* and *independent learner* algorithms, following Claus and Boutilier (1998). Third, there is the *game type*: some MARL algorithms are meant only for repeated games, while others are meant for more general stochastic games. LMRL2 itself is an independent learner, cooperative, stochastic game algorithm.

Joint Action Learners In a *joint action learner* scenario, after playing a game, each agent receives a reward, is told which game is to be played next (the next *state*), and is also told what actions were chosen by the other agents. There is significant joint action learner literature.

Only a small portion of the literature focuses on pure zero-sum games. One of the first zero-sum joint-action-learning algorithms was Minimax-Q (Littman, 1994), which assumes that the alternative agent is attempting to minimize one's own reward. Thus, rather than select the action which with the maximum reward achievable, Minimax-Q uses a minimax approach: it selects the highest reward achievable assuming that the alternative agent will select its action so as to minimize that reward. This solution may be determined by solving a linear program.

There are many general-sum joint-action learners. However, learning in general-sum games is very complicated due to the wide variety in reward structure. Even the goal of learning in this kind of game can be hard to define in some cases (Shoham et al., 2004). To simplify the issue, much of the literature simply aims to converge to an equilibrium of some sort during self-play (that is, playing against other agents who use the same algorithm as yourself). The most well-known algorithm of this kind is called Nash Q-Learning (Hu and Wellman, 2003), which has been proven to converge to the Q-values in some Nash Equilibrium under very restricted conditions, such as if each state has a global joint optimum. Other learning algorithms which have been designed to converge to equilibria include Wellman and Hu (1998); Greenwald et al. (2003); Jafari et al. (2001). Rather than convergence, some work has instead proposed playing a best response against some restricted classes of opponents (Weinberg and Rosenschein, 2004; Tesauro, 2004). There has also been work in combining the two, where the learner converges to Nash Equilibrium in self-play, and plays a best response when not in self-play (Conitzer and Sandholm, 2007).

Like the zero-sum game literature, the pure cooperative game literature is limited. Littman (2001b) introduced Team Q-learning, but as it is a straightforward extension of Q-Learning to cooperative games, it fails to handle miscoordination. To address this Wang and Sandholm proposed OAL (2002), which is guaranteed to converge to the optimal Nash Equilibrium in cooperative games: but due to necessary constraints on the reward func-

tion, converging to the optimal policy is not particularly interesting for joint action learners in cooperative games. Other work has imported techniques from the reinforcement learning community to enhance online performance. Notably Chalkiadakis and Boutilier applied Bayesian Reinforcement Learning to the multiagent scenario (2003). Here, agents have some prior knowledge on distributions of the game model and also the possible strategies that can be used by other learners. Each iteration of the game, an agent chooses a best action with respect to the distributions it is maintaining, and the distributions are then updated based on the individual actions of the other agents and the results from the joint action. Typical of Bayesian Reinforcement Learning methods, a bad prior can make the algorithm converge to a suboptimal policy in some cases, but Bayesian methods in general may help agents accumulate more reward while learning. Bayesian methods have also been used with policy search in a MARL context (Wilson et al., 2010), where the distributions are instead over policy parameters and possible roles of other agents. Unusually, agents here make decisions sequentially rather than simultaneously, which is rare in the MARL literature.

Another way to incorporate prior knowledge into learning is to construct the policy as a hierarchy. Hierarchical reinforcement learning has been shown to accelerate learning in the single agent case by allowing state abstraction and by reusing learned policy (Dietterich, 2000). This has been extended to the multiagent setting (Makar et al., 2001; Ghavamzadeh et al., 2006) by manually specifying the cooperation task at a higher level in the task hierarchy. This allows non-cooperative subtasks to be learned separately, and efficiently as single-agent learning problems. The authors have further used this idea in a setting where knowing the actions of other agents incurs a communication cost (Ghavamzadeh and Mahadevan, 2004).

Independent Learners An *independent learner* scenario differs from a joint action scenario in that, after playing a game, each agent is *not* told what action was chosen by each of the other agents. He is only told the reward he received and the game (state) to be played next time. This is a much more difficult learning problem. Additionally, because they are designed for a more general scenario, independent learners may participate in joint action games, but not the other way around.

Perhaps the best-known example of a general-sum independent learner is WoLF-PHC (Bowling and Veloso, 2001b, 2002). This algorithm is designed to meet two criteria for learning in general sum games: first, a learner should learn to play optimally against stationary opponents, and second, a learner should converge to a Nash Equilibrium when in self-play. WoLF-PHC is an independent-learning policy search approach which uses one of two different learning rates depending on whether the player is in some sense "winning" or "losing". This "WoLF" (Win or Learn Fast) principle has since been applied to other algorithms (Bowling and Veloso, 2001a; Banerjee and Peng, 2003; Bowling, 2005).

Other approaches, based on gradient descent, also try to converge to Nash Equilibria in self play (Abdallah and Lesser, 2008; Zhang and Lesser, 2010). Additionally, Kaisers and Tuyls linked evolutionary game theory to reinforcement learning to study the dynamic of Q-learning in multiagent problems. Their proposed algorithm, Frequency Adjusted Q-learning (FAQ), resembles regular Q-learning except that it changes the update method to compensate for the fact that actions are updated at different frequencies (Kaisers and Tuyls, 2010).

Cooperative learning presents a variety of special problems for independent learners as opposed to joint-action learners. Fulda and Ventura (2007) identified three problematic factors — "poor individual behavior", "action shadowing" (related to relative overgeneralization, as discussed later), and "equilibrium selection" (miscoordination) — and suggested that optimal performance might be achieved by solving them. However in their survey, Matignon et al. (2012) identified at least two more problems, and compared several independent learners in several games along with different exploration strategies to overcome these problems.

Some work has been done in cooperative games. Lauer and Riedmiller introduced Distributed Q-learning (2000), which addresses two major problems mentioned in (Fulda and Ventura, 2007). It tackles relative overgeneralization problem by only updating using the maximum Q-value — an approach we will develop further with LMRL2 — and it tackles the miscoordination problem by only changing the policy when the Q-value is updated. However this learner is vulnerable to games with stochastic rewards or transitions. Building on Distributed Q-learning, Hysteretic Q-learning (Matignon et al., 2007) uses two different learning rates to address stochasticity. One study (Bloembergen et al., 2010) adds our concept of leniency into FAQ, using a slightly different approach from what we do here. Leniency seems to help FAQ solve initial miscoordination difficulties in cooperative games.

Instead of changing the update procedure, researchers have also investigated the possibility of using different exploration strategies. Frequency Maximum Q-Value heuristic (FMQ), meant only for repeated games, used a form version of Boltzmann action selection where the Q-value was substituted by an expectation (Kapetanakis and Kudenko, 2002). FMQ was modified by Matignon et al. to handle stochasticity in repeated games (2008) and stochastic games (2009). The latter version of this algorithm was called *Swing between Optimistic or Neutral*, or SOoN.

Learners Based on the Nature of the Other Agents Most of the literature discussed so far focuses on learning in a certain kind of game. However, instead of making assumptions about the nature of a game, some methods are based on the nature of the other agents. In so-called Friend-or-Foe Q-learning (Littman, 2001a), the learner is essentially a mixture of two different Q-learners: one which updates the Q-table like a regular Q-learner if the other agent is thought to be a friend, and one which updates the Q-table using Minimax-Q if the agent is thought to be an opponent. Asymmetric Q-learning (Könönen, 2004), distinguishes between "leaders" and "followers" in the game, where a "leader" knows what action a "follower" will choose can guide the action selection of the "follower" (see also the leader-follower reward shaping in Babes et al. 2008). CMLeS provides different performance guarantees depending on different set of agents in the game (Chakraborty and Stone, 2013a,b). Specifically, if the agents are doing self-play, the algorithm will achieve a Nash Equilibrium joint policy; if the alternative agents are opponents with limited memory, the algorithm can guarantee a near-optimal response in polynomial time steps; and in other cases, the algorithm will play similarly to Minimax-Q.

Another area of research has studied adaptation to arbitrary agents of unknown type. Sullivan et al. (2006) studied FMQ and Lenient Learners in non-self play scenarios, and showed that while these learners could easily converge to an optimal Nash Equilbrium in self-play for various games, it was difficult for them to do so when paired with learners from entirely different algorithms. Stone et al. (2010) consider ad-hoc teams of multiple agents

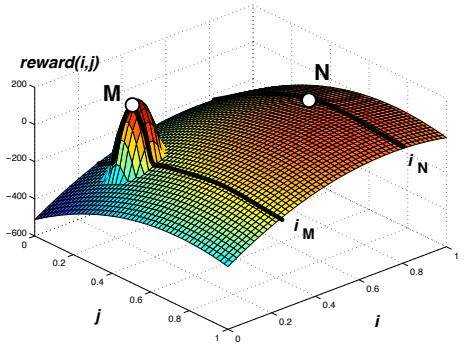


Figure 1: The relative overgeneralization pathology in multiagent learning. The axes i and j are the various actions that agents A_i and A_j may perform, and the axis $reward(i, j)$ is the joint reward received by the agents from a given joint action $\langle i, j \rangle$. Higher rewards are better. Joint action M has a higher reward than joint action N. However, the average (or sum) of all possible rewards for action i_M, of agent A_i is lower than the average of all possible rewards for action i_N. To illustrate this, bold lines show the set of outcomes from pairing i_M or i_N instead with other possible actions in j.

with different capabilities whose goals and utilities are aligned but which have had no prior coordination. They give a method to evaluate the performance of an agent in this scenario, and provide theoretical approaches to building a perfect ad hoc team. Additional follow-on methods have since been suggested (Stone and Kraus, 2010; Agmon et al., 2014; Genter et al., 2015; Barrett and Stone, 2015).

3. Lenient Learning

The notion of lenient learning, and the pathology it is meant to tackle, did not start in multiagent reinforcement learning research, but rather in an unexpected but surprisingly related topic: cooperative coevolutionary algorithms (CCEAs) (Potter and De Jong, 1994). CCEAs are multiagent stochastic optimization procedures and exhibit nearly identical pathologies to those found in repeated games.

Relative Overgeneralization Lenient learning was originally designed to combat a particular pathology in repeated games and CCEAs called *relative overgeneralization* (Wiegand, 2004). Relative overgeneralization occurs when a suboptimal Nash Equilibrium in the joint space of actions is preferred over an optimal Nash Equilibrium because each agents' action in the suboptimal equilibrium is a better choice when matched with *arbitrary* actions from the collaborating agents. Consider Figure 1, showing the reward over the joint action space for a two-agent game for agents A_i and A_j. Here, the joint candidate solution labeled M should clearly be preferred over another one, N. However if one assessed the quality of individual actions based on the sum or average reward received when paired with all possible other actions from the collaborating agent, then agent A_i's portion of the joint action N

(called i_N) would be preferred over its portion of the joint action M (called i_M). That is, quality$(i_M) = \sum_j \text{reward}(i_M, j) <$ quality$(i_N) = \sum_j \text{reward}(i_N, j)$. We will call such an approach an *average-based* algorithm.

This situation can easily arise in cooperative, repeated games for independent learners. Here is an example of relative overgeneralization in a repeated game:

		Agent 2		
		a	b	c
Agent 1	a	10	0	0
	b	0	5	6
	c	0	6	7

In this game, though $\langle a, a \rangle$ is clearly the best joint move for the two agents, if the agents sampled their actions randomly and based decisions on the average reward, action a would have the worst score, action c would have the best score, and b would be in the middle. Thus the agents would tend to converge to $\langle c, c \rangle$.

The obvious alternative this is to base the quality of a solution not on its *average* reward, but rather on its *best* reward when paired with various other actions. That is, quality$(i_M) = \max_j \text{reward}(i_M, j)$. In this case, a would be the best action, c the next highest, and b the worst. Thus the agents would tend to converge to the proper equilibrium, $\langle a, a \rangle$. We call this a *maximum-based* approach.

Relative overgeneralization is a specific and problematic subcase of the slightly more general notion of *action shadowing*, occasionally used by the MARL community, but has only been relatively recently studied (Panait et al., 2006b; Panait, 2006; Fulda and Ventura, 2007; Panait et al., 2008). One possible reason for this might be that common MARL test problems are relatively low in number of actions, and relative overgeneralization can only arise if each agent has at least three actions available, ideally many more. In contrast, in CCEAs the number of "actions", so to speak, is very high and often infinite, and so the pathology is a common occurrence there.

Stochastic Rewards and Lenient Learners Some reinforcement learning methods, so-called *optimistic methods* (Matignon et al., 2012), are generally maximum-based learners, or at least prefer superior results, for example, updating superior results with a higher learning rate. In a repeated game with deterministic rewards, these techniques are likely to perform very well. However, games with stochastic rewards will mislead such learners, since the highest rewards they see will often be due to noise. This problem isn't likely to occur with average-based learners.

Our approach instead begins with a maximum-reward learner and gradually shifts to an average-reward learner. The idea here is that early on none of the agents have a good understanding of their best joint actions, and so each agent must initially be *lenient* to the foolish and arbitrary actions being made by its collaborators. Later on, each agent focuses more on average reward, which helps escape the trap laid by stochastic rewards. We call this approach a *lenient multiagent reinforcement learning* algorithm.

Miscoordination Another common pathology which appears in repeated games is *miscoordination*, where two or more equivalent equilibria are offset in such a way that agents, each selecting what appears to be an optimal action, wind up with poor joint actions. For

example, in the game below, both actions a and b are reasonable for each agent. But if one agent chooses a while the other agent chooses b, they receive a low reward.

