Safe Strategies for Agent Modelling in Games

Peter McCracken and **Michael Bowling**
Department of Computing Science
University of Alberta
{peterm,bowling}@cs.ualberta.ca

Abstract

Research in opponent modelling has shown success, but a fundamental question has been overlooked: what happens when a modeller is faced with an opponent that cannot be successfully modelled? Many opponent modellers could do arbitrarily poorly against such an opponent. In this paper, we aim to augment opponent modelling techniques with a method that enables models to be used safely. We introduce ϵ-safe strategies, which bound by ϵ the possible loss versus a safe value. We also introduce the Safe Policy Selection algorithm (SPS) as a method to vary ϵ in a controlled fashion. We prove in the limit that an agent using SPS is guaranteed to attain at least a safety value in the cases when the opponent modelling is ineffective. We also show empirical evidence that SPS does not adversely affect agents that are capable of modelling the opponent. Tests with a domain of complicated modellers show that SPS is effective at eliminating losses while retaining wins in a variety of modelling algorithms.

Introduction

A common and important component of decision-making in a multiagent system is agent modelling[1]. The goal of agent modelling is to represent and extract knowledge of the behavior of other agents in the environment. This problem has been explored under a number of guises including plan recognition (Kautz & Allen 1986; Bauer 1995), behavior classification (Han & Veloso 1999), and opponent modelling (Riley & Veloso 2002). The idea of representing knowledge of other agents' behavior is also present in research in the field of game theory (e.g., fictitious play (Brown 1949; Robinson 1951), and finite automaton learning (Carmel & Markovitch 1996)). A possibly overlooked problem in this line of research is how a constructed model can then be used in the decision-making process. The common approach is to optimize the agent's behavior with respect to these constructed models, *i.e.*, playing a best-response to the modelled behavior of the other agents (Riley & Veloso 2002; Robinson 1951; Carmel & Markovitch 1996).

[1]We will use "opponent" and "agent" interchangeably without implying anything about the other agents' goals. Our experimental results, though, do focus on a competitive situation.

One of the common unstated assumptions in this work is that the other agents can be modelled by the modelling agent. Although this seems to be a critical assumption to even beginning the endeavour, it means much of the research has ignored an important possibility. What if one is interacting with agents that cannot be modelled, or (more likely) are just more complex than the agent's space of models? For example, an agent that is itself modelling may be too complex to be modelled. When an agent interacts with these difficult agents the resulting failure can be catastrophic. Consider fictitious play, which learns a model of other agents assuming their behavior is stationary. An agent with even a very simple dynamic behavior can result in the fictitious play agent doing arbitrarily poorly.

This paper seeks to address this problem by introducing a *safe* opponent modelling algorithm. This safe algorithm provides a lower bound guarantee on the resulting performance even when interacting with agents that cannot be modelled. Our approach does not change how a model is constructed, but rather focuses on how a model is used to affect decision-making. We introduce the concept of ϵ-safe strategies to bound the agent's worst-case performance. We then show how ϵ can be varied to gain both the advantages of safe strategies and the advantages of an accurate agent model. This approach complements the existing agent modelling work, by focusing on how an opponent model is used. It assumes that an agent model is available, constructed using existing agent modelling techniques. We use a very generic form for an agent model to facilitate the algorithm's use across the variety of these techniques. The safe opponent modelling approach, then, has broad application across the entire field of agent modelling.

The paper will be organized as follows. First, we will outline the required background material for normal form games, opponent modelling, safe strategies, and no-regret algorithms. Next we describe the Safe Policy Selection algorithm and prove its safety. We also show experimental results in an advanced opponent modelling domain that indicate there is little loss when an agent is capable of modelling the opponent. Finally, we conclude with a discussion of issues related to SPS.

Definitions and Background

Throughout this paper, we discuss issues related to policy selection in the context of normal form games. Normal form games can be considered an abstraction of an arbitrary multiagent system; their use here is meant to retain the generality of our algorithm. An efficient implementation of our approach would exploit the known structure of the game, which does not necessarily have to be in normal form. A normal form game (Owen 1995; Gintis 2000) is defined by the number of agents, the reward function, and the actions available to each agent. To play a normal form game, each agent i simultaneously choses an action a_i from their action set A_i. The combination of each of the agents' action choices is a joint action a from the set $A = A_1 \times \cdots \times A_n$. When discussing actions, $a_{\text{-}i} \in A_{\text{-}i}$ is the set of action choices made by all agents except for agent i. Each agent also has a reward function $R_i : A \rightarrow \mathbb{R}$. A two-player normal form game is often given in the form of a payoff matrix, as in Table 2.

