Hedged learning: Regret-minimization with learning experts

Yu-Han Chang YCHANG@CSAIL.MIT.EDU
CSAIL, Massachusetts Institute of Technology, 32 Vassar Street, Cambridge, MA 02139 USA

Leslie Pack Kaelbling LPK@CSAIL.MIT.EDU
CSAIL, Massachusetts Institute of Technology, 32 Vassar Street, Cambridge, MA 02139 USA

Abstract

In non-cooperative multi-agent situations, there cannot exist a globally optimal, yet opponent-independent learning algorithm. Regret-minimization over a set of strategies optimized for potential opponent models is proposed as a good framework for deciding how to behave in such situations. Using longer playing horizons and experts that learn as they play, the regret-minimization framework can be extended to overcome several shortcomings of earlier approaches to the problem of multi-agent learning.

1. Introduction

In recent years, there has been increasing interest in multi-agent learning. A large body of this work tries to marry game theoretic concepts such as Nash equilibrium to learning in various types of games. However, as Reinhard Selton, 1995 Economics Nobel Prize winner (along with Nash and Harsanyi) once wrote in a personal communication, "Game theory is for proving theorems, not for playing games."

What does this mean for AI researchers interested in designing algorithms that learn to play good strategies in multi-agent domains? There are three issues related to applying equilibrium results from game theory directly: computational efficiency, learning dynamics, and opponent assumptions. In some cases, straight computation of Nash or correlated Nash equilibria in a given game is quite useful, and there have been a number of recent advances exploiting game structure to compute such equilibria efficiently. Moreover, some of these algorithms use learning dynamics to converge to correlated equilibria, thus addressing the core of Selton's complaint: while the classic Nash equilibrium is a stable point, it is not necessarily the stable point of reasonable system dynamics.

Although these advances begin to resolve the issues of computational efficiency and learning dynamics, the problem of opponent assumptions remains more elusive. When we face an unknown opponent, we have no guarantee that the opponent will be playing strategically (as in classical Nash results), following any particular learning rule (as in the more recent work on correlated equilibria), or even playing vaguely intelligently. Perhaps the opponent has broken sensors or actuators, or lacks some crucial information. If we try to play our half of an equilibrium strategy, we may end up worse off if the opponent does not play its half of the strategy. To counter this problem, we would like to be able to model as many different types of potential opponents as possible. When one of our opponent models is correct, we would like to be performing optimally with respect to that model. If none of our models is correct, we would still like to avoid performing too poorly. The framework of regret minimizing, or hedging, algorithms provides a useful setup for approaching this problem. Using this framework, we can simultaneously achieve both of these goals: perform optimally if one of our models is correct, and still perform reasonably well if none of our models are correct. Since we know that it is impossible to design an algorithm that performs optimally with respect to all possible opponents (Nachbar & Zame, 1996), this is the best we can hope to do in a non-cooperative setting.

2. Mathematical setup

Repeated games form the simplest possible framework for studying multi-agent learning algorithms and their resulting behavior. We will focus on this setup, although most of the ideas presented in this paper can be easily extended to handle stochastic games as well.

Appearing in *Proceedings of the 22^{nd} International Conference on Machine Learning*, Bonn, Germany, 2005. Copyright 2005 by the author(s)/owner(s).

$$r_1 = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \qquad r_1 = \begin{bmatrix} 1 & -1 \\ 2 & 0 \end{bmatrix}$$

$$r_2 = -r_1 \qquad r_2 = \begin{bmatrix} 1 & 2 \\ -1 & 0 \end{bmatrix}$$

(a) Matching pennies (b) Prisoner's Dilemna

Figure 1. Common examples of matrix games.

The repeated game is a generalization of the traditional one-shot or matrix game. In the one-shot game, two players meet, choose actions, receive rewards based on the simultaneous actions taken, and the game ends. Their actions and payoffs are given as a reward matrix r_i for each player, $r_i : A_1 \times A_2 \to \mathbb{R}$, where A_i is the finite set of discrete actions available to player i. Some common examples of two-player matrix games are shown in Figure 1. The special case of a purely competitive two-player game is called a *zero-sum game* and must satisfy $R_1 = -R_2$.

Each player simultaneously chooses to play a particular action $a_i \in A_i$, possibly drawn from a mixed policy $\mu_i = PD(A_i)$, which is a probability distribution over possible actions, and receives reward based on the joint action taken. We use the terms *policy* and *strategy* interchangeably.