		Agent 2	
		a	b
Agent 1	a	10	0
	b	0	10

Stochastic Games and Deception Stochastic games add some new wrinkles to the above pathologies, because, at least for temporal difference learners, Q-values come partly from accumulated reward and partly from the backed-up rewards from follow-on states. This means that both miscoordination and relative overgeneralization, among other challenges, can arrive via backed-up rewards as well as immediate rewards.

The probabilistic transitions common in stochastic games present an additional hurdle for lenient learners, since such learners may consider only the highest backed-up rewards, even if they are very unlikely to occur. As was the case for probabilistic rewards, probabilistic transitions are largely overcome by the learner's gradual shift from being maximum-based to being average-based.

This hurdle leads to *deception*, another pathology which can arise in stochastic games. A deceptive game is one in which certain states have a high local reward but lead ultimately to states with poor future rewards: this is also known as the *delayed reward* problem. This creates a garden-path scenario where greedy agents are deceptively led away from the truly high-performing states.

4. The LMRL2 Algorithm

In (Panait et al., 2006b, 2013) we demonstrated a lenient learning algorithm, LMRL, for repeated games with independent learners, comparing it favorably to the FMQ algorithm (Kapetanakis and Kudenko, 2002) for variations of the Climb and Penalty games (Claus and Boutilier, 1998). We begin here with a slight modification of the algorithm, LMRL2, which is the degenerate case (for repeated games) of the full LMRL2 algorithm for stochastic games. We will then extend it to the stochastic game scenario.

LRML2 is a modified version of Q-learning which maintains per-action *temperatures* which are slowly decreased throughout the learning process. An action's temperature affects two things. First, it affects the degree of randomness of action selection: with high temperatures, action selection is largely random, and with low temperatures, action selection is greedily based on the actions with the highest Q-values. To do this, LRML2 applies a temperature-based Boltzmann Selection common in other algorithms in the literature. However, when the average temperature drops below a certain *minimum temperature*, and LRML2's action selection will suddenly become purely greedy. This minimum temperature was originally added to avoid floating point overflows common in Boltzmann Selection; but we have found that it also is beneficial for nearly all games.

Second, temperature affects the lenience of the algorithm: high temperatures cause LRML2 to be lenient, and so only mix rewards into Q-values if they are better than or equal to the current Q-value. With a low temperature LRML2 mixes all rewards into the

Q-values no matter what. Unlike action selection, lenience is not affected by the minimum temperature bound.

Repeated Games LRML2 for repeated games relies on the following parameters. Except for θ and ω, all of them will be fixed to the defaults shown, and will not be modified:

$\alpha \leftarrow 0.1$	learning rate
$\gamma \leftarrow 0.9$	discount for infinite horizon
$\delta \leftarrow 0.995$	temperature decay coefficient
$\mathit{MaxTemp} \leftarrow 50.0$	maximum temperature
$\mathit{MinTemp} \leftarrow 2.0$	minimum temperature
$\omega > 0$	action selection moderation factor (by default 1.0)
$\theta > 0$	lenience moderation factor (by default 1.0)

The α and γ parameters are standard parameters found in Q-learning, and their values here are typical settings from the literature. The parameters *MaxTemp,* δ, and *MinTemp* are also generally constants. The parameter ω determines the degree to which temperature affects the randomness of the Boltzmann action selection. In all cases but two (discussed later) we set this to 1.0. Finally and crucially, the θ parameter determines the degree to which temperature affects the dropoff in lenience. This parameter is the primary, and usually only, parameter which must be tuned, as different problems require different amounts of lenience.

LMRL2 maintains two tables: Q, a table of Q-values per action a, and T, a table of temperatures per-action. Initially $\forall a$:

$$Q(a) \leftarrow \text{initialize}(a) \qquad \triangleleft \textit{ See discussion below}$$
$$T(a) \leftarrow \mathit{MaxTemp}$$

In lenient learning, the choice of initial Q values is important to the operation of the algorithm. Consider if $\forall a : Q(a) = 0$ initially. What if a game consisted only of negative rewards? The lenient learner, at least during its high-temperature period, would refuse to merge *any* of them into the Q-values because they are too small. We have two strategies for initializing Q:

- **Initialize to Infinity** $\forall a : Q(a) = \infty$. This signals to LMRL2 to later reinitialize each Q value to the first reward received. Furthermore, as long as one or more action has an infinite Q-value, LRML2 will only select among such actions. This forces the algorithm to try every action at least once initially in order to initialize them. The disadvantage of this initialization approach is that if the game has stochastic rewards, and first reward received is high, LMRL2 will be locked to this high reward early on.
- **Initialize to Minimum** $\forall a : Q(a) =$ the minimum possible reward over any action the game. The disadvantage of this approach is that it requires that LMRL2 know the minimum reward beforehand.

We will by default initialize to the minimum possible reward in the game.

LMRL2 then iterates for some n times through the following four steps. First, it computes a mean temperature $\overline{T}$. Second, using this mean temperature it selects an action to perform. Third, it performs the action and receives a reward resulting from the joint actions of all agents (all agents perform this step simultaneously and synchronously). Fourth, it updates the Q and T tables. This iteration is:

1. Compute the mean temperature as: $\overline{T} \leftarrow \text{mean}_a\, T(a)$
2. Select a as follows. If $\overline{T} < \mathit{MinTemp}$, or if $\max_a Q(a) = \infty$, select $\text{argmax}_a\, Q(a)$, breaking ties randomly. Otherwise use Boltzmann Selection:
   (a) Compute the action selection weights as: $\forall a : W_a \leftarrow e^{\frac{Q(a)}{\omega \overline{T}}}$
   (b) Normalize to the action selection probabilities as: $\forall a : P_a \leftarrow \frac{W_a}{\sum_i W_i}$
   (c) Use the probability distribution P to select action a.
3. The agent does action a and receives reward r.
4. Update $Q(a)$ and $T(a)$ only for the performed action a as:

$$\mathit{Rand} \leftarrow \text{random real value between 0 and 1}$$

$$Q(a) \leftarrow \begin{cases} r & \text{if } Q(a) = \infty \qquad \text{(initialization was to infinity)} \\ (1-\alpha)Q(a) + \alpha r & \text{else if } Q(a) \leq r \text{ or } (\mathit{Rand} < 1 - e^{\frac{-1}{\theta T(a)}}) \\ Q(a) & \text{else} \end{cases}$$

$$T(a) \leftarrow \delta T(a)$$

5. Go to 1.

Note that each action has its own separate temperature which is decreased only when that action is selected. This allows LMRL2 to keep temperatures high for actions which have not been visited much and still require lenience and exploration, while permitting other actions to cool down.

Stochastic Games In stochastic games, the agents are at any particular time in some state s; and after performing their joint action, receive a reward r and transition to some new state s'. Accordingly, the extension of LMRL2 to stochastic games is largely the same as the repeated game version, except that the $Q(a)$ and $T(a)$ tables are modified to include the current state s: that is, they are now defined as $Q(s,a)$ and $T(s,a)$ respectively.

The iteration is largely the same, except in how $Q(s,a)$ and $T(s,a)$ are updated. First, $Q(s,a)$ is updated in standard Q-learning style to incorporate both the reward and expectation of future utility, as $r + \gamma \max_{a'} Q(s',a')$. However, if not all actions have been explored in s', then $\max_{a'} Q(s',a')$ will still be infinite, in which case it is ignored and $Q(s,a)$ just incorporates r.

Second and more interestingly, not only do we decrease $T(s,a)$, but we also fold into it some portion τ of the mean temperatures found for s'. τ is a new constant, fixed like α to 0.1. The idea is as follows: in many games (particularly episodic ones), early states are often explored much more than later states, and thus cool down faster. We want to keep

these early states hot long enough that propagation of future rewards from the later states will inform the Q-values of the early states before they are set in stone as the temperature falls. To do this, we take some of the temperature s' of the later state and back it up into s.

Because stochastic games can terminate, we wrap the entire process in an outer loop to repeatedly play the game:

Parameters:

$\alpha \leftarrow 0.1$	learning rate
$\gamma \leftarrow 0.9$	discount for infinite horizon
$\tau \leftarrow 0.1$	temperature diffusion coefficient
$\delta \leftarrow 0.995$	temperature decay coefficient
$\mathit{MaxTemp} \leftarrow 50.0$	maximum temperature
$\mathit{MinTemp} \leftarrow 2.0$	minimum temperature
$\omega > 0$	action selection moderation factor (by default 1.0)
$\theta > 0$	lenience moderation factor (by default 1.0)

Initially $\forall s, a$:

$Q(s,a) \leftarrow \text{initialize}(s,a)$ $\quad\quad\quad\quad$ $\triangleleft$ *See Previous Discussion*
$T(s,a) \leftarrow \mathit{MaxTemp}$

Repeat:

1. Current state $s \leftarrow$ initial state.

2. Repeat until the current state s is the end state (if any):

   (a) Compute the mean temperature for current state s as: $\overline{T}(s) \leftarrow \text{mean}_a\, T(s,a)$

   (b) Select a as follows. If $\overline{T}(s) < \mathit{MinTemp}$, or if $\max_a Q(s,a) = \infty$, select $\text{argmax}_a\, Q(s,a)$, breaking ties randomly. Otherwise use Boltzmann Selection:

      i. Compute the action selection weights in current state s as: $\forall a : W_a \leftarrow e^{\frac{Q(s,a)}{\omega \overline{T}(s)}}$

      ii. Normalize to the action selection probabilities in current state s as: $\forall a : P_a \leftarrow \frac{W_a}{\sum_i W_i}$

      iii. Use the probability distribution P to select action a.

   (c) The agent, in current state s, does action a, receives reward r, and transitions to new state s'.

(d) Update $Q(s,a)$ and $T(s,a)$ only for the performed action a as:

$$Rand \leftarrow \text{random real value between 0 and 1}$$

$$R \leftarrow \begin{cases} r & \text{if } \max_{a'} Q(s',a') = \infty \\ r + \gamma \max_{a'} Q(s',a') & \text{else} \end{cases}$$

$$Q(s,a) \leftarrow \begin{cases} R & \text{if } Q(s,a) = \infty \text{ (initialization was to infinity)} \\ (1-\alpha)Q(s,a) + \alpha R & \text{else if } Q(s,a) \leq R \text{ or } (Rand < 1 - e^{\frac{-1}{\theta T(s,a)}}) \\ Q(s,a) & \text{else} \end{cases}$$

$$T(s,a) \leftarrow \delta \times \begin{cases} (1-\tau)T(s,a) + \tau\overline{T}(s') & \text{if } s' \text{ is not the end state (if any)} \\ T(s,a) & \text{else} \end{cases}$$

(e) $s \leftarrow s'$

If we defined a repeated game as consisting of an initial state s which always transitions to the end state as s', this algorithm degenerates to the repeated version of LMRL2 discussed earlier.

5. Comparison with Other Methods

We begin with a comparison of LMRL2 against six other independent-learner algorithms in self-play in several cooperative test problems. The algorithms are standard (classic) Q-Learning, Distributed Q-Learning, Hysteretic Q-Learning, WoLF-PHC, SOoN (Swing between Optimistic or Neutral), and FMQ. The different techniques use a variety of parameters and have several action selection methods as options. We note that WoLF-PHC is a general-sum algorithm rather than strictly a cooperative one, and that FMQ can only be applied to repeated games. In a MARL context, standard Q-Learning is sometimes known as *Decentralized Q-Learning*. We briefly discuss all except for standard Q-Learning below.

Distributed Q-Learning We believe that Distributed Q-Learning (Lauer and Riedmiller, 2000) was the earliest independent-learner algorithm specifically designed for cooperative multiagent learning scenarios. In Distributed Q-Learning, each learner has a Q-table and a policy table. Unlike the regular Q-Learning, where a value in the Q-table is updated by combining it with some portion of the reward and follow-on utility, in Distributed Q-Learning the Q-value is completely replaced by the new reward and follow-on utility (that is, the learning rate is effectively 1.0), but only if doing so would increase it. This makes Distributed Q-Learning a maximum-based learner; and it is intended to address relative overgeneralization. Because it is highly optimistic, the algorithm has problems with stochasticity: indeed Lauer and Riedmiller (2000) acknowledged that this was still an open question.

To deal with miscoordination, Distributed Q-Learning has another trick up its sleeve: its policy is only updated when the Q-value of the policy's chosen action is no longer the highest such value. At this point a new action for the policy is chosen at random among those with the highest Q-values. The idea here is to cause agents to lock onto the earliest-discovered good action (and Nash Equilibrium) even when other equivalent Equilibria exist. Through this implicit agreement among agents, miscoordination can be avoided.

Hysteretic Learning Matignon et al. (2007) proposed Hysteretic Learning to address Distributed Q-Learning's vulnerability to stochasticity. Hysteretic Learning is not a fully maximum-based learner: rather, if updating the Q-value would reduce it, it is reduced with a smaller learning rate than when it would be if it were increased. Hysteretic Learning does not attempt to solve miscoordination explicitly, but experiments suggest that the algorithm is very robust to miscoordination issues, in part due to the randomization in its exploration strategy (Matignon et al., 2012).

FMQ Rather than change the update strategy, as is done in the previous two methods, the Frequency Maximum Q-Value (FMQ) heuristic instead tries changing the action-selection (exploration) strategy (Kapetanakis and Kudenko, 2002). FMQ is only meant for repeated games. FMQ uses a modified version of Boltzmann exploration, similar to the one used in LMRL2 as discussed later: the value produced via Boltzmann is sometimes called the *expected reward* or *estimated value*. However, this alone is not sufficiently informative to deal with relative overgeneralization. Thus, FMQ adds an additional term to its Q-value when using it during action selection. This additional term is the product of the *highest reward* seen so far when selecting that action, the *frequency* of getting that reward, and a *weighting factor* c which controls how much this term affects action selection: that is, how much action selection is based on the "average" reward versus typical "high" rewards.