A policy (or strategy), π_i, is a multinomial probability distribution over actions. The space of all possible policies for agent i is Π_i. When an agent plays according to a given policy, the agent samples from the policy and plays the action indicated by the sample value. A deterministic policy assigns all of the probability mass to a single action. The value of a policy, given the actions of the other players, is

$$V(\pi_i, a_{\text{-}i}) = \sum_{a_i \in A_i} \pi_i(a_i) R_i(a_i, a_{\text{-}i})$$

The value of a policy, given the policies of the other players, is

$$V(\pi_i, \pi_{\text{-}i}) = \sum_{a_i \in A_i} \sum_{a_{\text{-}i} \in A_{\text{-}i}} \pi_i(a_i) \pi_{\text{-}i}(a_{\text{-}i}) R_i(a_i, a_{\text{-}i})$$

The function $V(\pi_i)$, where $V : \Pi \rightarrow \mathbb{R}$, will be used to abbreviate $V(\pi_i, \pi_{\text{-}i})$. The goal of each agent is to play according to the policy that maximizes its reward.

In a repeated normal form game, the agents play the same game repeatedly, and maintain a history of the results of past games. The action sets and the reward functions remain constant for all timesteps, but the agents are able to vary their strategies over time. We are interested in the algorithms that chose strategies for the agents. In this section, we explore two example algorithms: fictitious play, which attempts to exploit opponents, is typical of many opponent modellers, and minimax, which guarantees a safety value but often accepts lower payoffs than necessary. We also look at no-regret algorithms, which are a class of algorithms that are able to guarantee an outcome at least as good, on average, as the best of a set of strategies provided to it.

Fictitious Play

Opponent modelling algorithms are learning algorithms that attempt to build an explicit model of the opponent, and then choose actions according to that model. A traditional game-theoretic method of opponent modelling is fictitious play (Fudenberg & Levine 1999), which was originally developed as an iterative method for "solving" a game. Fictitious play records the frequencies of the opponents' joint actions. It then uses that model, along with the payoffs in the normal form game, to compute the best-response – the action with the highest expected payoff. That action is then deterministically selected.

According to Bowling & Veloso in (2002), a best-response learner is one such that if the opponents' policy converges to a static policy, the learner will eventually converge to the best-response to that policy. Under this definition, fictitious play is a best-response learner. It will eventually build an accurate model of any static opponents' strategies, and will act to maximize its reward given that opponent model.

The trouble with fictitious play, and opponent modelling in general, is that modelling an arbitrary opponent is not feasible; there are too many possible types of agents in order for any known generic system to be able to model them all. As an example, consider the infinite recursion of models that would be required for an opponent modeller to create a perfect model of another perfect opponent modeller. Thus, opponent modellers must make assumptions about their opponents. A frequent assumption, used by algorithms like fictitious play, is that the other agents are Markov, *i.e.*, that the other agents have a static distribution over actions for all timesteps. Clearly this assumption is not always valid, particularly against learning algorithms that adapt their strategies over time.

Because of the assumptions made by opponent modellers, they can be exploited by opponents that do not meet the modeller's expectations. If an opponent can predict how the modeller will react to a given history, it can know how to play to maximally exploit the modeller. For instance, if an opponent knows that the agent uses fictitious play, it could compute what that agent's "best-response" is, and then act to maximize its reward given the knowledge of the agent's impending action.

Although fictitious play is a simple example of an opponent modeller, most of the problems with it remain in more complex modellers. Mixed strategies and complex meta-reasoning can lessen some of the problems, but not eliminate them completely. By deviating from a safe policy, opponent modellers open themselves up for exploitation.

Minimax Strategies

Game theoretic optimal solutions in normal form games maximize reward in the worst case (von Neumann & Morgenstern 1944). A strategy that maximizes a reward assuming that the opponent will try to minimize the reward is known as a minimax strategy. The minimax algorithm is designed for adversarial games, in which agents necessarily attempt to minimize their opponents' payoffs in order to maximize their own. Unlike opponent modelling, minimax strategies are intended to be as unexploitable as possible. Regardless of the actions of the opponent, a minimax policy π^* is able to guarantee a minimum level of reward, r^*. Any non-optimal choices on the part of the opponent only increase the payoff to the minimax player. Because of this guaranteed reward, minimax strategies can be viewed as a *safe* strategy.

A significant advantage of the minimax strategy over other algorithms is that it is independent of the policy played by the opponent. This means that a minimax solution can be calculated ahead of time for any game, and this strategy can be put into effect regardless of the actions of the opponent. Unlike learning algorithms, such as opponent modellers, there is no initial period of low effectiveness while the model is being built.

Like the opponent modelling strategies, it is the assumptions made by minimax agents that are their main weakness. Minimax strategies assume that the opponents are optimal, and that the goals of the opponents are opposite the goals of the agent. In cases where these assumptions are not true, minimax players can end up settling for much lower payoffs than are allowed by the game and the opponent. This means that minimax players do not follow a best-response policy, unless the opponent is optimal and adversarial.