In general, we would like to play a *best response* to the opponent's choice of action. If all players are playing best responses to the other players' strategies, then the game is said to be in *Nash equilibrium.* Once all players are playing a Nash equilibrium, no single player has an incentive to unilaterally deviate from his equilibrium policy. There is general agreement that Nash equilibrium is an appropriate solution concept for one-shot games. In contrast, for repeated games there are a range of different possibilities. We would like to be able to learn about our opponent using knowledge gained from previous periods of play. Reinforcement learning researchers initially focused their attention on learning a single stationary policy μ to play at each time period. This μ tries to maximize the learner's expected rewards in all future time periods, and is usually be based upon equilibrium strategies computed at each state of the stochastic game (Littman, 1994) (Hu & Wellman, 1998).

Modern game theory often takes a more general view of optimality in repeated games. The machine learning community has also recently begun adopting this view (Chang & Kaelbling, 2001) (de Farias & Meggido, 2004). The key difference is the treatment of the history of actions taken in the game. Recall that in the stochastic game model, we took $\mu_i = PD(A_i)$. Here we redefine $\mu_i : H \to A_i$, where $H = \bigcup_t H^t$ and H^t is the set of all possible histories of length t. Histories are observations of joint actions, $h^t = (a_i, a_{-i}, h^{t-1})$. Player i's strategy at time t is then expressed as $\mu_i(h^{t-1})$. For simplicity, we will assume $A = A_1 = A_2$.

Definition 1 *A τ-length behavioral strategy μ^τ is a mapping from all possible histories H^τ to actions $a \in A$. Let M^τ be the set of all possible τ-length behavioral strategies μ^τ.*

We note that $|M^\tau| = |A|^{|A|^{2\tau}}$. In the case where we take $H^t = H$, we could even consider learning algorithms themselves to be a possible "behavioral strategy" for playing a repeated game.

This definition of our strategy space is clearly more powerful, and allows us to define a much larger set of potential equilibria. However, when the opponent is not rational, it is no longer advantageous to find and play an equilibrium strategy. In fact, given an arbitrary opponent, the Nash equilibrium strategy may return a lower payoff than some other action. Indeed, the payoff may be even worse than the original Nash equilibrium value. Thus, we turn to regret minimization algorithms.

2.1. Regret-minimization

In repeated games, the standard regret minimization framework enables us to perform almost as well as the best action, if that single best action were played in every time period. Suppose we are playing using some regret-minimizing algorithm which outputs action choices $a_t \in A$ at each time period. Then our reward over T time periods is $R(T) = \sum_{t=1}^{T} r_{a_t}(t)$.

Definition 2 *Our regret is defined to be $R_{\max}(T) - R(T)$, where $R_{\max}(T) = \max_{a \in A} \sum_{t=1}^{T} r_a(t)$. If our algorithm randomizes over possible action choices, we also define expected regret to be $R_{\max}(T) - E[R(T)]$. The set of actions against which we compare our performance is called the comparison class.*

Both game theorists and online learning researchers have studied this framework (Fudenburg & Levine, 1995) (Freund & Schapire, 1999). We will refer frequently to the EXP3 algorithm (and its variants) explored by Auer et al. (1995). In the original formulation of EXP3, we choose single actions to play, but we do not get to observe the rewards we would have received if we had chosen different actions. The authors show that the performance of EXP3 exhibits a regret bound of $2\sqrt{e-1}\sqrt{TN \ln N}$. Generally speaking, these regret-minimizing algorithms hedge between possible actions by keeping a weight for each action that is updated according to the action's historical performance. The probability of playing an action is then its fraction of the total weights mixed with the uniform distribution. Intuitively, better experts perform better, get assigned higher weight, and are played more often. Sometimes these algorithms are called experts algorithms, since we can think of the actions as being recommended by a set of experts.

It is important to note that most of these existing methods only compare our performance against strategies that are best responses to what are often called *oblivious* or *myopic* opponents. That is, the opponent does not learn or react to our actions, and essentially plays a fixed string of actions. Our best response would be to play the single best-response action to the empirical distribution of the opponent's actions. Under most circumstances, however, we might expect an intelligent opponent to change their strategy as they observe our own sequence of plays.