FMQ are designed to deal with stochasticity in relative overgeneralization problems, but its learned policy can still be wrong if the variance of the reward function is high.

SOoN SOoN may be thought as a heavily modified FMQ. First, the frequency term is broken into two terms. The *myopic frequency* is the the same frequency term in original FMQ algorithm, and the *farsighted frequency* is used to deal with deception in games. Both frequency values are updated using temporal-difference style methods rather than a simple average. Additionally, the estimated value is computed as a linear interpolation between an Optimistic Q-value (the Q-value in Distributed Q-Learning) and an Average Q-value (the classical Q-Learning value). The algorithm introduces two new parameters, α_f and α_g, to replace the weighting factor c in FMQ to control the impact of the frequency terms.

WoLF-PHC Given its notoriety, we chose WoLF-PHC as our exemplar general-sum algorithm to compare against. WoLF-PHC is a policy hill-climbing algorithm and so is somewhat different from the various Q-Learning methods (Bowling and Veloso, 2001b). WoLF-PHC compares the expected value of current policy with the expected value of the average policy. If the former is lower, then the agent is "losing", else it is "winning". WoLF-PHC then uses one of two different learning rates depending on whether it is winning or losing (the "winning" learning rate is higher). As it is meant for general-sum games, WoLF-PHC can be straightforwardly applied to cooperative games: but obviously since it is more general-purpose, a comparison against it is somewhat unfair.

5.1 Test Problems

We tested against twelve games, either from the literature or of our own devising. This collection was meant to test a diverse array of situations, including: stochastic and repeated games, recurrent and episodic games, deterministic and stochastic rewards, deterministic and stochastic state transition functions, deceptive problems, miscoordination, and relative

overgeneralization. The games are defined in Appendix **??**, but their various features are summarized here.

We tested with four repeated-game test problems from the literature. The widely used *Climb* and *Penalty* games (Claus and Boutilier, 1998) are designed to test some degree of relative overgeneralization and miscoordination. We also included versions of the Climb game with partially stochastic and fully stochastic rewards, here designated *Climb-PS* and *Climb-FS* respectively (Kapetanakis and Kudenko, 2002). In these versions, the reward for a joint action was potentially one of two different values with certain probabilities; though the expected reward was always the same as in the original Climb game. Stochasticity greatly complicates the problem: we noticed that no existing algorithm can completely solve Climb-FS.

We also tested against several stochastic games. The *Boutilier* game (Boutilier, 1999) was a repeated game with deterministic transitions which distributed a miscoordination situation among several stages. The *Common Interest* game (Vrancx et al., 2008) is also recurrent, but with stochastic transitions and some miscoordination. This game is notable in that its rewards are unusually close to zero compared to other games; this is the reason that LMRL2 required a modification of its ω parameter to perform well in this game.

The remaining games are of our own design. The *Gradient 1* game is a deterministic episodic game designed to be highly deceptive and to cause miscoordination. *Gradient 2* is similar, except that it incorporates fully stochastic rewards. The *Heaven and Hell* game also causes miscoordination and deception, but is recurrent. This game has a high-reward state ("heaven"), a low-reward state ("hell"), and two medium ("purgatory") states of different levels of reward. Choice of a high-reward action will deceptively transition to a lower-reward future state, and the converse is also true. The *Relative Overgeneralization 1* game (or RO 1) causes miscoordination and relative overgeneralization not in local rewards but in propagated utilities from later states. *Relative Overgeneralization 2* (or RO 2) causes relative overgeneralization in both local rewards *and* in propagated utilities. Finally *Relative Overgeneralization 3* (or RO 3) causes miscoordination and relative overgeneralization from propagated utilities, but does so entirely through a stochastic transition function. We summarize the properties of each game in Table 1.

5.2 Parameters

All the techniques had a variety of tunable parameters, and many of them could apply different action selection methods. We considered two such methods, *epsilon-greedy selection* and *Boltzmann selection*. Epsilon-greedy selection is characterized by an initial random action selection probability ϵ which is optionally reduced each timestep by multiplying it by a cut-down factor ν. The version of Boltzmann selection was the one we employed in the LMRL2 algorithm, and so was characterized by the *MaxTemp, MinTemp,* δ, and ω parameters. Boltzmann selection could be feasibly used by LMRL2, Q-Learning, and Hysteretic Q-Learning, though usually only LMRL2 benefitted from it (other learners generally worked better with Epsilon-greedy). FMQ used its own algorithmic-specific version of Boltzmann selection. Table 2 shows the default (baseline) parameter settings we set for each of the techniques.

Test Problem	Repeated Game	Episodic Game	Deception	Stochastic Transitions	Stochastic Rewards	Relative Over-generalization	Miscoordination
Boutilier			×				×
Common Interest			×	×			×
Gradient 1		×	×				×
Gradient 2		×	×		×		×
Heaven and Hell			×				×
RO1		×	×			×	
RO2		×	×			×	
RO3		×	×	×		×	×
Climb	×	×				×	
Climb-PS	×	×			×	×	
Climb-FS	×	×			×	×	
Penalty	×	×					×

Table 1: Properties of each test problem

Test Problem	Default Parameters
LMRL2	$\alpha : 0.1,\ \gamma : 0.9,\ \tau : 0.1,\ \delta : 0.995,$ MaxTemp : 50, MinTemp : 2, $\omega : 1,\ \theta : 1$, Boltzmann
Q	$\alpha : 0.1,\ \gamma : 0.9,\ \epsilon : 0.1,\ \nu : 1.0$, Epsilon-greedy
Distributed Q	$\gamma : 0.9,\ \epsilon : 0.1,\ \nu : 1.0$, Epsilon-greedy
Hysteretic Q	$\alpha : 0.1,\ \gamma : 0.9,\ \epsilon : 0.1,\ \nu : 1.0,\ \beta : 0.01$, Epsilon-greedy
WoLF-PHC	$\alpha : 0.1,\ \gamma : 0.9,\ \epsilon : 0.1,\ \nu : 1.0,\ \delta_w : 0.03,\ \delta_l : 0.06$, Epsilon-greedy
SOoN	$\alpha : 0.1,\ \gamma : 0.9,\ \epsilon : 0.1,\ \alpha_f : 0.05,\ \alpha_g : 0.3$, Epsilon-greedy
FMQ	$\alpha : 0.1,\ \gamma : 0.9,\ c : 10,$ MaxTemp : 500, MaxMove : 2000, FMQ-specific Boltzmann

Table 2: Default Parameter Settings for each technique.

To make the comparison as fair as possible, we then optimized the above parameter settings and action selection method choices on a per-problem, per-technique basis, through a combination of manual tuning and automated hill-climbing. Table 3 shows the resulting optimized parameter changes.

5.3 Comparison

We compared all the aforementioned techniques using all appropriate test problems (FMQ was included only for repeated games). We considered two possible measures of successful convergence on a problem. First, a technique may converge to the *correct policy* for a given game, meaning that it is optimal if it follows this policy. Second, a technique may converge to the more general *complete policy*, meaning that it determines the correct joint action for *every state*, even ones which would never be visited if it followed a correct policy. For some games (all repeated games, and games with fully stochastic transitions) correct and complete policies are the same.

One might imagine that the superior technique would be the one which converged to the complete policy the most often. However what if in doing so it also converged to a great many entirely wrong policies? Consider the following exaggerated scenario. Technique A converged to 2 complete policies, 98 correct policies, and no incorrect policies; while technique B converged to 3 complete policies but 97 incorrect policies. Would we then

Test Problem	LMRL2	Q-Learning	Distributed Q	Hysteretic Q	WoLF-PHC	SOoN	FMQ
Boutilier	—	α:0.05 γ:0.8	—	—	ν:0.997	—	
Common Interest	ω:0.1	α:0.03	ϵ:0.2 ν:0.9	β:0.02	—	α:0.05 α_f:0.3 ϵ:0.05	
Gradient 1	θ:10^7	α:0.15 γ:1 ϵ:0.35	ϵ:0.4	ϵ:0.4	α:0.15 γ:1 δ_l:0.05 ϵ:0.5	α_g:0.05 α_f:0.1 ϵ:0.3	
Gradient 2	—	α:0.95 ϵ:0.3	ϵ:0.01	β:0.15 MinTemp:0.015 MaxTemp:500 δ:0.999 Boltzmann	α:0.15 γ:1 ϵ:0.4	α_g:0.1 α_f:0.3 ϵ:0.05	
Heaven and Hell	θ:10^7	α:0.15 γ:0.7	—	α:0.15 β:0.03 γ:0.7	α:0.2 γ:0.8 ϵ:0.2	α_g:0.05 α_f:0.03	
RO 1	θ:10^3	α:0.15 γ:1 ϵ:0.13	ϵ:0.15	β:0.0001 ϵ:0.2	δ_w:0.06 δ_l:0.12 ϵ:0.01	α_g:0.02	
RO 2	θ:10^3	ϵ:0.25	ϵ:0.4	α:0.3 β:0.0001 γ:1 ϵ:0.5	δ_w:0.06 δ_l:0.12 ϵ:0.01	α_g:0.01	
RO 3	ω:0.3	α:0.13 ϵ:0.13	ϵ:0	β:0.0001 ϵ:0.05	δ_w:0.06 δ_l:0.12	α:0.05 α_g:0.03 α_f:0.03	
Climb	θ:10^7	ϵ:0.0	ϵ:0.1	β:0.0001	δ_l:0.21 ϵ:0.01	—	—
Climb-PS	θ:10^3	ϵ:0.01	ϵ:0	β:0.01 MinTemp:2 MaxTemp:40 δ:0.99 Boltzmann	δ_l:0.21 ϵ:0.01	α_g:0.2 α_f:0.05 ϵ:0.05	—
Climb-FS	θ:10	α:0.05 γ:0.8	ϵ:0.01	β:0.001 ν:0.99	δ_l:0.18 ϵ:0.01	α_g:0.2 α_f:0.3 α:0.03	c:200
Penalty	—	α:0.05 ϵ:0.35	ϵ:0.1	β:0.01 ϵ:0.12	ϵ:0.28	—	—

Table 3: Tuned Parameter Settings. Shown are deviations from the default settings (in Table 2) for each method when tuned to perform best on a given problem.

Test Problem	Iterations	Complete	Correct
Boutilier	30000	$\overline{\text{L Q D H W S}}$	$\overline{\text{L Q D H W S}}$
Common Interest	40000	H $\overline{\text{L W Q}}$ S D	
Gradient 1	40000	D H S $\overline{\text{Q L}}$ W	$\overline{\text{L D S}}$ H Q W
Gradient 2	40000	W S Q L H D	L S $\overline{\text{Q W H}}$ D
Heaven and Hell	40000	$\overline{\text{D L}}$ W H S Q	$\overline{\text{L D H W S Q}}$
RO 1	30000	$\overline{\text{D H}}$ S L Q W	$\overline{\text{L D H S}}$ Q W
RO 2	30000	L $\overline{\text{S H}}$ D $\overline{\text{Q W}}$	$\overline{\text{L H S}}$ D $\overline{\text{Q W}}$
RO 3	30000	L $\overline{\text{Q S W H}}$ D	L $\overline{\text{Q S W H}}$ D
Climb	15000	$\overline{\text{D H S L}}$ F Q W	
Climb-PS	15000	S L F H D Q W	
Climb-FS	15000	L S $\overline{\text{F D}}$ H Q W	
Penalty	15000	$\overline{\text{D H F S L Q}}$ W	

L=LMRL2 Q=Q-Learning D=Distributed Q H=Hysteretic Q W=WoLF-PHC S=SOoN F=FMQ

Table 4: Summary of Statistically Significant differences among methods for complete and correct solution counts on different test problems. Techniques are ordered left to right in decreasing performance. $\overline{\text{Overbars}}$ group together settings with no statistically significant differences among them. In the Common Interest, Penalty, and various Climb games, correct solutions are by definition complete. Also shown is the number of iterations run on each test problem.

really consider Technique B to be superior? For this reason, we must consider *both* of these approaches as comparison measures.

Our interest is not in convergence time so much as quality of the converged result. As such we ran each technique for until both the value function stabilized and the policy remained unchanged. We used the largest iteration number required among all learners for our formal experiments. This often benefitted LMRL2: being based on a temperature schedule, LMRL2 converges more slowly than other (greedier) techniques. However, it also benefitted other methods which occasionally needed large numbers of iterations and small settings of ϵ to produce good results. Not surprisingly, the number of iterations needed was closely correlated to the maximum path length of the game.

Because the results are from Bernoulli trials, we needed a large number of runs to do proper statistical comparison. We chose 10,000 iterations up front, and compared differences in complete or correct solution counts. Statistical significance was verified using the Marasquilo procedure for χ^2. We used $p = 0.05$ for the entire experiment, Bonferroni-corrected to $p = 0.0026315789$ per problem.

5.4 Results

Table 4 summarizes the rank order among the methods and statistically significant differences. Table 5 shows the actual results.

- LMRL2 fell in the top statistical significance tier for "complete" eight times, two times more than the next-best method (Distributed Q-Learning).