No-Regret Algorithms

No-regret algorithms are learning algorithms that guarantee, in the limit, that the average obtained reward is no worse than that obtained by playing any static strategy (Jafari *et al.* 2001). Hedge, described in (Auer *et al.* 1995), is a simple no-regret algorithm that uses probabilistic strategies to balance exploration and exploitation of actions. Actions that have provided better payoffs in the past are played with higher frequency. Auer *et al.* show that, with proper parameter selection, Hedge is expected to obtain a payoff no worse than $\sqrt{2T \ln K}$ less than the best payoff from any fixed action, where T is the number of games and K is the number of actions.

While the definition of no-regret algorithms concerns no-regret versus fixed strategies, no-regret algorithms can generally be used to guarantee no-regret versus given dynamic strategies, as well. For instance, in Hedge, the base actions could be replaced by experts that recommend policies. Hedge could probabilistically choose between the experts, preferring those that have given higher rewards in the past. Used in this way, Hedge could select between a safe policy and an opponent modeller, and guarantee zero regret versus both of them. However, as we will see later, the algorithm presented in this paper has some advantages over no-regret algorithms when used to guarantee safety.

ϵ-Safe Strategies

We present here ϵ-safe strategies, which have a controlled level of safety, and the Safe Policy Selection algorithm (SPS), which defines how to control the safety level. An agent that uses SPS is guaranteed in the limit to have an average payoff of at least r^*, the expected value of playing π^*. Unlike normal safe strategies, though, an agent using ϵ-safe strategies and SPS can also take advantage of its underlying opponent modeller to earn rewards greater than r^*. In this section we will first describe ϵ-safe strategies, then present Safe Policy Selection, and conclude with a theoretical proof showing the safety of the algorithm.

A strategy π is considered ϵ-safe if it is guaranteed to have an expected payoff no lower than ϵ less than r^*, the safety value of the game; *i.e.*,

$$V(\pi, a_{\text{-}i}) \geq r^* - \epsilon, \, \forall a_{\text{-}i} \in A_{\text{-}i}$$

In general, there can be multiple strategies for a game that are ϵ-safe for any given value of ϵ. This set of strategies we denote $\mathrm{SAFE}(\epsilon)$. The $\mathrm{SAFE}()$ function implicitly depends on the chosen value of r^*, which we assume in this paper to be the minimax value of the game. Two special cases include $\mathrm{SAFE}(0)$, which is the set of safe strategies that guarantee a payoff of r^*, and $\mathrm{SAFE}(\beta)$, which contains all possible strategies. The special value β is the difference between r^* and the lowest possible payoff in the game. The benefit of an ϵ-safe strategy over a safe strategy is that, although it can possibly lose by a greater margin, it can also win by a greater margin. The greater number of strategies afforded by relaxing the safety constraint allows more freedom for the agent to choose a strategy that will earn a larger reward.

In order to choose the best strategy from $\mathrm{SAFE}(\epsilon)$, an opponent modeller is used. In this context, an opponent modeller can be considered an evaluation function, $M : \Pi \rightarrow \mathbb{R}$, that returns an estimate of the expected value of a policy when played against the modelled opponent. An agent that uses an ϵ-safe strategy will play according to the policy π_ϵ where:

$$\pi_\epsilon \doteq \operatorname*{argmax}_{\pi \in \mathrm{SAFE}(\epsilon)} M(\pi)$$

It should be noted that $M(\pi)$ is not necessarily the same as $V(\pi)$. If the opponent modeller does not have a correct model of the opponent, it is possible that $M(\pi_\epsilon) > r^*$ and $V(\pi_\epsilon) < r^*$, and the agent would be better off playing π^* than π_ϵ. These are the cases in which ϵ-safe strategies are useful, because the value of ϵ limits the amount which π_ϵ can lose. The policy

$$\pi_{\max} \doteq \operatorname*{argmax}_{\pi \in \Pi} M(\pi)$$

is the policy that M considers to be the best-response to the opponent. Note that $\pi_{\max}$ could be π^*, if the opponent modeller believes that the opponent is optimal. In cases where $V(\pi_{\max}) < r^*$, ϵ-safe strategies will prevent the agent from getting arbitrarily bad rewards.

If the value of ϵ remains fixed, an ϵ-safe strategy suffers the same disadvantages as the safety policy, because it is not necessarily a best-response to the true opponent. It also suffers the disadvantages of an opponent modeller, because it can be exploited since it deviates from the safety strategy. Because of this, it is necessary to change the value of ϵ in response to how well the agent is actually performing.

Safe Policy Selection in Normal Form Games

Table 1 shows the Safe Policy Selection algorithm for using ϵ-safe strategies in a normal form game. It is assumed that the agent knows A and R_i, and also that the agent can observe the actions of the opponent.