For example, consider the game of repeated Prisoner's Dilemma. If we follow the oblivious opponent assumption, then the best choice of action would always be to "Defect." Given any fixed opponent action, the best response would always be to defect. This approach would thus miss out on the chance to earn higher rewards by cooperating with opponents such as a "Tit-for-Tat" opponent, which cooperates with us as long as we also cooperate. These opponents can be called *reactive* opponents. Mannor and Shimkin (2001) propose a super-game framework for extending the regret-minimization framework to handle such cases. In the super-game framework, we evaluate an expert's performance not based on single periods of play; instead each time we play an action or strategy, we commit to playing it for multiple time steps in order to allow the environment to react to our strategy. de Farias and Meggido (2004) also explore this problem using a different performance metric.

3. Extending the experts framework

Our extensions to the regret-minimization framework follow along the lines of the super-game setup. Instead of choosing actions from A, we choose behavioral strategies from M^τ. M^τ also replaces A as our comparison class, essentially forcing us to compare our performance against more complex and possibly better performing strategies. While executing $\mu^\tau \in M^\tau$ for some number of time periods λ, the agent receives reward at each time step, but does not observe the rewards he would have received had he played any of his other possible strategies. This is reasonable since the opponent may adapt differently as a particular strategy is played, causing a different cumulative outcome over λ time periods. Thus, the opponent could be an arbitrary black-box opponent or perhaps a fixed finite automaton. While the inner workings of the opponent are unobservable, we will assume the agent is able to observe the action that the opponent actually plays at each time period.

For example, we might consider an opponent whose action choices only depend on the previous τ-length history of joint actions. Thus, we can construct a Markov model of our opponent using the set of all possible τ-length histories as the state space. If our optimal policy is ergodic, we can use the mixing time of the policy as our choice of λ, since this would give us a good idea of the average rewards possible with this policy in the long run. We will usually assume that we are given λ.

Definition 3 *Let M be a Markov decision process that models the environment (the opponent), and let π be a policy in M such that the asymptotic average reward $V_M^\pi = \lim_{T\to\infty} V_M^\pi(i,T)$ for all i, where $V_M^\pi(i,T')$ is the average undiscounted reward of M under policy π starting at state i from time 1 to T'. The ϵ-commitment time λ_π of π is the smallest T such that for all $T' \geq T$, $|V_M^\pi(i,T') - V_M^\pi| \leq \epsilon$ for all i.*

Thus, if we are executing a policy π learned on a particular opponent model M, then we must run the policy for at least λ time periods to properly estimate the benefit of using that policy. For example, in the Prisoner's Dilemma game shown in Figure 1, assuming a Tit-for-Tat opponent, we might fix $\lambda = 10$, since the average reward of playing "always cooperate" for n time periods is always within $\frac{2}{n}$ of the long-run reward. After playing this policy for 10 periods, we know that we will gain an average reward within 1/5 of the long-term average reward of 1. This is due to the fact that in the first time period, the opponent may still be playing "defect", giving us a reward of -1 for that time period. We will then receive reward of 1 in each ensuing period.

Given a fixed commitment length λ, we may like to be able to evaluate all possible strategies in order to choose the optimal strategy. This would entail enumerating all possible behavioral strategies over λ periods. Since the hedging algorithm will essentially randomize between strategies for us, we only need to consider deterministic behavioral strategies. However, there are still $|A|^{|A|^{2\lambda}}$ possible strategies to evaluate. Not only would this take a long time to try each possible strategy, but the regret bounds also become exceedingly weak. The expected regret after T time periods is:

$$2\sqrt{e-1}|A|^{|A|^{2\lambda}/2}|A|^{2\lambda}\sqrt{T\lambda \ln |A|},$$

Clearly this amounts to a computationally infeasible approach to this problem. In traditional MDP solution techniques, we are saved by the Markov property of the state space, which reduces the number of strategies we need to evaluate by allowing us to reuse information learned at each state. Without any assumptions about the opponent's behavior, as in the classic regret minimization framework, we cannot get such benefits.

4. Learning Algorithms as Experts

However, we might imagine that not all policies are useful or fruitful ones to explore, given a fixed commitment length of λ. In fact, in most cases, we probably have some rough idea about the types of policies that may be appropriate for a given domain. For example, in our Prisoner's Dilemma example, we might expect that our opponent is either a Tit-for-Tat player, an Always-Defect or Always-Cooperate player, or a "Usually Cooperate but Defect with probability p player", for example.