Test Problem	LMRL2	Q-Learning	Distributed Q	Hysteretic Q	WoLF-PHC	SOoN	FMQ
Boutilier	**10000**/**10000**	**10000**/**10000**	**10000**/**10000**	**10000**/**10000**	**9998**/ **9998**	**9996**/ **9996**	
Common Interest	**9971**	9942	2080	**9990**	**9968**	9896	
Gradient 1	*8407/**10000**	8609/ 8695	**9999**/ **9999**	9925/ 9926	2329/ 2335	8966/ **9986**	
Gradient 2	548/ **9997**	2430/ 5995	0/ 1266	157/ 5609	**5147**/ 5897	3329/ 6770	
Heaven and Hell	**9991**/**10000**	8289/ 9942	**10000**/**10000**	9848/**10000**	9954/**10000**	9530/ **9975**	
RO 1	†9368/**10000**	3350/ 3350	**10000**/**10000**	**10000**/**10000**	1943/ 2280	9877/**10000**	
RO 2	**9999**/**10000**	2/ 514	9258/ 9258	9897/**10000**	2/ 512	9924/**10000**	
RO 3	**7332**/ **7332**	5337/ 5337	2272/ 2272	2737/ 2737	2769/ 2769	5171/ 5171	
Climb	**9999**	1661	**10000**	**10000**	387	**10000**	9956
Climb-PS	9930	1820	2821	7454	391	**9995**	9857
Climb-FS	**9016**	1763	3874	2558	676	8723	3894
Penalty	**9999**	**9997**	**10000**	**10000**	9951	**10000**	**10000**

* With $\alpha = 0.05$ this value rose to 9207. This changed the ranking of "complete" results such that LMRL2 moved from worse than Q-Learning to better than Q-Learning (both statistically significant differences).

† With $\alpha = 0.05$ this value rose to 9750, but did not change any ranking.

Table 5: Complete/Correct solution counts among methods on various test problems. Boldface values indicate methods which were not statistically significantly different with the highest-performing method for a given problem. Note that in the Common Interest, Penalty, and various Climb games, correct solutions are by definition complete.

- LMRL2 fell in the top statistical significance tier for "correct" eleven times, three times more than the next-best method (Swing between Optimistic or Neutral).
- LMRL2 was uniquely statistically significantly best in RO 1 and Climb-FS.
- SOoN was uniquely statistically significantly best in Climb-PS.
- The Gradient 2 game proved very challenging to all methods. Yet WoLF-PHC, which often failed in other games, was statistically significantly best at Gradient 2 in "complete". LMRL2 underperformed WoLF-PHC on Gradient 2 in "complete", but outperformed it (and all others) in "correct".
- Changing α improved LMRL2 twice, and once statistically significantly, but not by much.
- All methods easily solved the Boutilier game.

In short, LMRL2 performed very well. It often did not reach 100%, but it was usually easily close enough to fall in the top statistical significance tier for "complete", and almost always in the top tier for "correct". No other method was as consistent across games. Furthermore, LMRL2 did so with very few changes to its default parameters: in all cases at most one parameter needed to be tuned. We are particularly interested in the fact that while LMRL2 performed very well in RO 2 and RO 3 — games which exhibit the relative overgeneralization pathology for which LMRL2 was designed — it underperformed in RO 1!

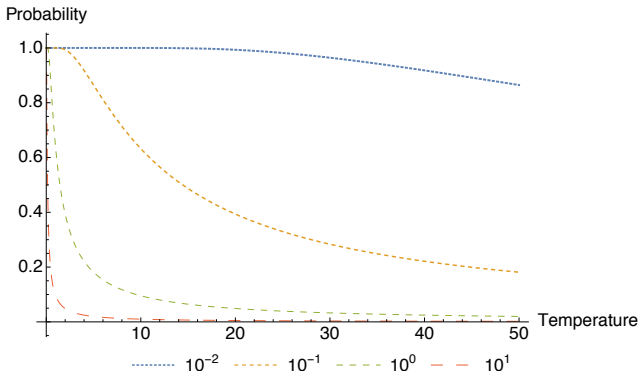


Figure 2: Probability that LMRL2 will accept poor results, by current temperature, for various θ values.

6. Analysis and Discussion

We have experimented with a number of variations of LMRL2 and have examined different issues which arise when using it. In the following section, we discuss five of these issues. First, we discuss the critical issue of initialization of Q-values and its impact on LMRL2's success. Second, we discuss how LMRL2 deals with issues in miscoordination. Third, we examine the disadvantages of Boltzmann selection and compare alternatives. Fourth, we consider the impact of *latching*, whereby temperature is propagated to earlier states only when it is hotter than those states. Finally, we examine what happens when LMRL2 is paired with other techniques in non-self-play, and so consider if, so to speak, two heads are better than one.

6.1 Lenience and Q-value Initialization

We first consider how θ affects lenience. LMRL2 (initially), Hysteretic Q-learning, and Distributed Q-learning are all *optimistic learners*, meaning that they unilaterally update the Q-values to more closely reflect new results if the new results are superior: but they are more conservative when the new results are inferior. That is, they incorporate some portion of maximum-based learning. More specifically: LMRL2 updates to inferior results only with a certain probability (which increases over time). Hysteretic Q-learning updates to inferior results at a fixed, slower rate. And Distributed Q-learning never updates to inferior results.

In LMRL2, optimism is controlled by θ. Figure 2 shows the probability curves of doing a Q-value update corresponding to different θ values and different temperatures. We can see that, with some given initial temperature (perhaps 50), a higher θ value has *two* effects: a higher initial probability of doing Q-value updates and a different probability curve shape. With very high θ, the curve approaches the maximum-learner rule employed by Distributed Q-learning. On the other hand, with a very low θ value, the curve approaches the averaging-learner rule employed by standard Q-learning. We note that a similar relationship can also be made with Hysteretic Q-learning. In Hysteretic Q-learning, improved results are updated

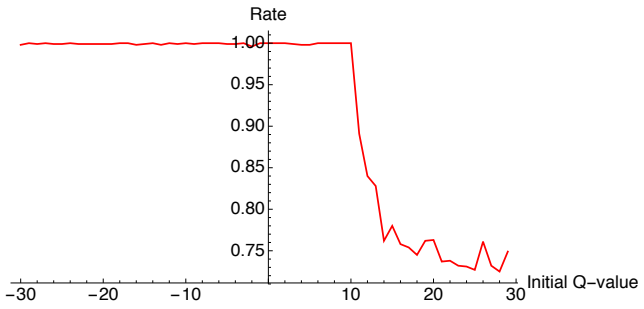


Figure 3: Rate of complete solutions for the LMRL2 in the Climb game, by various initial Q-values. Each point reflects the average of 1000 independent runs.

using α as usual, but worse results are updated using a smaller learning rate $\beta \leq \alpha$. If $\beta = 0$, Hysteretic Q-learning more or less degenerates to regular Q-learning.

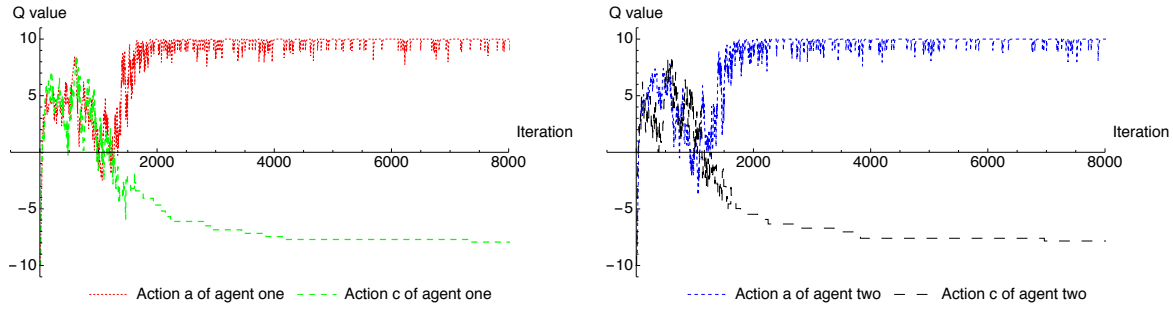
As a lenient learner, LMRL2's optimism is also tempered by the current *temperature*. As can be seen in Figure 2, the initial temperature has a significant effect on the progression of lenience in the system: and indeed if the temperature is infinity, LMRL2 becomes fairly similar to Distributed Q-learning.

In Section 4 we spent significant time discussing Q-value initialization options: this is because optimistic learners, and the like, are very sensitive to initialization. If the Q-values are initialized too high, then these learners have no easy way to adjust them to correct settings: indeed Distributed Q-learning may be completely unable to learn at all. Because in traditional average-based Q-learning the initial estimate of Q-value is less of a concern for proper convergence (Jaakkola et al., 1994), many algorithms simply ignore the discussion of initialization, setting the initial value to 0.

LMRL2 is initially highly optimistic, and so poor initialization can severely hinder it. If the Q-value is greater than the actual feedback, initially the lenient learner will largely refuse to update the Q-value, and so the Q-value won't change, and LMRL2 will thus just pick randomly from among actions during action selection. The temperature of all of the actions then will drop roughly the same rate. Only when the temperature drops enough that LMRL2 shifts to an average-based mode will it start to properly update the Q-values at all. Figure 3 shows how strongly the choice of initial Q values can effect performance: here LMRL2 quickly drops off in performance when the initial value rises to above 10.

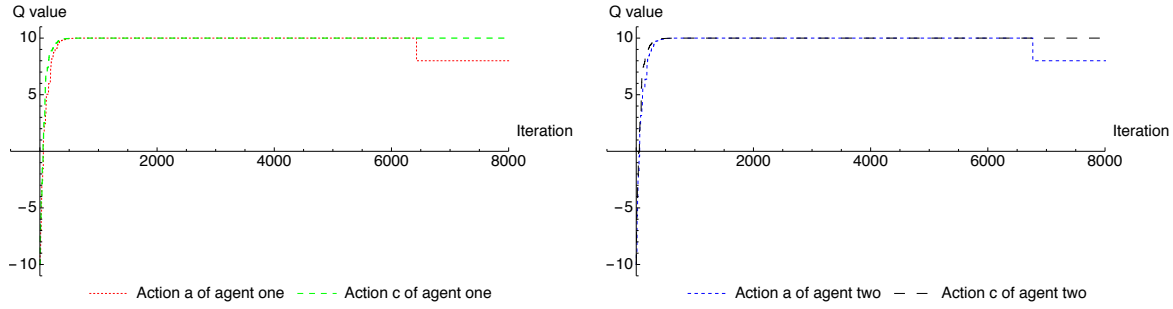
6.2 Miscoordination

To solve the miscoordination problem, an algorithm must break the symmetry among the various optimal equilibria. One way to do this is to always ϵ-greedily select the action with best Q-value, (crucially) breaking ties by selecting the action used last time. This is the approach used in Distributed Q-learning: but this method will only work in self-play. Another way to break symmetry is via different convergence times, which we will talk about in Section 6.5. LMRL2 at present simply relies on the fact that miscoordination is unstable, and that with enough random action, it will likely eventually work its way out.



(a) Agent One (b) Agent Two

Figure 4: Q-values of two agents in the Penalty game with $\theta = 0.1$



(a) Agent One (b) Agent Two

Figure 5: Q-values of two agents in the Penalty game with $\theta = 10^7$

Miscoordination is primarily a problem for a lenient learner early on, when miscoordinated actions both appear to have the same Q-values due to the use of maximum-based learning. However at some point one action will randomly have a slightly higher Q-value than the other, and as the temperature drops, action selection will increasingly choose that action. We have found that this tends to create a sudden collapse to one equilibrium or another. What often happens is that one agent starts using a given action more often, which quickly makes the miscoordinated action a bad deal for the other agent.

Figure 4 shows the Q-value of two LMRL2 agents playing the Penalty game, which has two miscoordinated equillibria, $\langle a, a\rangle$ and $\langle c, c\rangle$. During the first several hundred iterations, the Q-value of both actions increases due to lenience, though neither reaches the actual value of 10, due to a low lenience factor ($\theta = 0.1$). Thereafter, the Q-value of both actions for both agents start to drop, as lenience starts to wear off. At some point, action a is slightly preferred by at least one agent, which at this point is sufficient to quickly lock the agents into the equilibrium $\langle a, a\rangle$.

If learners are being lenient for a very long time, this lock-in could also take a long time: but it will eventually happen. In our experiments, we have found that low lenience is not necessary: even with a high value of θ, lock-in to a given equilibrium will occur at some point. We experimented with θ values of 1, 10, 100, 1000, 10,000, and 100,000 on the Penalty game: in every case, LMRL2 worked around miscoordination and found the correct policy in 1000 out of 1000 runs.

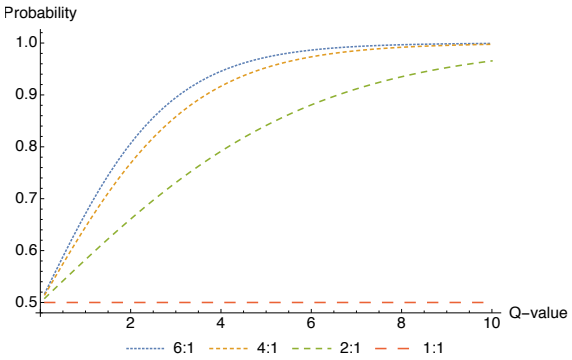


Figure 6: Given two actions A and B, with Q-values $Q_A \geq Q_B$, the probability of selecting action A over B using Boltzmann Selection, for various settings of Q_A. The different lines reflect the ratio of $Q_A : Q_B$, that is, how much larger Q_A is compared to Q_B.

Here is a typical example. Figure 5 shows the Q-values of the agent in the same game but with *extreme* lenience level $\theta = 10^7$. In this run, both miscoordinated equilibria have converged to near 10, and stay like this for a long time, but at about 6500 iterations, Agent 1 suddenly collapses to action a. This then forces Agent 2 to collapse to action a about 250 iterations later.