The given update policy for ϵ in Step 4 is:

$$\epsilon^{(t+1)} \leftarrow \epsilon^{(t)} + f(t+1) + V(\pi^{(t)}, a_{\text{-}i}^{(t)}) - r^*$$

The Safe Policy Selection Algorithm

Parameters: The matrix game, G, and decay function, f

$r^* \leftarrow$ the safety value of G
Initialize opponent model, M
$\epsilon^{(1)} \leftarrow f(1)$
At each timestep, t, from 1 to T:

(1) $\pi^{(t)} \leftarrow \underset{\pi \in \mathrm{SAFE}(\epsilon^{(t)})}{\operatorname{argmax}} M(\pi)$

(2) play action $a_i^{(t)}$ according to $\pi^{(t)}$

(3) update M with opponents' actions, $a_{\text{-}i}^{(t)}$

(4) $\epsilon^{(t+1)} \leftarrow \epsilon^{(t)} + f(t+1) + V(\pi^{(t)}, a_{\text{-}i}^{(t)}) - r^*$

Table 1: The Safe Policy Selection Algorithm for Player i

Since the reward function is known, the value of $V(\pi^{(t)}, a_{\text{-}i}^{(t)})$ can easily be calculated. The function $f : \mathbb{N} \rightarrow \mathbb{R}$ is the decay function for updating ϵ. It is an arbitrary function that has the properties:

$$f(t) > 0 \quad (1)$$

$$\lim_{T\rightarrow\infty} \frac{\sum_{t=1}^{T} f(t)}{T} = 0 \quad (2)$$

We choose $f(t) = \frac{\beta}{t}$. Informal experiments show that this choice works well, and that functions that decay at a slower rate are more exploitable in the short run.

Theoretical Analysis

In this section, we show that, in the limit, following SPS will guarantee an average reward at least as great as r^*.

Theorem 1 *As $T \rightarrow \infty$, the worst case average reward of following the Safe Policy Selection algorithm will be at least that of the safety policy.*

Proof: The worst case opponents always choose their actions $a_{\text{-}i}^{(t)}$ to maximize the loss of the agent. Because the loss is bounded by $\epsilon^{(t)}$, the worst case opponents ensure

$$V(\pi^{(t)}, a_{\text{-}i}^{(t)}) = r^* - \epsilon^{(t)}$$

The total difference between the expected payoff for the played policy and the safety policy is

$$\begin{aligned}
& \sum_{t=1}^{T} r^* - \sum_{t=1}^{T} V(\pi^{(t)}, a_{\text{-}i}^{(t)}) \\
= & \sum_{t=1}^{T} r^* - \left(\sum_{t=1}^{T} \left(r^* - \epsilon^{(t)} \right) \right) = \sum_{t=1}^{T} \epsilon^{(t)} \\
= & \sum_{t=1}^{T} \left(\epsilon^{(t-1)} + f(t) + V(\pi^{(t-1)}, a_{\text{-}i}^{(t-1)}) - r^* \right) \\
= & \sum_{t=1}^{T} f(t) + \sum_{t=1}^{T} \left(V(\pi^{(t-1)}, a_{\text{-}i}^{(t-1)}) - (r^* - \epsilon^{(t-1)}) \right) \\
= & \sum_{t=1}^{T} f(t)
\end{aligned}$$

Thus, in the short run, the exploitability of the agent is bounded by the sum $\sum_{t=1}^{T} f(t)$. In the limit, as $T \rightarrow \infty$, the time average expected difference over all T timesteps is zero, due to Condition 2 on $f(t)$. Therefore, in the limit, an agent using SPS will have an average reward at least as good as the safety policy. □

	R	P	S
R	0	-1	1
P	1	0	-1
S	-1	1	0

Table 2: The payoff matrix for Player One in Rock-Paper-Scissors.

Experimental Analysis

The previous section showed only that SPS is guaranteed to be safe. In practise, however, SPS also has the property that it does not significantly impact the ability of agents to properly model the opponent. To test the claims made in the previous section, we experiment with ϵ-safe strategies and SPS in the game of Rock-Paper-Scissors. Although Rock-Paper-Scissors itself is simple, it provides a complex environment for opponent modelling, and opponent modellers are available that have many degrees of subtlety. These experiments show that ϵ-safe strategies are able to improve on all but the best opponent modellers, without a detrimental effect on the modellers that could not be improved.

Rock-Paper-Scissors

Rock-Paper-Scissors (RPS) is a simple two-player, zero-sum matrix game, with the payoff matrix defined in Table 2. The optimal game theoretic strategy in RPS is to play all actions with equal probability. This is a Nash equilibrium and has an expected value of 0, regardless of what the opponent plays. This value will be used as r^* in our experiments with ϵ-safe strategies.

Because RPS is a simple game, it is an effective domain for testing opponent modellers. Players can focus on modelling without the distraction of complicated game rules or stochastic events. Agents can use pattern detection, statistical analysis, or other complicated methods in order to model their opponents, while attempting to not be modelled by any opponent. See (Billings 2000) for more information on the types of opponent modelling used in the RPS domain. A simple giveaway that the "optimal" solution is not sufficent in tournament settings is that, as seen in Table 3, the random player placed 20th out of 25 competitors.