Given particular opponent assumptions, such as possible behavioral models, we may then be able to use a learning algorithm to estimate the model parameters based on observed history. For example, if we believe that the opponent may be Markov in the τ-length history of joint actions, we can construct a Markov model of the opponent and use an efficient learning algorithm (such as E3 from Kearns and Singh (1998)) to learn the ϵ-optimal policy in time polynomial to the number of states, $|A|^{2\tau}$. In contrast, the hedging algorithm needs to evaluate each of the exponentially large number of possible policies, namely $|A|^{|A|^{2\tau}}$ possible policies. To make this precise, we state the following lemma.

Proposition 4 *Given a model of the opponent that is Markov in the τ-length history of joint actions $\{a^i_{t-\tau}, a^{-i}_{t-\tau}, \ldots, a^i_{t-1}, a^{-i}_{t-1}\}$, and given a fixed mixing time λ, the number of actions executed by E3 and a hedging algorithm such as EXP3 in order to arrive at an ϵ-optimal policy is at most $O\left(|A|^{10\tau}\right)$ for E3, and at least $O\left(|A|^{|A|^{2\tau}}\right)$ for the hedging algorithm.*

Of course, using this method, we can no longer guarantee regret minimization over all possible policies, but as we will discuss in the following section, we can choose a subset of fixed policies against which we can compare the performance of any learning algorithms we decide to use, and we can guarantee no-regret relative to this subset of fixed policies, as well as relative to the the learning algorithms.

In some ways, using learning algorithms as experts simply off-loads the exploration from the experts framework to each individual learning algorithm. The computational savings occurs because each learning algorithm makes particular assumptions about the structure of the world and of the opponent, thus enabling each expert to learn more efficiently than hedging between all possible strategies.

4.1. Example

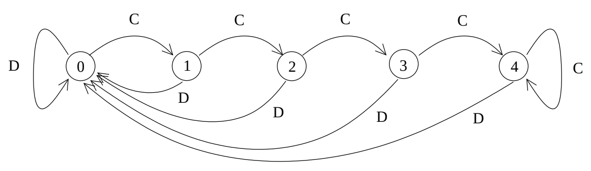

Figure 2. A possible opponent model with five states. Each state corresponds to the number of consecutive "Cooperate" actions we have just played.

For example, consider again the repeated Prisoner's Dilemma game. We might believe that the opponent reacts to our past level of cooperation, cooperating only when we have cooperated a consecutive number of times. If the opponent cooperates only when we have cooperated four periods in a row, then the opponent model shown in Figure 2 would correctly capture the opponent's state dynamics. This model is simpler than a full model using all possible 4-period histories, since it assumes that the opponent's state is completely determined by our half of the joint history. In the figure, the labeled transitions correspond to our actions, and the opponent only cooperates when it is in state 4; otherwise it defects.

To learn the optimal policy with respect this opponent model, a learning algorithm would simply have to visit all the state-action pairs and estimate the resulting reward for each possible action at each state. Since we assume that the opponent model is Markov, we can use an efficient learning algorithm such as E3.

Note that using this particular model, we can also learn the optimal policy for an opponent that cooperates if we cooperate for some given n consecutive periods, where $n \leq 4$. However, if $n \geq 5$, learning using this model will no longer result in the optimal policy. Whereas choosing the cooperate action from state 4 results in a good reward when $n \leq 4$, when $n \geq 5$ the same action results in a bad reward since the opponent will most likely play defect. The problem is that the 5-state model is no longer sufficient to capture the opponent's state dynamics, and is no longer Markov.

5. The Hedged Learner

Since our chosen learning algorithms will sometimes fail to output good policies, we propose to incorporate them as experts inside a hedging algorithm that hedges between a set of experts that includes our learners. This allows the hedging algorithm to switch to using the other experts if a particular learning algorithm fails. It might fail due to incorrect opponent assumptions, such as in the previous section's example, or the learning algorithm may simply be ill-suited for the particular domain, or it may fail for any other reason. The point is that we have a backup plan, and the hedging algorithm will eventually switch to using these other options.

We study two methods for adding learning experts into a regret-minimization algorithm such as Auer et al.'s EXP3. It is straightforward to extend our results to other variants of EXP3 such as EXP3.P.1, which guarantees similar bounds that hold uniformly over time and with probability one.