6.3 Exploration Strategy

A reinforcement learner may be roughly divided into two parts, the *learning policy* (or *action selection*) and the *update rule* (Singh et al., 2000). When applied to cooperative MARL, significant effort has been expended in the literature on the update rule, but the learning policy has received less attention; but coordination among learners involves both parts. Here we examine LMRL2's choices of learning policy as part of its exploration strategy.

LMRL2 begins using a modified version of Boltzmann Selection (Kaelbling et al., 1996), but when the temperature drops below some minimum value, it suddenly jumps to a fully greedy strategy (that is, $\epsilon = 0$). Our original reason to have LMRL2 do this was that with low temperatures, Boltzmann Selection is subject to floating-point overflow problems. However we have found that this scheme also happens to perform quite well compared to a traditional Boltzmann Selection algorithm. We begin here with an analysis of Boltzmann Selection as used in LMRL2, and particularly with the use of the ω action-selection tuning parameter. We then go on to some analysis of why jumping to a fully greedy strategy later on seems to be an effective mechanism.

Boltzmann Selection Strategy In two of the games, LMRL2 must adjust the parameter ω to achieve good performance. One of the difficulties with Boltzmann Selection is that when Q-values are close (and importantly, when they are all small), Boltzmann Selection does not distinguish among them well (Kaelbling et al., 1996). For example, Figure 6 shows a simple repeated game with two actions A and B, where the Q value of action A

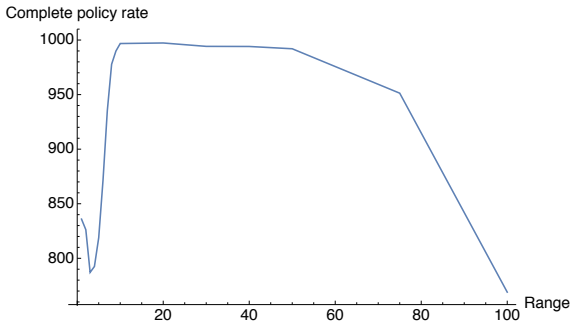


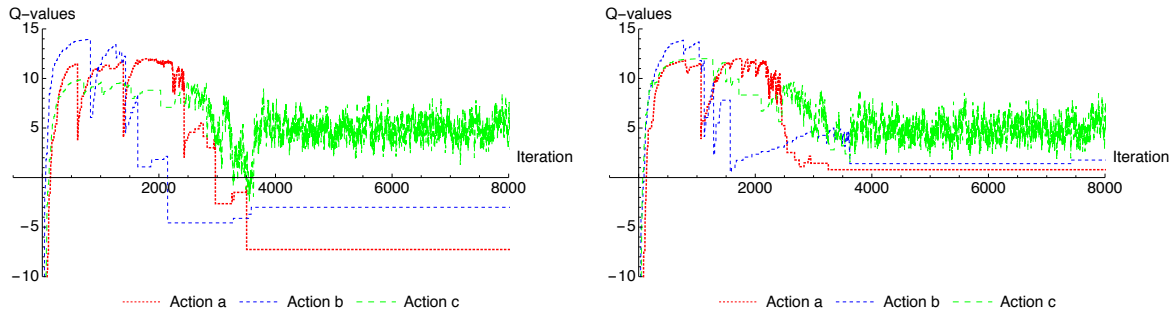
Figure 7: Rate of complete solutions for the LMRL2 in the Common Interest game, using Ranked Boltzmann Selection with values of n ranging from 1 to 100. Each point reflects the average of 1000 independent runs.

Test Problem	Original LMRL2 Results	LRML2 with Ranked Boltzmann Selection
Boutilier	**10000**/**10000**	**10000**/**10000**
Common Interest	**9971**	**9968**
Gradient 1	8407/**10000**	**9983**/**10000**
Gradient 2	**548**/ **9997**	**502**/ 9431
Heaven and Hell	**9991**/**10000**	**9996**/ **9997**
RO 1	**9368**/**10000**	8593/**10000**
RO 2	**9999**/**10000**	5267/ 6390
RO 3	**7332**/ **7332**	1739/ 1739
Climb	**9999**	9971
Climb-PS	9930	**9990**
Climb-FS	9016	**9217**
Penalty	**9999**	**9993**

Table 6: Comparison of original results from Table 5 with Ranked Boltzmann Selection using $w = 10$. Boldface values indicate methods which were not statistically significantly different with the highest-performing method for a given problem.

is some n times that of B (that is, the ratio between the two is $n : 1$). We note that even for large ratio differences between the two actions, as the Q-values become relatively small, Boltzmann quickly approaches uniform selection (0.5 probability for each). We believe this explains the necessity to change ω for LMRL2 in the Common Interest game: it has very small rewards, hence very small Q-values.

To experiment with this, we designed a new version of Boltzmann Selection called *Ranked Boltzmann Selection*. Here, when selecting an action, we first rank-ordered the possible actions by their current Q-values, breaking ties arbitrarily. We then stretched the rank orderings uniformly under some maximum value w. For example, if there were four actions, their values would be set to $1/4w$, $1/2w$, $3/4w$, and w. We then performed Boltzmann Selection using these resulting values. The point of Ranked Boltzmann Selection is that, given a value of n, it is insensitive to the exact Q-values in question, and thus to "close" Q-values among actions. w is essentially a rigorous, tunable replacement for ω in the ranked case.



(a) Agent One (b) Agent Two

Figure 8: Q-values of actions of two agents using Boltzmann Selection in the Fully Stochastic Climb game, with no minimum temperature. Note that optimal joint action $\langle a, a\rangle$ is abandoned at about step 2500 in favor of suboptimal joint action $\langle c, c\rangle$.

We tested this procedure on the Common Interest game for various values of w from 1 to 100, with 1000 trials each. Figure 7 shows the average rate of complete policy solutions. As can be seen, the algorithm performed much better for certain values than others: and for sufficiently low values of w it performed poorly.

We hoped that Ranked Boltzmann Selection might permit us to eliminate ω entirely as a parameter, since with a good selection of w it would be insensitive to variations among Q-values. To this end, we chose the highest-performing setting from the previous experiment ($w = 10$) and applied it to every test problem. Table 6 compares this against the original results from Table 5. Unfortunately, although some games saw an improvement, many others dropped significantly in performance. Indeed for RO3, which also required a tuned ω value, Ranked Boltzmann Selection performed quite poorly.

Uncoordinated Exploration and Greedy Strategies Uncoordinated exploration in independent learners occurs when one or more agents is selecting an explorative action while the remaining agents are selecting exploitative (greedy) actions. As a result, exploitative agents may receive misleadingly poor reward due to other agents' exploration. As the number of agents increases, this becomes much more likely. For example, with an ϵ-greedy strategy, with n agents in the game, the probability that all agents are exploit is $(1-\epsilon)^n$, and the probability that all the agents are explore is ϵ^n. As n increases, these probabilities become exponentially smaller.

Besides posing a scaling problem, uncoordinated exploration can also cause a difficulty we call *optimal policy destruction*. Here, the agents have found and converged to the optimal policy, but because certain agents then choose to do exploration, the learning process is destabilized and the agents as a whole wind up re-converging to some inferior solution. This may be particularly easy in relative overgeneralization scenarios, and in strongly stochastic games, where optimal solutions can be made to look bad by a single wayward agent or random reward.

Figure 8 shows a situation like this. Here we used a pure Boltzmann Selection strategy (that is, no minimum temperature) in the Fully Stochastic Climb game. We can see that at

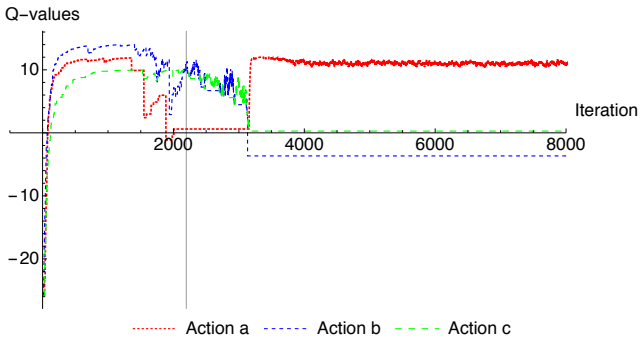


Figure 9: Q-values of actions of one LMRL2 agent in the Fully Stochastic Climb game. The vertical bar indicates where the agent commits to a fully greedy selection strategy, even though the optimal action (a) has not yet been discovered.

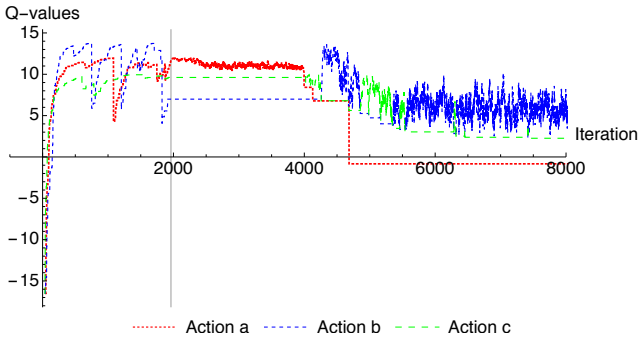


Figure 10: Q-values of actions of one LMRL2 agent in the Fully Stochastic Climb game. The vertical bar indicates where the agent commits to a fully greedy selection strategy.

around 2000 steps, Action a has the highest Q-value for both agents ($\langle a, a\rangle$ is the optimal joint solution). If we switched to fully greedy selection at this point, the agent would stick with this optimal policy. But in this experimental run, the Q-value then decreases enough that it can never recover, and the agents reconverge to $\langle c, c\rangle$. The only plausible source of this optimal policy destruction phenomenon is the uncoordinated exploration from the Boltzmann Selection strategy.

This lock-in would naturally occur when agents are no longer very lenient, but are still fairly explorative. We believe this is the reason that jumping to fully greedy selection strategy is effective in LMRL2: it occurs at the point where optimal policy destruction often shows up. It's possible to converge to the correct solution even when lock-in has occurred *before* discovery of the solution. If the agents lock into suboptimal actions, repeated play may still bring the Q-values low enough that greedy selection then selects a different joint action. This "optimism in the face of uncertainty" (Kaelbling et al., 1996) technique is widely used in many RL algorithms. Figure 9 shows an example run where has this occurred. Action

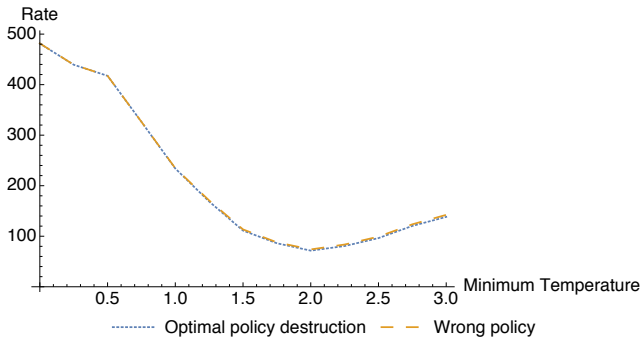


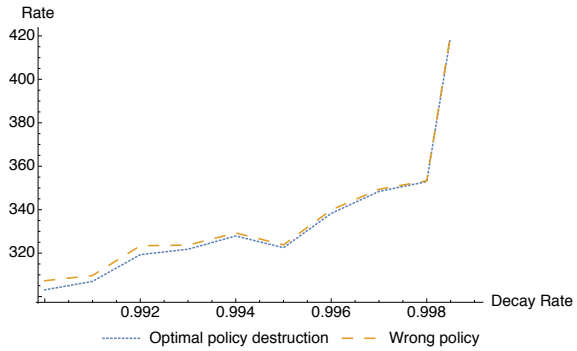
Figure 11: Optimal policy destruction and wrong-policy rates (out of 1000 runs) for different minimum temperature settings. Results shown are the average of ten 1000-run trials.

a is the optimal action. After the greedy selection began, only the action with the highest Q-value is updated (and this is presently not a). Eventually the other actions worsen to the point that a is then tried.

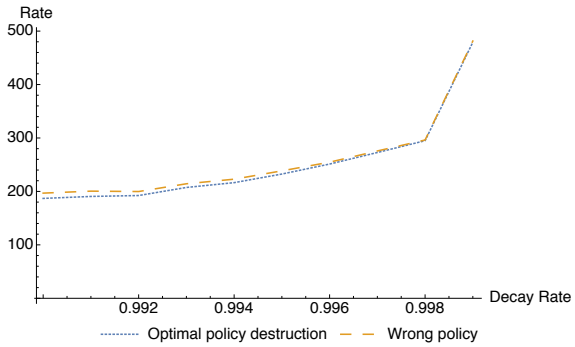
However, this greedy selection lock-in is not a silver bullet for two reasons. First, agents will not adopt the greedy strategy simultaneously: one will become greedy while the other is still exploring. Second, occasionally when the reward function of the optimal Nash Equilibrium has a very large variance, or punishment for uncoordinated behavior is very high, the Q-value of the optimal action can also accidentally be pulled down irrevocably. Figure 10 shows an example. Here we see that the Q-value of action a drops several times after the greedy selection begins, eventually causing other actions to be preferred.

However, generally the greedy strategy usually helps to reduce the rate of optimal policy destruction, particularly when the right minimum temperature is chosen (hence the right lock-in time). Figure 11 shows the optimal policy destruction and incorrect-policy rates, out of 1000 runs, for different minimum temperatures. It is clear that the two rates are strongly linked, and with a suitable minimum temperature (here 2.0), by reducing optimal policy destruction we think we can also eliminate most incorrect policies.