The opponent modellers used for these experiments came from the first International RoShamBo Programming Competition, described in (Billings 2000). The source code for the competition and players is freely available. These players make an effective test bed for ϵ-safe strategies and SPS, because of their complexity, diversity, and their previously established success rates.

Experimental Setup

The RoShamBo competitors consisted of 24 opponent modellers, and a minimax player that chose actions randomly. The experiments were run as round robin tournaments, in which each competitor played 1,000 games against every other competitor. Fifty tournaments were run for each experiment, and the results given are the average total reward over the tournaments.

To determine the effect of ϵ-safe strategies and SPS, a set of tournaments was run for each competitor, in which the competitor was used as an opponent modeller in the SPS algorithm. All other opponents were left the same. For a baseline ranking of the opponent modellers, one set of tournaments was run with all of the competitors unmodified. The effect of SPS is evaluated by how well the modified opponent modellers fared when compared to their unmodified counterparts. We hope to show that all of the modellers will perform as well or better than their counterparts.

At each timestep, the opponent modellers recommend an action, a. In an unmodified player, a will be the action taken. In an ϵ-safe player, an evaluation function over strategies is created such that $\pi_{\max}$ is considered to be the policy that chooses a with probability 1, and other policies are ranked by their probability of choosing a. The played policy, $\pi^{(t)}$, is calculated by solving a linear system with constraints ensuring that the loss is no greater than $\epsilon^{(t)}$, and maximizing the probabilty of a.

Results

Table 3 shows an overview of the results of augmenting the RoShamBo players with SPS. To analyze these results, we will look at the effect of SPS on total score, tournament rank, and match scores.

The 'Score' columns in Table 3 show the sum of the rewards obtained in the 24 matches against the other players, averaged across 50 tournaments. The 'Sig.' column indicates whether the change in score between the unmodified player and the ϵ-safe player is statistically significant with 99% confidence. Of the 24 players, 15 experienced a significant increase in score. Four players had a decreased overall score, but none of these reductions were significant. It should be noted that all of the ϵ-safe players obtained better scores than r^*, even though five of the original players had negative scores. As expected, the change in scores was most significant for weaker players, because these were the players that were most frequently unable to model their opponents.

The 'Rank' columns in Table 3 show the ranking of each player in the original tournament, and the rank of the ϵ-safe version of that opponent in the modified tournament. The ranks were calculated by sorting the average scores for each player over the 50 tournaments. The values give an indication of how well each RoShamBo player would have done if it had used SPS in the tournament. In total, 15 out of the 24 players increased in rank when SPS was used. One player decreased in rank, although the actual difference in scores was not significant. Like the scores, the change in rank was most significant for the weaker opponents.

Player	Rank		Score		Sig.
	O	ϵ	O	ϵ	
Iocaine Powder	1	1	4123	4116	
Phasenbott	2	2	3533	3584	
Simple Modeller	3	3	2786	2868	
MegaHAL	4	3	2751	2838	
Biopic	5	5	2346	2408	✓
Boom	6	6	1852	1903	
ACT-R Lag2	7	7	1790	1784	
Robertot	8	6	1779	1920	✓
Shofar	9	9	1643	1617	
Bugbrain	10	9	1591	1736	✓
RussRocker4	11	4	1583	2731	✓
Simple Predictor	12	5	1399	2296	✓
Sweet Rocky	13	12	893	1386	✓
Piedra	14	13	835	1315	✓
Marble	15	6	808	2090	✓
Granite	16	6	763	2094	✓
Vroomfondel	17	18	713	696	
Majikthise	18	18	699	720	
Mixed Strategy	19	13	591	1142	✓
Random (Optimal)	20	-	-36	-	
ZQ Bot	21	6	-260	1742	✓
Multi-strategy	22	13	-3159	903	✓
Inocencio	23	12	-3546	1177	✓
Knucklehead	24	20	-5810	154	✓
Peterbot	25	15	-19674	101	✓

Table 3: Overall results for the RPS experiments. 'Rank' is the placement in the tournament, based on the average score, and 'Score' is the total reward against all opponents, averaged across the 50 tournaments. The columns labelled 'O' are from the original competition, and the 'ϵ' columns show the results when the player uses SPS. 'Sig.' shows whether the difference in scores is statistically significant with 99% confidence.

In tournament settings like the RoShamBo competition, the effect of SPS on rank is two-fold. The first effect, which we have seen, is that the score of the ϵ-safe player rises since it no longer loses against other players. The second effect is that the players that previously were able to win against the ϵ-safe player have their scores reduced by a corresponding amount. Thus, the score of the ϵ-safe player is relatively closer to the scores of the other players, which can lead to increases in rank. A prime example of this effect is Peterbot and Knucklehead, whose ranks rose to 15 and 20 when using SPS, respectively. However, Peterbot's score of 101 was less than Knucklehead's score of 154. The reason that Peterbot's rank increased by five more spots than Kunckle-head's is because the scores of the other players decreased. Most of the average players were able to win almost all of their games against the original Peterbot, which made a significant contribution to their score. Thus, when these players were no longer able to win against Peterbot, their scores dropped enough that Peterbot was able to surpass them.