We are given N fixed experts, to which we must add M learning experts. We assume that $\lambda_i = 1$ for all $i \in N$ and refer to these experts as *static experts*. These static experts are essentially the pure action strategies of the game. For all $i \in M$, we assume $\lambda_i > 1$ and note that M can also include behavioral strategies. When it is clear from context, we will often write N and M as the number of experts in the sets N and M, respectively.

- **Naive approach:** Let $\lambda_{max} = \max_i \lambda_i$. Once an expert is chosen to be followed, follow that expert for a λ_{max}-length commitment phase. At the end of each phase, scale the accumulated reward by $\frac{1}{\lambda_{max}}$ since EXP3 requires rewards to fall in the interval [0,1] and update the weights as in EXP3.
- **Hierarchical hedging:** Let E_0 denote the top-level hedging algorithm. Construct a second-level hedging algorithm E_1 composed of all the original N static strategies. Use E_1 and the learning algorithms as the $M+1$ experts that E_0 hedges between.

5.1. Naive approach

The Naive approach may seem like an obvious first method to try. However, we will show that it is distinctly inferior to hierarchical hedging.

Theorem 5 *Suppose we have a set N of static experts, and a set M of learning experts with time horizons λ_i. Using a naive approach, we can construct an algorithm with regret bound*

$$2\sqrt{e-1}\sqrt{\lambda_{\max}T(N+M)\ln(N+M)}.$$

Proof. We run EXP3 with the $M+N$ experts, with a modification such that every expert, when chosen, is followed for a commitment phase of length $\lambda_{\max}$ before we choose a new expert. We consider each phase as one time period in the original EXP3 algorithm, and note that the accumulated rewards for an expert over a given phase falls in the interval $[0, \lambda_{\max}]$. Thus, the regret bound over $\frac{T}{\lambda_{\max}}$ phases is $2\lambda_{\max}\sqrt{e-1}\sqrt{\frac{T}{\lambda_{\max}}(N+M)\ln(N+M)}$, and the result follows immediately. $\square$

5.2. Hierarchical hedging

The Naive Approach suffers from two main drawbacks, both stemming from the same issue. Because the Naive Approach follows all experts for $\lambda_{\max}$ periods, it follows the static experts for longer than necessary. Intuitively, this slows down the algorithm's adaptation rate. Furthermore, we also lose out on much of the safety benefit that comes from hedging between the pure actions. Whereas a hedging algorithm over the set of pure actions is able to guarantee that we attain at least the safety (minimax) value of the game, this is no longer true with the Naive approach since we have not included all possible $\lambda_{\max}$-length behavioral experts. Thus, each expert available to us may incur high loss when it is run for $\lambda_{\max}$ periods. Hierarchical Hedging addresses these issues.

Theorem 6 *Suppose we have a set N of static experts, and a set M of learning experts with time horizons λ_i, $\max_i \lambda_i > |N|$. We can devise an algorithm with regret bound:*

$$\begin{array}{rl} & 2\sqrt{e-1}\sqrt{TN\ln N} \\ + & 2\sqrt{e-1}\sqrt{\lambda_{\max}T(M+1)\ln(M+1)} \end{array}.$$

This upper bound on the expected regret improves upon the Naive Approach bound as long as

$$\lambda_{\max} \geq \frac{\sqrt{\ln N}}{\sqrt{\ln(M+N)} - \sqrt{\ln(M+1)}}.$$

In practice, we will often use only one or two learning algorithms as experts, so M is small. For $M = 1$, the bound would thus look like:

$$2.63\sqrt{TN \ln N} + 3.10\sqrt{\lambda_{\max} T}.$$

However, we note that these are simply upper bounds on regret. In Section 5.3, we will compare actual performance of these two methods in a some test domains.