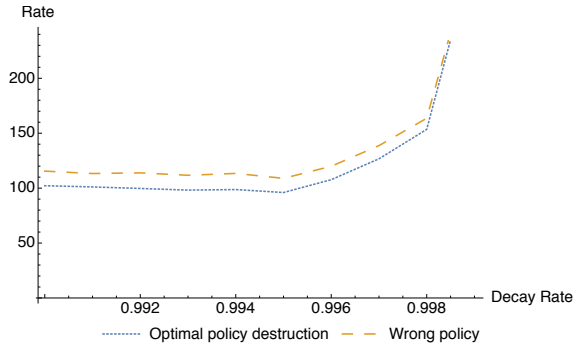
We also studied optimal policy destruction under different combinations of decay rate and minimum temperature. Figure 12 shows the resulting optimal policy destruction and wrong-policy rates. We note that many of the plots form partially bowl-shaped curves, which suggests that there is reasonable range of temperature decay for various minimum temperatures: often this is somewhere around $0.996 - 0.998$. This can be seen as the complement to Figure 11, where we had a range of good minimum temperatures for a fixed decay. However with very low minimum temperatures, the "lowest" points on curve simply slide towards the minimum temperature decay rate. This is expected, since a lower minimum temperature means later lock-in time, and thus a smaller decay rate is preferred. We also note that the difference between the optimal policy destruction and wrong-policy rates can become large, but will rapidly converge after reaching the "lowest" point on the curve. We think the reason for this is that the lower temperature decay rate drops the temperature too rapidly, thus the "optimal action" cannot reach its "optimum" given the short lenience period.



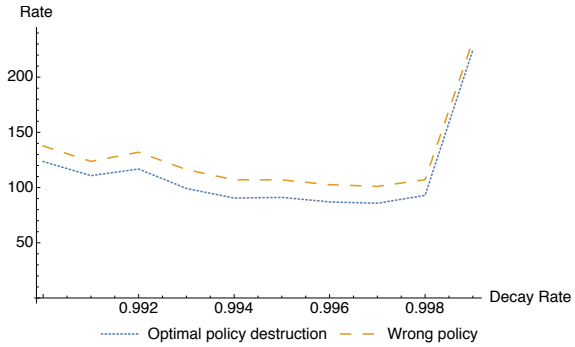
(a) minimum temperature = 0.5



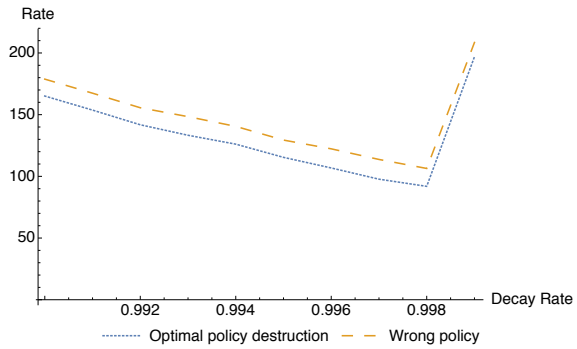
(b) minimum temperature = 1.0



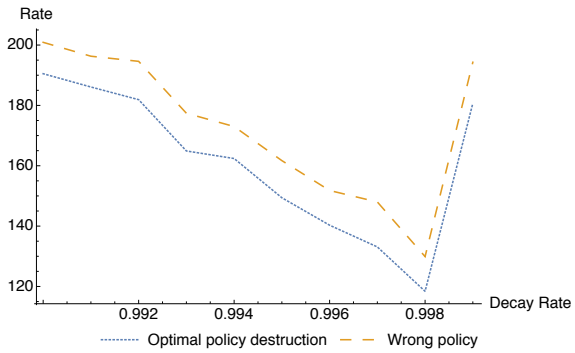
(a) minimum temperature = 1.5



(b) minimum temperature = 2.0



(a) minimum temperature = 2.5



(b) minimum temperature = 3.0

Figure 12: Optimal policy destruction and wrong-policy rates (out of 1000 runs) for different minimum temperature settings with different temperature decay rate. Results shown are the average of ten 1000-run trials.

Test Problem	Without Latching	With Latching
Boutilier	**10000**/**10000**	**10000**/**10000**
Common Interest	**9971**	**9955**
Gradient 1	**8407**/**10000**	7851/ 9834
Gradient 2	**548**/ **9997**	374/ 8775
Heaven and Hell	**9991**/**10000**	**10000**/**10000**
RO 1	9368/**10000**	**9507**/**10000**
RO 2	**9999**/**10000**	**9999**/**10000**
RO 3	**7332**/ **7332**	767/ 767

Table 7: Learning result of using latching in games with same parameter settings.

6.4 Latching

One of the critical features of LMRL2 is its propagation of temperatures from state to state, notionally in order to keep earlier states "hot" long enough to be lenient to late propagated utilities. Given a previous state s, action a, and next state s', this is done as:

$$T(s,a) \leftarrow \delta \times \begin{cases} (1-\tau)T(s,a) + \tau \operatorname{mean}_{a'} T(s',a') & \text{if } s' \text{ is not the end state (if any)} \\ T(s,a) & \text{else} \end{cases}$$

This can not only heat up states, but cool them down as well, which would seem to be an undesirable quality. We have considered an alternative (called *latching*), where states can only be heated up:

$$T(s,a) \leftarrow \delta \times \begin{cases} (1-\tau)T(s,a) + \tau \operatorname{mean}_{a'} T(s',a') & \text{if } s' \text{ is not the end state (if any)} \\ & \quad \text{and } T(s,a) < \operatorname*{mean}_{a'} T(s',a') \\ T(s,a) & \text{else} \end{cases}$$

It turns out that this method does not work particularly well. Table 7 compares LMRL2 with and without latching on the various stochastic games from our testbed (latching is irrelevant to repeated games). This is a very counterintuitive result. Our experiment shows that latching only does better in two cases, and considerably worse in the Gradient games and in RO 3. (Note that in Table 7 we purposely maintained the same (difficult) level of Bonferroni correction to be consistent with other results in the paper). Finer-grained examination on a per-state basis (not reported here) did not reveal a conclusive reason as to why.

6.5 Non-self play

Last, we were interested in knowing how well LMRL2 performed when faced with a learning method of some other kind (that is, not in self-play). This is a follow-on to similar work we did using LMRL (Sullivan et al., 2006). To test the performance of LMRL2 when paired with some other learner, we re-ran all the test problems where LMRL2 was paired with some other algorithm. As we wanted to see what would happen to LMRL2 if simply faced with a different kind of learner, and so we did *not* re-tune the parameters of the other agents, but rather kept them at their default settings (from Table 2).

	Alternative Learner					
Test Problem	LMRL2	Q-Learning	Distributed Q	Hysteretic Q	WoLF-PHC	SOoN
Boutilier	**10000**/**10000**	**10000**/**10000**	**10000**/**10000**	**10000**/**10000**	**10000**/**10000**	**10000**/**10000**
Common Interest	**9971**	9888	**9997**	9867	**9977**	9889
Gradient 1	**8407**/**10000**	574/ 1387	7458/ 8266	602/ 1383	63/ 85	1778/ 8601
Gradient 2	**548**/ **9997**	39/ 691	7/ 1119	46/ 706	30/ 376	94/ 2186
Heaven and Hell	**9991**/**10000**	9838/ **9999**	**9981**/**10000**	9830/ **9996**	9787/ **9999**	9855/**10000**
RO 1	9368/**10000**	8031/ 9625	**9998**/ **9998**	7970/ 9617	6554/ 6931	5307/ 7786
RO 2	**9999**/**10000**	0/ 1	6004/**10000**	0/ 0	0/ 0	0/ 14
RO 3	**7332**/ **7332**	899/ 899	2817/ 2817	884/ 884	530/ 530	542/ 542
Climb	**9999**	27	9933	7	0	31
Climb-PS	**9930**	3	56	6	1	26
Climb-FS	**9016**	7	2	24	22	25
Penalty	**9999**	8475	**10000**	8514	9417	**9999**

Table 8: LMRL2 performance when paired with an alternative learning algorithm. Results shown are Complete/Correct solution counts for various test problems. Note that in the Common Interest, Penalty, and various Climb games, correct solutions are by definition complete.

As shown in Table 8, for most of the games and most of the learners, LMRL2 performed best when in self-play. There was one chief exception: for two problems (RO1 and Common Interest), the combination of LMRL2 and Distributed Q-learning outperformed LMRL2 in self-play by a statistically significant margin. To understand why Distributed Q-learning would help LMRL2 in some cases, we plotted the Q-values for the two learners in State 1 of the Common Interest game. State 1 is the state that LMRL2 would normally fail. In State 1, there are two optimal Nash Equilibria, $\langle a, b\rangle$ and $\langle b, a\rangle$, and the primary difficulty in avoiding miscoordination, which which our lenient learner sometimes fails at due to the sensitivity of Boltzmann selection.

The Q-values are shown in Figure 13. It is easy to see that Distributed Q-learning converges much faster than LMRL2. After it has converged, we are effectively dealing with single-agent rather than multiagent reinforcement learning: LMRL2 is just trying to get the best result it can in an essentially fixed environment. This fast convergence doesn't restrict LMRL2's ability to find an optimal Nash Equilibrium, however, since it will converge to which ever Equilibrium Distributed Q-learning has chosen. Here, Distributed Q-learning has effectively forced LMRL2 into its decision.

This result reveals another way we might solve miscoordination. The way Distributed Q-learning normally approaches the miscoordination problem is to make an implicit deal that agents will stick with the first Nash Equilibrium they have chosen. But an alternative would be for one learner to converge to a Nash Equilibrium faster than the other learner, forcing the second learner to adopt its decision.

7. Conclusions

We have introduced LMRL2, a reinforcement learning algorithm for stochastic cooperative independent-learner games, and which can also be used for repeated games. LMRL2 is

Test Problem	LRML2 (Self Play)	Distributed Q (Self Play)	LRML2 + Distributed Q
Boutilier	**10000**/**10000**	**10000**/**10000**	**10000**/**10000**
Common Interest	9971	2080	**9997**
Gradient 1	8407/**10000**	**9999**/ **9999**	7458/ 8266
Gradient 2	**548**/ **9997**	0/ 1266	7/ 1119
Heaven and Hell	**9991**/**10000**	**10000**/**10000**	**9981**/**10000**
RO 1	9368/**10000**	**10000**/**10000**	**9998**/ **9998**
RO 2	**9999**/**10000**	9258/ 9258	6004/**10000**
RO 3	**7332**/ **7332**	2272/ 2272	2817/ 2817
Climb	**9999**	**10000**	9933
Climb-PS	**9930**	2821	56
Climb-FS	**9016**	3874	2
Penalty	**9999**	**10000**	**10000**

Table 9: Comparison of LMRL2 in self play, Distributed Q-Learning in Self-Play, and the scenario where LMRL2 controls one agent and Distributed Q-Learning controls the other. Results shown are Complete/Correct solution counts for various test problems. Note that in the Common Interest, Penalty, and various Climb games, correct solutions are by definition complete.

particularly meant to overcome the *relative overgeneralization* pathology which can appear in cooperative multiagent games of three or more actions per agent, and to do so successfully in the face of stochasticity in the reward or transition function. The algorithm is an extension of an earlier version of ours, LMRL, which was meant only for repeated games.

We compared LMRL2 against several other algorithms which may be applied to cooperative independent-learner games, both stochastic and repeated. This comparison was done over a collection of test problems drawn from the literature and our own devising, and meant to test a wide variety of scenarios and pathologies. Our results have shown that LRML2 is an effective and robust learner, generally outperforming the other algorithms on the test problems in several different ways. LMRL2 placed in the top statistical significance tier for "complete" policies for eight problems, two more times than other algorithms. Further, it placed in the top tier for "correct" policies for eleven problems, three more times than other algorithms. And LMRL2 was uniquely best for two problems. And unlike some other methods, it did so with only a small amount of tuning: though LMRL2 has many available parameters, it performs well even when almost all of them are fixed to default values.

Future Work Because it works on a temperature schedule, LMRL2 is not a particularly fast algorithm. It often takes rather more time to converge than other more greedy methods. Because we are more interested in the converged result than in the speed of learning, this was not a vital issue for us: but as future work, we wish to examine how to speed up LMRL2's convergence without hurting its performance. We also want to investigate other possible stable exploration methods which achieve coordinated exploration: while LMRL2's current exploration strategy is generally fixed, it is nonetheless overcomplicated.

We have also been surprised by our *latching* results. We would have expected to get better performance by restricting pushed-back temperature to only those situations where it would heat up the earlier states, but the results are mixed and disappointing. We intend to examine this more carefully as future work.

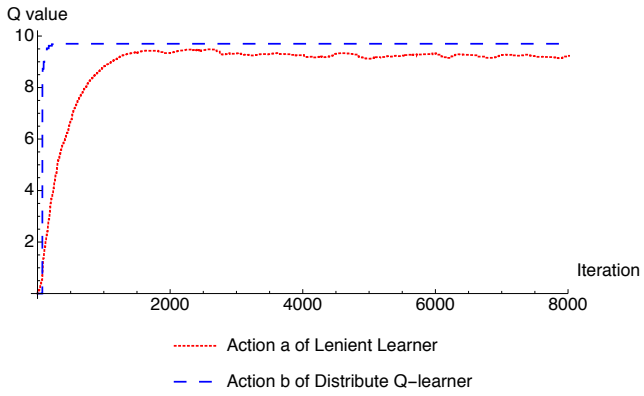


Figure 13: Q-values of two learners in State 1 of Common Interest game

Finally, we note that while the work in this paper reflects somewhat larger and more complex games than are found in much of the MARL literature, it does not scale beyond the literature in important ways. For example, while the algorithm works for any $N \geq 2$ number of agents, the test problems presented are for $N = 2$. In future work, we will extend the research to more complex and more practical domains, scaling both in terms of agent numerosity and heterogeneity, and involving continuous state and action spaces. To this end, we wish to adapt lenience with function approximation and policy gradient methods.