When the original player obtains a positive reward in individual matches, we expect that the SPS-augmented player

Player: Iocaine Powder			
Opponent	Score		Sig.
	O	ϵ	
Phasenbott	111	95	
MegaHAL	54	55	
Simple Modeller	33	30	
Inocencio	296	320	
Knucklehead	562	559	
Peterbot	988	987	

(a) Matches with Iocaine Powder

Player: RussRocker4			
Opponent	Score		Sig.
	O	ϵ	
Iocaine Powder	-519	-14	✓
Phasenbott	-530	-6	✓
MegaHAL	-94	-5	✓
Inocencio	189	222	
Knucklehead	477	478	
Peterbot	992	992	

(b) Matches with RussRocker4

Table 4: The results of individual matches, with against and without using ϵ-safe strategies. In the 'O' columns, the player was unmodified, and in the 'ϵ' column, the player used ϵ-safe strategies. The opponents were all unmodified.

will obtain the same reward. When the original player's reward was negative, we expect that the SPS-augmented player's reward will increase to $r^* - \sum_{t=1}^{T} f(t)$, which is about -7.5 for the RPS experiments. The actual results are close to the expected results; on average, a penalty of about -4.5 was incurred in the cases when the reward was already positive [2]. In the cases when the reward was originally negative, the ϵ-safe score was increased to -3.7.

In order to show more detail in how SPS affects match results, Table 4 gives the results of individual matches. Only selected results are shown, but the rest of the results follow the general trend shown in these two tables.

Table 4(a) shows the match results for Iocaine Powder, the strongest player, against the three other strongest players, and against the three weakest players. Without SPS, Iocaine Powder was able to model all six of the opponents, although it only had slight leads against the stronger opponents. Thus, using ϵ-safe strategies would not be expected to help Iocaine Powder. The important thing to note is that SPS did not significantly interfere with the ability to model the opponents.

Table 4(b) shows the match results for RussRocker4, against the three strongest and three weakest opponents. Unmodified, RussRocker4 was defeated by all three of the stronger opponents, but was able to win against the three weaker opponents. The ϵ-safe version of RussRocker4 maintained its lead against the weak opponents. Against the strong opponents, the large negative values were all increased to small negative values, obtaining a reward only slightly less than the safety value of 0.

In almost every case, when SPS is used, the ability of players to model their opponents is unaffected. However, there were two anomalous cases when using SPS substantially interferred with the ability of an agent to model an opponent. In both cases, the opponent was Peterbot. ACT-R Lag2 and Knucklehead were able to earn an average reward of 397 and 803, respectively, against Peterbot, but when using SPS they earned average rewards of 196 and 279. In general, anomalies like this are possible when arbitrary opponent modellers are used, since even slightly different behaviours could trigger different styles of play in opponents. It is also possible for anomalies like these to work in favor of the ϵ-safe players, as well. For instance, when Peterbot used SPS against Inocencio, its score changed from -794 to 228, even though ϵ-safe strategies should not be expected to increase a negative score to a large positive one.

Results with a No-Regret Algorithm

As described previously, no-regret algorithms perform the same basic function as SPS; they ensure that the long run results are no worse than those of a safety strategy. As a test of how SPS performs in practise against existing algorithms, we repeated the RoShamBo experiments using Hedge (Auer *et al.* 1995) as an example no-regret algorithm.

Hedge can be used to choose between multiple strategies by treating each strategy as a basic action in the algorithm. Hedge then decides with what probability each strategy is followed, and guarantees in the limit that the regret of the played actions versus each strategy is zero. This is stronger than the guarantees made by SPS, because SPS has no theoretical guarantee that it will perform as well as the opponent modeller. In our experiments, the value used for Hedge's η parameter was the value recommended in (Auer *et al.* 1995); $\eta = \ln(1 + \sqrt{2 \ln K/T})$, where $K = 2$ (the actions were 'follow the random strategy' or 'follow the opponent modeller') and $T = 1000$ (the number of games). At each timestep, a probability p was calculated, using the Hedge algorithm, for following the action recommended by the opponent modeller. Thus, players augmented with Hedge play policies that consist of linear combinations of the safety strategy and the recommended action.

Table 5 shows a subset of the results for the Hedge experiments. Like SPS, using Hedge significantly improves the scores of the weak and middle players. Overall, 12 players had a significant increase in score, and 14 of the players increased in rank. However, Hedge caused significant reductions in the scores of five of the stronger players, and 6 players decreased in rank. Thus, it seems that Hedge is an effective way to increase the score of weak and mediocre players; but is dangerous to use with the strongest players. SPS did not adversely affect the scores of the strongest players, despite the lack of proof for SPS's best response properties.

[2] When the anomalous cases discussed later are removed from this calculation, the average penalty decreases to -2.5.