Proof. Using the bounds shown to be achieved by EXP3, our top-level hedging algorithm E_0 achieves performance

$$R_{E_0} \geq \max_{i \in M+\{E_1\}} R_i - 2\sqrt{e-1}\sqrt{T(M+1)\ln(M+1)}.$$

Now consider each of the $|M|+1$ experts. The $|M|$ learning experts do not suffer additional regret since they are not running another copy of EXP3. The expert E_1 is running a hedging algorithm over $|N|$ static experts, and thus achieves performance bounded by

$$R_{E_1} \geq \max_{j \in N} R_j - 2\sqrt{e-1}\sqrt{\lambda_{\max} TN \ln N}.$$

Combining this with the above, we see that

$$\begin{aligned} R_{E_0} \quad \geq \quad & \max_{i \in M+N} R_i \\ & -2\sqrt{e-1}\sqrt{TN \ln N} \\ & -2\sqrt{e-1}\sqrt{\lambda_{\max} T(M+1)\ln(M+1)}. \quad \square \end{aligned}$$

Proposition 7 *The Hierarchical Hedging algorithm will attain at least close to the safety value of the single-shot game.*

Proof. From an argument similar to Freund and Schapire (1999), we know that the second-level expert E_1 will attain at least the safety value (or minimax) value of the single-shot game. Since the performance of the overall algorithm E_0 is bounded close to the performance of any of the experts, including E_1, the Hierarchical Hedger E_0 must also attain close to the safety value of the game. □

As desired, hierarchical hedging is an improvement over the naive approach since: (1) it no longer needs to play every expert for $\lambda_{\max}$-length commitment phases and thus should adapt faster, and (2) it preserves the original comparison class by avoiding modifications to the original experts, allowing us to achieve at least the safety value of the game.

Remark. It is also possible to speed up the adaptation of these hedged learners by playing each expert i for only λ_i time periods, weighting the cumulative rewards received during this phase by $1/\lambda_i$, and using this average reward to update the weights. Applied to the hierarchical hedger, we would play each learning algorithm i for λ_i-length phases and the second-level hedging algorithm E_1 for N-length phases. In practice, this often results is some performance gains.

Table 1. Comparison of the performance of the different methods for structuring the hedged learner.

	Regret Bound	Actual Expected Regret	Actual Performance
Naive	125,801	34,761	-96,154
Hierarchical	36,609	29,661	-8,996

5.3. Practical comparisons

We can verify the practical benefit of hierarchical hedging with a simple example. We consider the repeated game of Matching Pennies, shown in Figure 1. Assume that the opponent is playing a hedging algorithm that hedges between playing "Heads" and "Tails" every time period. This is close to a worst-case scenario since the opponent will be adapting to us very quickly.

We run each method for 200,000 time periods. The Hierarchical Hedger consists of 9 single-period experts grouped inside E_1 and one 500-period expert. The Naive Hedger runs all the experts for 500 periods each. The results are given in Table 1, along with the expected regret upper bounds we derived in the previous section. As expected, the hierarchical hedger achieves much better actual performance in terms of cumulative reward over time, and also achieves a lower expected regret. However, the regret for the naive approach is surprisingly low given that its performance is so poor. This is due to a difference in the comparison classes that the methods use. In the naive approach, our performance is compared to experts that choose to play a single action for 500 time periods, rather than for a single time period. Any single action, played for a long enough interval against an adaptive opponent, is a poor choice in the game of matching pennies. The opponent simply has to adapt and play its best response to our action, which we are then stuck with for the rest of the interval. Thus the expected rewards for any of the experts in the naive approach's comparison class is rather poor. For example, the expected reward for the "Heads" expert is -98,582. This explains why our expected regret is small, even though we have such high cumulative losses; we are comparing our performance against a set of poor strategies!

6. Experimental Results

Since the worst-case bounds we derived in the previous section may actually be quite loose, we now present some experimental results using this approach of hedged learning. We consider the repeated Pris-

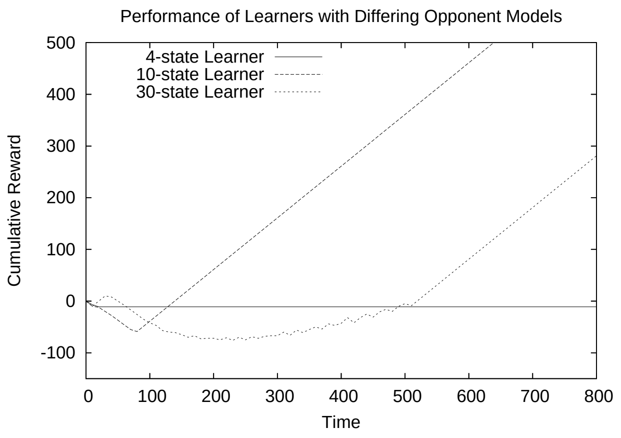

Figure 3. This graph shows the performance of learning algorithms against a Tit-for-Ten-Tats opponent. As the opponent model grows in size, it takes longer for the learning algorithm to decide on an optimal policy.