Acknowledgments

The authors thank the reviewers for their feedback. The work presented in this paper is supported by NSF NRI grant 1317813.

Appendix A. Game Defintions

Below we provide tables defining each game in this study, accompanied by illustrations. Each table shows a state S, two agent actions A_1 and A_2, a resulting joint *reward*, and a resulting transition to new state S'.

Actions Actions may be explicitly labeled, such as b or c, or they may have the word *any*, meaning "any action", or (for agent 2) they may have the word *same*, meaning "the same action that agent 1 selected", or they may have the word *other*, meaning "a different action than agent 1 selected". **Boldface** actions indicate members of a complete policy. **Boldface** actions shown in circled states (such as②) indicate members of a correct policy.

States, Rewards, and Transitions The *start state* is always state 1. The absorbing *end state* (if any) is indicated with the word "end". All other states are numbered. If indicated with a single number (or "end"), then rewards and transitions are deterministic. Otherwise, a distribution is shown. For example, 1 (10%), 2 (90%) means "10% of the time choose 1, 90% of the time choose 2".

Illustrations Tables are accompanied by illustrations, which show the games as finite-state automata. States are shown by their numbers: the start state is 1. The absorbing state, if any, is *End*. Each state has a two-player game reward matrix corresponding to the actions

$\langle a,a\rangle$	$\langle a,b\rangle$
$\langle b,a\rangle$	$\langle b,b\rangle$

or

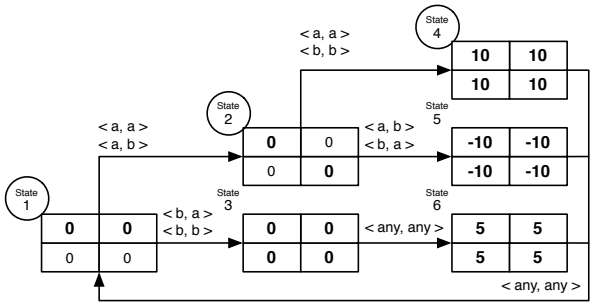
$\langle a,a\rangle$	$\langle a,b\rangle$	$\langle a,c\rangle$
$\langle b,a\rangle$	$\langle b,b\rangle$	$\langle b,c\rangle$
$\langle c,a\rangle$	$\langle c,b\rangle$	$\langle c,c\rangle$

actions for players $\langle A_1, A_2\rangle$ respectively. Values in the matrices indicate rewards for joint actions. If the values are of the form x/y, this indicates that 50% of the time value x is rewarded, and 50% of the time y is rewarded. **Boldface** values indicate joint actions which are part of correct policies. Some states have their state number circled: **boldface** values in these states are part of complete policies. Directed transition edges between states are labeled with the joint actions which trigger those transitions. In some cases a joint action will trigger a transition with a certain probability, else it will trigger some other transition. Transitions may also be labeled ⟨any, any⟩, indicating that all joint actions trigger that transition.

Boutilier

From (Boutilier, 1999).

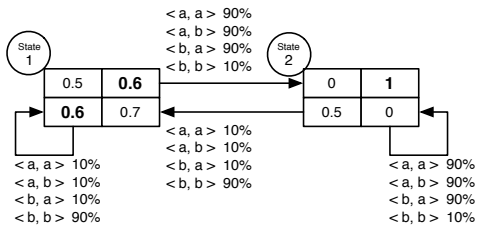
S	A_1	A_2	Reward	S'
①	**a**	**any**	0	2
	b	any	0	3
②	**any**	**same**	0	4
	any	other	0	5
3	**any**	**any**	0	6
④	**any**	**any**	10	1
5	**any**	**any**	-10	1
6	**any**	**any**	5	1



Common Interest

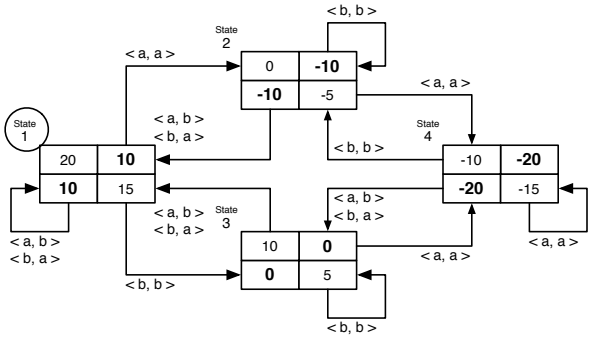
From (Vrancx et al., 2008).

S	A_1	A_2	Reward	S'
①	a	a	0.5	1 (10%), 2 (90%)
	any	**other**	0.6	1 (10%), 2 (90%)
	b	b	0.7	1 (90%), 1 (10%)
②	a	a	0	1 (10%), 2 (90%)
	a	**b**	1.0	1 (10%), 2 (90%)
	b	a	0.5	1 (10%), 2 (90%)
	b	b	0	1 (90%), 1 (10%)



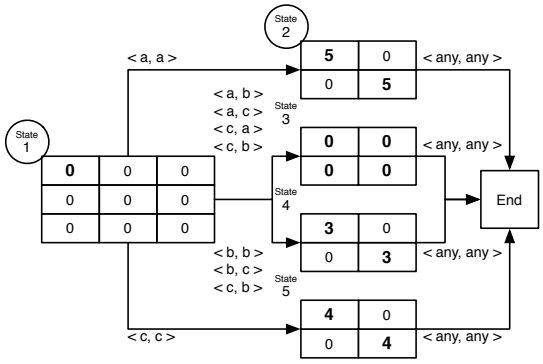
Heaven and Hell

S	A_1	A_2	Reward	S'
①	a	a	20	2
	any	**other**	10	1
	b	b	15	3
2	a	a	0	4
	any	**other**	-10	1
	b	b	-5	2
3	a	a	10	4
	any	**other**	0	1
	b	b	5	2
4	a	a	-10	4
	any	**other**	-20	3
	b	b	-15	2



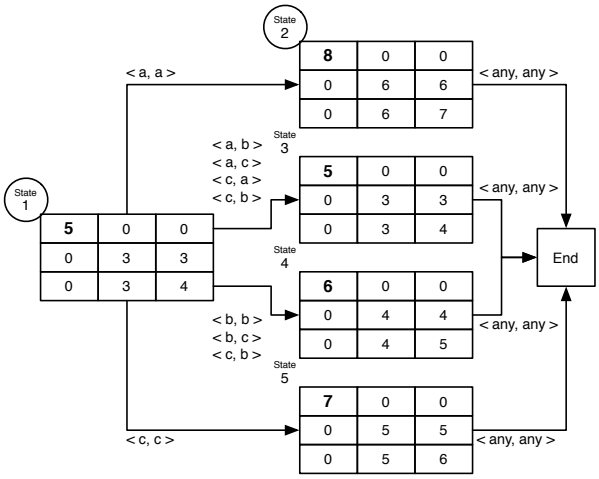
Relative Overgeneralization 1 (RO 1)

S	A_1	A_2	Reward	S'
①	**a**	**a**	0	2
	a	b	0	3
	a	c	0	3
	b	a	0	3
	b	b	0	4
	b	c	0	4
	c	a	0	3
	c	b	0	4
	c	c	0	5
②	**a**	**other**	5	end
	b	other	0	end
3	**any**	**any**	0	end
4	**a**	**other**	3	end
	b	other	0	end
5	**a**	**other**	4	end
	b	other	0	end



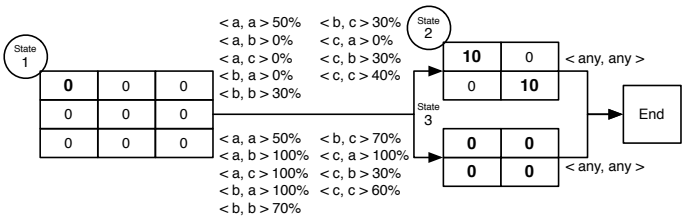
Relative Overgeneralization 2 (RO 2)

S	A_1	A_2	Reward	S'
	a	**a**	5	2
	a	b	0	3
	a	c	0	3
	b	a	0	3
①	b	b	3	4
	b	c	3	4
	c	a	0	3
	c	b	3	4
	c	c	4	5
	a	a	8	end
	a	b	0	end
	a	c	0	end
	b	a	0	end
②	b	b	6	end
	b	c	6	end
	c	a	0	end
	c	b	6	end
	c	**c**	7	end
	a	a	5	end
	a	b	0	end
	a	c	0	end
	b	a	0	end
3	b	b	3	end
	b	c	3	end
	c	a	0	end
	c	b	3	end
	c	**c**	4	end
	a	a	6	end
	a	b	0	end
	a	c	0	end
	b	a	0	end
4	b	b	4	end
	b	c	4	end
	c	a	0	end
	c	b	4	end
	c	**c**	5	end
	a	a	7	end
	a	b	0	end
	a	c	0	end
	b	a	0	end
5	b	b	5	end
	b	c	5	end
	c	a	0	end
	c	b	5	end
	c	**c**	6	end



Relative Overgeneralization 3 (RO3 3)

S	A_1	A_2	Reward	S'
	a	**a**	0	2 (50%), 3 (50%)
	a	b	0	3
	a	c	0	3
	b	a	0	3
①	b	b	0	2 (30%), 3 (70%)
	b	c	0	2 (30%), 3 (70%)
	c	a	0	3
	c	b	0	2 (30%), 3 (70%)
	c	c	0	2 (40%), 3 (60%)
②	**a**	**other**	10	end
	b	other	0	end
3	**any**	**any**	0	end

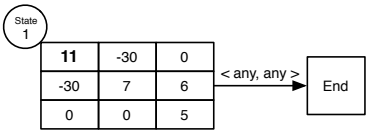


Climb, Climb Partially Stochastic, and Climb Fully Stochastic

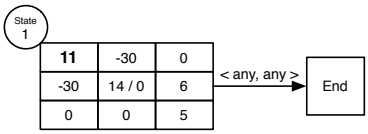
Climb is from (Claus and Boutilier, 1998). Climb Partially Stochastic and Climb Fully Stochastic are from (Kapetanakis and Kudenko, 2002).

			Reward			
				Climb		*Climb*	
S	A_1	A_2	*Climb*	*Partially Stochastic*		*Fully Stochastic*	S'
	a	**a**	11	11	10 (50%),	12 (50%)	end
	a	b	-30	-30	5 (50%),	-65 (50%)	end
	a	c	0	0	8 (50%),	-8 (50%)	end
	b	a	-30	-30	5 (50%),	-65 (50%)	end
①	b	b	7	14 (50%), 0 (50%)	14 (50%),	0 (50%)	end
	b	c	6	6	12 (50%),	0 (50%)	end
	c	a	0	0	5 (50%),	-5 (50%)	end
	c	b	0	0	5 (50%),	-5 (50%)	end
	c	c	5	5	10 (50%),	0 (50%)	end

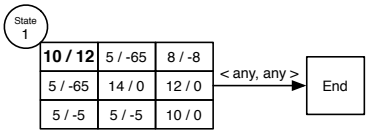
Climb



Climb Partially Stochastic



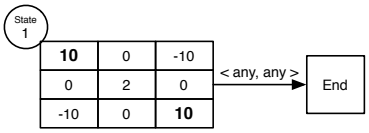
Climb Fully Stochastic



Penalty

From (Claus and Boutilier, 1998).