Player	Rank		Score		Sig.
	O	H	O	H	
Iocaine Powder	1	1	4123	3932	✓
Phasenbott	2	2	3533	3399	✓
Simple Modeller	3	4	2786	2529	✓
MegaHAL	4	4	2751	2632	✓
Biopic	5	5	2346	2373	
Boom	6	7	1852	1800	
ACT-R Lag2	7	11	1790	1373	✓
Robertot	8	6	1779	2030	✓
Shofar	9	11	1643	1552	
Bugbrain	10	9	1591	1715	✓
RussRocker4	11	5	1583	2424	✓
Simple Predictor	12	6	1399	2069	✓
Sweet Rocky	13	14	893	905	
Piedra	14	13	835	833	
Marble	15	12	808	1443	✓
Granite	16	12	763	1479	✓
Vroomfondel	17	18	713	678	
Majikthise	18	17	699	755	
Mixed Strategy	19	14	591	926	✓
Random (Optimal)	20	-	-36	-	-
ZQ Bot	21	13	-260	1127	✓
Multi-strategy	22	14	-3159	743	✓
Inocencio	23	12	-3546	1148	✓
Knucklehead	24	22	-5810	-550	✓
Peterbot	25	22	-19674	-1309	✓

Table 5: A selection of overall results for the RPS experiments, using Hedge as the safety guarantee algorithm.

Hedge is not as effective as SPS for mitigating loss. Even in cases when both algorithms improve on the original score, the ϵ-safe score tends to be larger, and SPS does not reduce the scores of the strong players. There were 16 cases where the ϵ-safe score improved significantly over the Hedge score, and only one case in which Hedge significantly improved upon SPS.

Discussion

In this section, we present some issues related to ϵ-safe strategies and Safe Policy Selection. We discuss when they should be used, how they can be feasibly implemented, and how they differ from the class of no-regret algorithms.

Appropriate Use of ϵ-Safe Strategies

We saw in the results from the RoShamBo tournament that using ϵ-safe strategies appropriately is able to prevent weaker opponent modellers from losing by large amounts. The result was that weak players became mediocre players, and mediocre players became stronger players. However, the effect of ϵ-safe strategies on the strongest players was negligible. The SPS algorithm is not a magic solution to turn a bad player into the best player, since its success is limited by the underlying opponent modeller. Therefore, there seems to be a conflict: in order to win, the opponent modeller must already work well, and if the opponent modeller already works well, SPS is not needed. This begs the question, why use SPS at all?

Using ϵ-safe strategies could be thought of as a type of insurance for opponent modellers. While a small penalty could be paid in cases when the opponent modeller is successful, large losses are prevented in the cases when the opponent modeller is incapable of modelling the other players. For a perfect opponent modeller, of course doing anything but following the opponent model will be detrimental, and SPS cannot improve on it. In general, though, the type and skill of opponents will not be known ahead of time, and therefore it will not be known how well the agent will fare. Also, as opponents become more sophisticated, the probabilty that there can be one universal modeller becomes small. Even for extremely good modellers, the ability to retreat in an intelligent way to a safe strategy will become more essential.

Making SPS Tractable

The SPS algorithm assumes that the best strategy, π_ϵ, can be chosen from the set $\mathrm{SAFE}(\epsilon)$. In general, iterating over $\mathrm{SAFE}(\epsilon)$ to find π_ϵ is impossible since $|\mathrm{SAFE}(\epsilon)|$ is infinite. Several solutions are possible for this problem, such as function optimization or approximation of $\mathrm{SAFE}(\epsilon)$. Function optimization is the approach taken in the RPS domain. Instead of being enumerated explicitly, $\mathrm{SAFE}(\epsilon)$ is defined by a set of constraints, and the opponent modeller provides an optimization objective. If the constraints are linear, as with RPS, then π_ϵ can be found with linear programming.

Another method of making SPS tractable is to use an approximation $\mathrm{X}(\epsilon)$ of $\mathrm{SAFE}(\epsilon)$. $\mathrm{X}(\epsilon)$ could be a discretized version of $\mathrm{SAFE}(\epsilon)$, or some other abstraction that can be optimized over more efficiently. If an approximation is used, it should satisfy the properties ($\forall\epsilon,\epsilon'$):

$$\begin{aligned} \mathrm{X}(\epsilon) &\neq \emptyset \\ \mathrm{X}(\epsilon) &\subseteq \mathrm{SAFE}(\epsilon) \\ \epsilon < \epsilon' &\Rightarrow \mathrm{X}(\epsilon) \subseteq \mathrm{X}(\epsilon') \\ \pi_{\max} &\in \mathrm{X}(\beta) \end{aligned}$$

Note that this implies any approximation of $\mathrm{SAFE}(\epsilon)$ should be conservative; if any unsafe strategy is deemed safe by the approximation, then long term safety can no longer be guaranteed. Using an approximation or reduced strategy space is meant to confine computation to areas where maximization can be done efficiently. In general, better approximations of $\mathrm{SAFE}(\epsilon)$ will result in better performance of the algorithm.