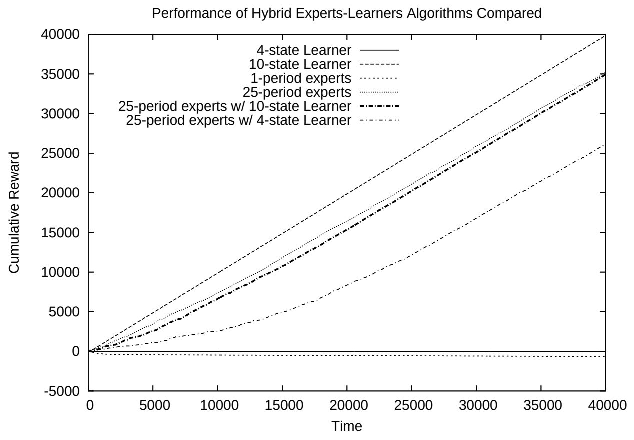

Figure 4. This chart shows the performance of different learning, hedging, and hedging learning algorithms in a game of repeated prisoner's dilemma against a Tit-for-Ten-Tats opponent.

oner's Dilemma game, and we first assume that the unknown opponent is a "Tit-for-Ten-Tats" opponent. That is, the opponent will only cooperate once we have cooperated for ten time periods in a row.

We use a variety of different opponent models with simple learning algorithms, pure hedging algorithms that only hedge between static experts, and hedged learning algorithms that combine learning algorithms with static experts. First, we note that larger opponent models are able to capture a larger number of potential opponent state dynamics, but require both a longer commitment phase λ and a larger number of iterations before a learning algorithm can estimate the model parameters and solve for the optimal policy. For example, Figure 3 shows the performance of three different n-state learners, with $n = 4, 10, 30$. As discussed earlier in Section 4, the 4-state learner is unable to capture the opponent's state dynamics and thus learns an "optimal" policy of defecting at every state. This results in an average reward of zero per time step. On the other hand, the 10-state and 30-state learners lose some rewards while they are exploring and learning the parameters of their opponent models, but then gain an average reward of 1 after they have found the optimal policy of always cooperating.

Figure 4 shows the performance of various learning and hedging algorithms. The "1-period experts" hedging algorithm hedges between single periods of cooperating and defecting. This myopic algorithm is unable to learn the cooperative outcome and thus ends up achieving the single-shot Nash equilibrium value of 0. It assigns a very high weight to the Defect expert. On the other hand, the "25-period experts" hedging algorithm switches between two experts which either cooperate or defect for all possible 25-period histories. This algorithm realizes that the "always cooperate" expert attains higher reward and thus eventually plays Cooperate with probability approaching 1. The hedged 10-state learner is also able to achieve the cooperative outcome. It achieves cumulative reward only slightly lower than the unhedged 10-state learner, since it quickly realizes that the "always cooperate" policy and the learned optimal policy both return higher rewards than the "always defect" policy.

One main benefit of the hedged learning approach becomes evident when we observe the performance of the hedged 4-state learner. Even though the 4-state model is unable to capture the state dynamics and the learning algorithm thus fails to learn the cooperative policy, the hedged 4-state learner is able to achieve average rewards of 1 as it assigns larger and larger weight to the "always cooperate" expert and learns to ignore the recommendations of the failed learning expert. We have wisely hedged our bets between the available experts and avoided placing all our bets on the learning algorithm.

Another major benefit of using hedged learners occurs when the environment is non-stationary. For example, assume that the opponent switches between playing Tit-for-Ten-Tats ("Cooperate for 10 Cooperates") and "Cooperate for 10 Defects" every 15,000 time periods. While the unhedged learner becomes confused with each switch, the hedged learner is able to adapt as the opponent changes and gains higher cumulative rewards