S	A_1	A_2	Reward	S'
	a	**a**	10	end
	a	b	0	end
	a	c	-10	end
	b	a	0	end
①	b	b	2	end
	b	c	0	end
	c	a	-10	end
	c	b	0	end
	c	**c**	10	end

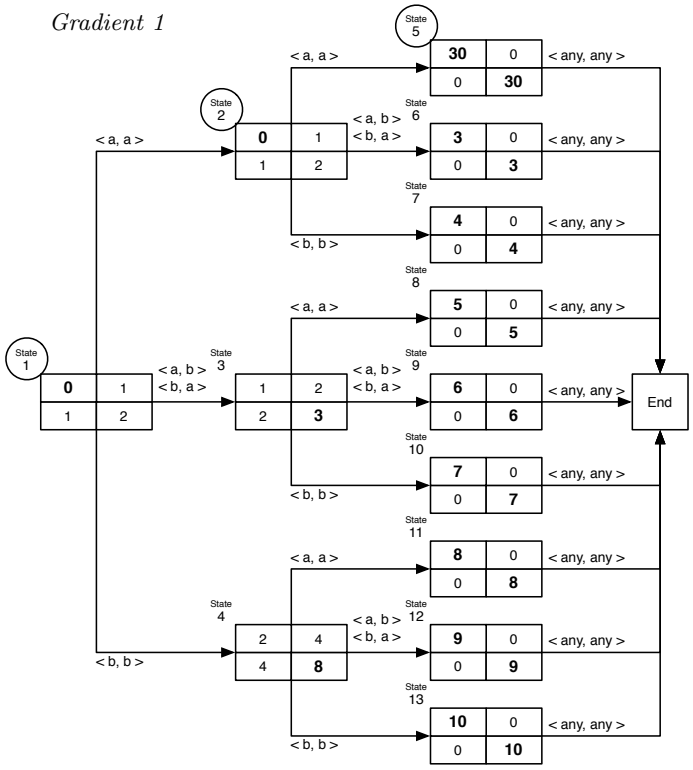


Gradient 1, Gradient 2

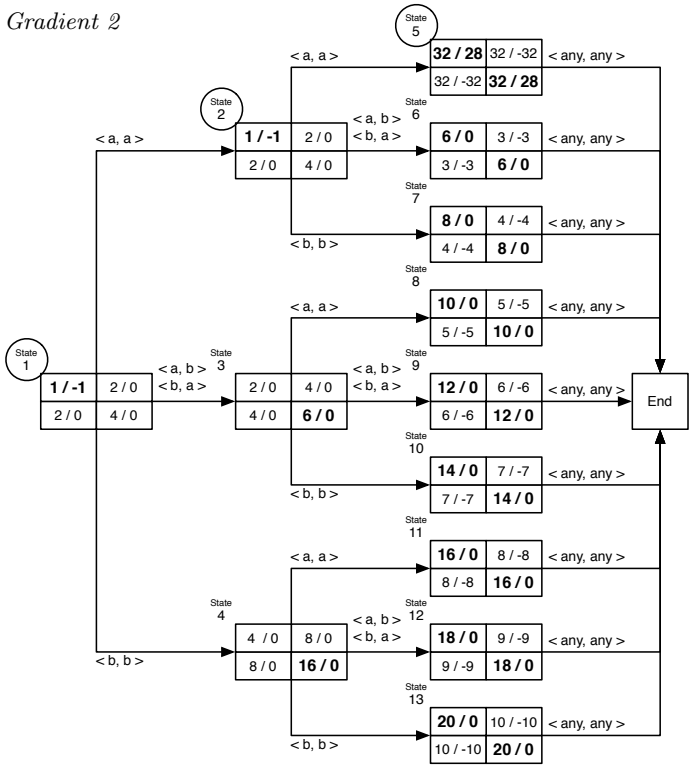
			Reward			
S	A_1	A_2	*Gradient 1*		*Gradient 2*	S'
①	**a**	**a**	0	1 (50%),	-1 (50%)	2
	any	other	1	2 (50%),	0 (50%)	3
	b	b	2	4 (50%),	0 (50%)	4
②	**a**	**a**	0	1 (50%),	-1 (50%)	5
	any	other	1	2 (50%),	0 (50%)	6
	b	b	2	4 (50%),	0 (50%)	7
3	a	a	1	2 (50%),	0 (50%)	8
	any	other	2	4 (50%),	0 (50%)	9
	b	**b**	3	6 (50%),	0 (50%)	10
4	a	a	2	(50%),	0 (50%)	11
	any	other	4	8 (50%),	0 (50%)	12
	b	**b**	8	16 (50%),	0 (50%)	13
⑤	**any**	**same**	30	32 (50%),	28 (50%)	end
	any	other	0	32 (50%),	-32 (50%)	end
6	**any**	**same**	3	6 (50%),	0 (50%)	end
	any	other	0	3 (50%),	-3 (50%)	end
7	**any**	**same**	4	8 (50%),	0 (50%)	end
	any	other	0	4 (50%),	-4 (50%)	end
8	**any**	**same**	5	10 (50%),	0 (50%)	end
	any	other	0	5 (50%),	-5 (50%)	end
9	**any**	**same**	6	12 (50%),	0 (50%)	end
	any	other	0	6 (50%),	-6 (50%)	end
10	**any**	**same**	7	14 (50%),	0 (50%)	end
	any	other	0	7 (50%),	-7 (50%)	end
11	**any**	**same**	8	16 (50%),	0 (50%)	end
	any	other	0	8 (50%),	-8 (50%)	end
12	**any**	**same**	9	18 (50%),	0 (50%)	end
	any	other	0	9 (50%),	-9 (50%)	end
13	**any**	**same**	10	20 (50%),	0 (50%)	end
	any	other	0	10 (50%),	-10 (50%)	end

(Figures on following page)

Gradient 1



Gradient 2



References

Sherief Abdallah and Victor Lesser. A multiagent reinforcement learning algorithm with non-linear dynamics. *Journal of Artificial Intelligence Research*, 33:521–549, 2008.

Noa Agmon, Samuel Barrett, and Peter Stone. Modeling uncertainty in leading ad hoc teams. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 397–404, 2014.

Monica Babes, Munoz Enrique Cote De, and Michael L. Littman. Social reward shaping in the prisoner's dilemma. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, volume 3, pages 1389–1392, 2008.

Bikramjit Banerjee and Jing Peng. Adaptive policy gradient in multiagent learning. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 686–692, 2003.

Samuel Barrett and Peter Stone. Cooperating with unknown teammates in complex domains: A robot soccer case study of ad hoc teamwork. In *AAAI Conference on Artificial Intelligence*, 2015.

Daan Bloembergen, Michael Kaisers, and Karl Tuyls. Lenient frequency adjusted Q-learning. In *Benelux Conference on Artificial Intelligence (BNAIC)*, 2010.

Craig Boutilier. Sequential optimality and coordination in multiagent systems. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 1, pages 478–485, 1999.

Michael Bowling. Convergence and no-regret in multiagent learning. In *Advances in Neural Information Processing Systems (NIPS)*, volume 17, pages 209–216, 2005.

Michael Bowling and Manuela Veloso. Convergence of gradient dynamics with a variable learning rate. In *International Conference on Machine Learning (ICML)*, pages 27–34, 2001a.

Michael Bowling and Manuela Veloso. Rational and convergent learning in stochastic games. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 2, pages 1021–1026, 2001b.

Michael Bowling and Manuela Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136(2):215–250, 2002.

Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 38(2):156–172, 2008.

Doran Chakraborty and Peter Stone. Cooperating with a markovian ad hoc teammate. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1085–1092, 2013a.

Doran Chakraborty and Peter Stone. Multiagent learning in the presence of memory-bounded agents. *Autonomous Agents and Multiagent Systems*, 2013b.

Georgios Chalkiadakis and Craig Boutilier. Coordination in multiagent reinforcement learning: A Bayesian approach. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 709–716, 2003.

Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *National Conference on Artificial Intelligence*, pages 746–752, 1998.

Vincent Conitzer and Tuomas Sandholm. AWESOME: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. *Machine Learning*, 67(1-2):23–43, 2007.

Thomas G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13(1):227–303, November 2000.

Nancy Fulda and Dan Ventura. Predicting and preventing coordination problems in cooperative Q-learning systems. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 780–785, 2007.

Katie Genter, Tim Laue, and Peter Stone. The robocup 2014 SPL drop-in player competition: Encouraging teamwork without pre-coordination. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1745–1746, 2015.

Mohammad Ghavamzadeh and Sridhar Mahadevan. Learning to communicate and act using hierarchical reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, volume 3, pages 1114–1121, 2004.

Mohammad Ghavamzadeh, Sridhar Mahadevan, and Rajbala Makar. Hierarchical multi-agent reinforcement learning. *Autonomous Agents and Multiagent Systems*, 13(2):197–229, 2006.

Amy Greenwald, Keith Hall, and Roberto Serrano. Correlated Q-learning. In *AAAI Spring Symposium*, volume 3, pages 242–249, 2003.

Junling Hu and Michael P. Wellman. Nash Q-learning for general-sum stochastic games. *Journal of Machine Learning Research*, 4:1039–1069, 2003.

Tommi Jaakkola, Michael I. Jordan, and Satinder P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6:1185–1201, 1994.

Amir Jafari, Amy Greenwald, David Gondek, and Gunes Ercal. On no-regret learning, fictitious play, and Nash equilibrium. In *International Conference on Machine Learning (ICML)*, volume 1, pages 226–233, 2001.

Leslie Pack Kaelbling, Michael L. Littman, and Andrew M. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4(1):237–285, 1996.

Michael Kaisers and Karl Tuyls. Frequency adjusted multi-agent Q-learning. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, volume 1, pages 309–316, 2010.

Spiros Kapetanakis and Daniel Kudenko. Reinforcement learning of coordination in cooperative multi-agent systems. In *National Conference on Artificial Intelligence*, pages 326–331, 2002.

Ville Könönen. Asymmetric multiagent reinforcement learning. *Web Intelligence and Agent Systems*, 2:105–121, 2004.

Martin Lauer and Martin Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *International Conference on Machine Learning (ICML)*, pages 535–542, 2000.

Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *International Conference on Machine Learning (ICML)*, volume 94, pages 157–163, 1994.

Michael L. Littman. Friend-or-foe Q-learning in general-sum games. In *International Conference on Machine Learning (ICML)*, volume 1, pages 322–328, 2001a.

Michael L. Littman. Value-function reinforcement learning in markov games. *Cognitive Systems Research*, 2(1):55–66, 2001b.

Rajbala Makar, Sridhar Mahadevan, and Mohammad Ghavamzadeh. Hierarchical multi-agent reinforcement learning. In *Proceedings of the 5th International Conference on Autonomous Agents (AGENT)*, pages 246–253, 2001.

Laëtitia Matignon, Guillaume J. Laurent, and Nadine Le Fort-Piat. Hysteretic Q-learning: an algorithm for decentralized reinforcement learning in cooperative multi-agent teams. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 64–69. IEEE, 2007.

Laëtitia Matignon, Guillaume J. Laurent, and Nadine Le Fort-Piat. A study of FMQ heuristic in cooperative multi-agent games. In *Workshop on Multi-Agent Sequential Decision Making in Uncertain Multi-Agent Domains (at AAMAS)*, volume 1, pages 77–91, 2008.

Laëtitia Matignon, Guillaume J. Laurent, and Nadine Le Fort-Piat. Coordination of independent learners in cooperative markov games. Technical report, Institut FEMTO-ST, UniversitPé de Franche-Comté, https://hal.archives-ouvertes.fr/hal-00370889/document, 2009.

Laëtitia Matignon, Guillaume J. Laurent, and Nadine Le Fort-Piat. Independent reinforcement learners in cooperative Markov games: a survey regarding coordination problems. *The Knowledge Engineering Review*, 27(01):1–31, 2012.

Liviu Panait. *The Analysis and Design of Concurrent Learning Algorithms for Cooperative Multiagent Systems*. PhD thesis, George Mason University, Fairfax, Virginia, 2006.

Liviu Panait, R. Paul Wiegand, and Sean Luke. A sensitivity analysis of a cooperative coevolutionary algorithm biased for optimization. In *Genetic and Evolutionary Computation Conference (GECCO)*, pages 587–584, 2004.

Liviu Panait, Sean Luke, and R. Paul Wiegand. Biasing coevolutionary search for optimal multiagent behaviors. *IEEE Transactions on Evolutionary Computation*, 10(6):629–645, 2006a.

Liviu Panait, Keith Sullivan, and Sean Luke. Lenient learners in cooperative multiagent systems. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2006b.

Liviu Panait, Karl Tuyls, and Sean Luke. Theoretical advantages of lenient learners: an evolutionary game theoretic perspective. *Journal of Machine Learning Research*, 9:423–457, March 2008.

Liviu Panait, Keith Sullivan, and Sean Luke. Lenience towards teammates helps in cooperative multiagent learning. Technical Report GMU-CS-TR-2013-2, Department of Computer Science, George Mason University, 4400 University Drive MSN 4A5, Fairfax, VA 22030-4444 USA, 2013.

Mitchell A. Potter and Kenneth A. De Jong. A cooperative coevolutionary approach to function optimization. In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer, editors, *Parallel Problem Solving from Nature III (PPSN)*, volume 866 of *Lecture Notes in Computer Science*, pages 249–257, 1994.

Yoav Shoham, Rob Powers, and Trond Grenager. On the agenda(s) of research on multi-agent learning. In *AAAI Fall Symposium on Artificial Multiagent Learning*, 2004.

Satinder P. Singh, Tommi Jaakkola, Michael L. Littman, and Csaba Szepesvári. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 38 (3):287–308, 2000.

Peter Stone and Sarit Kraus. To teach or not to teach?: Decision making under uncertainty in ad hoc teams. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, volume 1, pages 117–124, 2010.

Peter Stone, Gal A. Kaminka, Sarit Kraus, and Jeffrey S. Rosenschein. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *AAAI Conference on Artificial Intelligence*, 2010.

Keith Sullivan, Liviu Panait, Gabriel Balan, and Sean Luke. Can good learners always compensate for poor learners? In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 804–806, 2006.

Gerald Tesauro. Extending Q-learning to general adaptive multi-agent systems. In *Advances in Neural Information Processing Systems (NIPS)*, 2004.

Peter Vrancx, Karl Tuyls, and Ronald Westra. Switching dynamics of multi-agent learning. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, volume 1, pages 307–313, 2008.

Xiaofeng Wang and Tuomas Sandholm. Reinforcement learning to play an optimal nash equilibrium in team Markov games. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1571–1578, 2002.

Michael Weinberg and Jeffrey S. Rosenschein. Best-response multiagent learning in non-stationary environments. In *International Conference on Autonomous Agents and Multi-agent Systems (AAMAS)*, volume 3, pages 506–513, 2004.

Michael P. Wellman and Junling Hu. Conjectural equilibrium in multiagent learning. *Machine Learning*, 33(2-3):179–200, 1998.

Rudolph Paul Wiegand. *An Analysis of Cooperative Coevolutionary Algorithms*. PhD thesis, Department of Computer Science, George Mason University, 2004.

Aaron Wilson, Alan Fern, and Prasad Tadepalli. Bayesian policy search for multi-agent role discovery. In *AAAI Conference on Artificial Intelligence*, 2010.

Chongjie Zhang and Victor Lesser. Multi-agent learning with policy prediction. In *AAAI Conference on Artificial Intelligence*, pages 927–934, 2010.