SPS and No-Regret Algorithms

We saw in the previous section that Hedge, a no-regret algorithm, did not perform as well as SPS in the RoShamBo tournaments. While the performance of Hedge does not necessarily reflect on the peformance of other no-regret algorithms, there is a common trait of all algorithms in this class. A fundamental difference between the approaches taken by the SPS algorithm and by no-regret algorithms is the space of policies that each considers. No-regret algorithms are only capable of stochastically switching between strategies that are provided to the algorithm. Although these strategies can be dynamic, such as an opponent modeller strategy, the fact that there is only a finite number of strategies limits the flexibility of no-regret algorithms.

The SPS algorithm takes a much more general approach. Instead of switching between a finite set of strategies, it can choose to play any strategy that satisfies the given safety constraint. This can lead to policies that would never be considered by a no-regret player. This is important for two reasons. First, it means the agent has more freedom to choose appropriate policies. Second, it could be seen as a form of exploration, which may enable the opponent modeller to create more accurate models of how the other players react to given styles of play. The drawback of this approach is that it requires more computation, and may require a more sophisticated modeller in order to accurately evaluate arbitrary policies.

Conclusion

In a multi-agent learning environment, the ability to react appropriately to the actions of other agents is essential. Part of reacting appropriately includes knowing when to take advantage of, or compensate for, perceived weaknesses in another agent, and when to retreat to the safety of a known policy. In this paper we have shown theoretically that SPS is capable of ensuring safety for an agent, and practically that it still allows the agent to model an opponent and exploit weaknesses. In the Rock-Paper-Scissors domain, SPS was applied to a large collection of real opponent modellers, and significantly improved the performance of most of them, while not significantly harming any of them.

Our main direction of future research with ϵ-safe strategies is to extend their use to more complicated games. Rock-Paper-Scissors worked very well as a domain for testing their effect in a pure opponent modelling enviroment. However, a more practical demonstration of their effectiveness would need to involve a larger game. In particular, we would like to apply ϵ-safe strategies to stochastic games, which have multiple states. Preliminary work with a stochastic domain shows that the translation of the algorithm to account for multiple states is non-trivial. Also, the theoretical guarantee of safety no longer exists when actions have side effects other than immediate reward, such as state transitions.

Acknowledgements

We would like to thank Dale Schuurmans for help in working out some of the theory for safe opponent modelling, and the anonymous reviewers for finding errors and providing some suggestions to polish the paper. We would also like to acknowledge NSERC and iCORE, along with the Alberta Ingenuity Fund through its funding of the Alberta Ingenuity Centre for Machine Learning, for their support of this research.

References

Auer, P.; Cesa-Bianchi, N.; Freund, Y.; and Schapire, R. E. 1995. Gambling in a rigged casino: The adversarial multi-arm bandit problem. In *36th Annual Symposium on Foundations of Computer Science*, 322–331. Milwaukee, WI: IEEE Computer Society Press.

Bauer, M., ed. 1995. *IJCAI 95 Workshop on The Next Generation of Plan Recognition Systems: Challenges for and Insight from Related Areas of AI (Working Notes)*.

Billings, D. 2000. The first international roshambo programming competition. *International Computer Games Association Journal* 23(1):3–8,42–50.

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

Brown, G. W. 1949. Some notes on computation of games solutions. RAND Report P-78, The RAND Corporation, Santa Monica, California.

Carmel, D., and Markovitch, S. 1996. Learning models of intelligent agents. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*. Menlo Park, CA: AAAI Press.

Fudenberg, D., and Levine, D. K. 1999. *The Theory of Learning in Games*. The MIT Press.

Gintis, H. 2000. *Game Theory Evolving*. Princeton University Press.

Han, K., and Veloso, M. 1999. Automated robot behavior recognition applied to robotic soccer. In *Proceedings of the 9th International Symposium of Robotics Research (ISSR'99)*, 199–204.

Jafari, A.; Greenwald, A.; Gondek, D.; and Ercal, G. 2001. On no-regret learning, fictitious play, and Nash equilibrium. In *Proceedings of the Eighteenth International Conference on Machine Learning*, 226–223.

Kautz, H., and Allen, J. 1986. Generalized plan recognition. In *Proceedings of the 5th National Conference on Artificial Intelligence (AAA1-86)*, 32–37.

Owen, G. 1995. *Game Theory*. Academic Press.

Riley, P., and Veloso, M. 2002. Planning for distributed execution through use of probabilistic opponent models. In *Proceedings of the Sixth International Conference on AI Planning and Scheduling*, 77–82.

Robinson, J. 1951. An iterative method of solving a game. *Annals of Mathematics* 54:296–301.

von Neumann, J., and Morgenstern, O. 1944. *Theory of Games and Economic Behavior*. John Wiley and Sons.