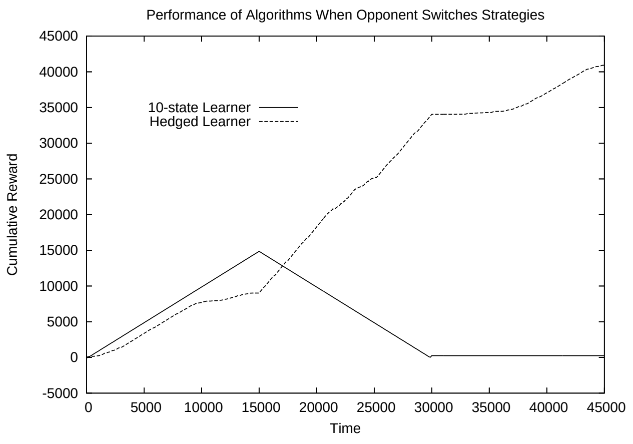

Figure 5. In these trials, the opponent switches strategy every 15,000 time periods. It switches between playing Tit-for-Ten-Tats ("Cooperate for 10 Cooperates") and "Cooperate for 10 Defects". While the modeler becomes confused with each switch, the hedging learner is able to adapt as the opponent changes and gain higher cumulative rewards.

(Figure 5). Note that when the opponent does its first switch, the unhedged learner continues to use its cooperative policy, which was optimal in the first 15,000 periods but now returns negative average reward. In contrast, the hedged learner is able to quickly adapt to the new environment and play a primarily defecting string of actions. Figure 6 shows how the hedging algorithm is able to change the probabilities with which it plays each expert as the environment changes, i.e. when the opponent switches strategies.

7. Conclusion

Hedged learning is able to incorporate various possible opponent models as well as various possible fixed strategies into its hedging framework. It is thus able to benefit from the efficiencies of using learning algorithms to learn optimal policies over these models, while ensuring that it has the option of falling back on the fixed strategies in case the learning algorithms fail to output any good policies. Even when the opponent switches strategies during play, hedged learning is able to adapt to its changing environment and switch to using the expert best adapted to the new opponent strategy. To make this approach concrete, we showed two possible methods for combining learning algorithms with a fixed set of static experts, and provided regret bounds in each case. Finally, we note that all of our results still hold for > 2 players, since we only require observations of our own reward and the history of actions. Furthermore, we can also extend

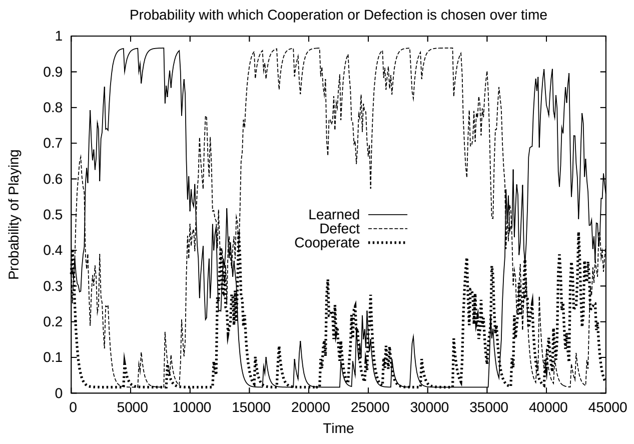

Figure 6. Graph showing the probability with which the weighted hedger plays either a cooperating strategy or a defecting strategy against the switching opponent over time.

these techniques for use in stochastic games.

References

Auer, P., Cesa-Bianchi, N., Freund, Y., & Schapire, R. E. (1995). Gambling in a rigged casino: the adversarial multi-armed bandit problem. *Proceedings of the 36th Symposium on Foundations of Computer Science.*

Chang, Y., & Kaelbling, L. P. (2001). Playing is believing: The role of beliefs in multi-agent learning. *NIPS.*

de Farias, D. P., & Meggido, N. (2004). How to combine expert (or novice) advice when actions impact the environment. *Proceedings of NIPS.*

Freund, Y., & Schapire, R. E. (1999). Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, *29*, 79–103.

Fudenburg, D., & Levine, D. K. (1995). Consistency and cautious fictitious play. *Journal of Economic Dynamics and Control*, *19*, 1065–1089.

Hu, J., & Wellman, M. P. (1998). Multiagent reinforcement learning: Theoretical framework and an algorithm. *Proceedings of the 15th ICML.*

Kearns, M., & Singh, S. (1998). Near-optimal reinforcement learning in polynomial time. *ICML.*

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. *ICML.*

Mannor, S., & Shimkin, N. (2001). Adaptive strategies and regret minimization in arbitrarily varying Markov environments. *Proc. of 14th COLT.*

Nachbar, J., & Zame, W. (1996). Non-computable strategies and discounted repeated games. *Economic Theory.*