Accident or Intention: That Is the Question (in the Noisy Iterated Prisoner's Dilemma)

Tsz-Chiu Au
Department of Computer Science
University of Maryland
College Park, MD 20742
chiu@cs.umd.edu

Dana Nau
Department of Computer Science
University of Maryland
College Park, MD 20742
nau@cs.umd.edu

ABSTRACT

This paper focuses on the Noisy Iterated Prisoner's Dilemma, a version of the Iterated Prisoner's Dilemma (IPD) in which there is a nonzero probability that a "cooperate" action will accidentally be changed into a "defect" action and vice versa. Tit-For-Tat and other strategies that do quite well in the ordinary (non-noisy) IPD can do quite badly in the Noisy IPD.

This paper presents a technique called *symbolic noise detection*, for detecting whether anomalies in player's behavior are deliberate or accidental. The key idea is to construct a model of the other agent's behavior, and watch for any deviation from this model. If the other agent's next action is inconsistent with this model, the inconsistency can be due either to noise or to a genuine change in their behavior; and we can often distinguish between two cases by waiting to see whether this inconsistency persists in next few moves.

We entered several different versions of our strategy in the 2005 Iterated Prisoner's Dilemma competition, in Category 2 (noisy environments). Out of the 165 contestants in this category, our programs consistently ranked among top ten. The best of our programs ranked third, and it was beaten only by two "master-slave strategy" programs that each had a large number of "slave" programs feeding points to them.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*; I.6.8 [**Simulation and Modeling**]: Types of Simulation—*Gaming*

General Terms

Design, Experimentation, Measurement, Theory

Keywords

Adaptation; Cooperation; Coordination; Game theory; Learning; Multi-agent systems; Noise; Opponent modeling; Prisoner's Dilemma; Repeated game; Tit-for-tat

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

1. INTRODUCTION

The Iterated Prisoner's Dilemma (IPD) has become well known as an abstract model of a class of multi-agent environments in which agents accumulate payoffs that depend on how successful they are in their repeated interactions with other agents. An important variant of the IPD is the *Noisy* IPD, in which there is a small probability, called the *noise level*, that accidents will occur. In other words, the noise level is the probability of executing "cooperate" when "defect" was the intended move, or vice versa.

Accidents can cause difficulty in cooperations with others in real-life situations, and the same is true in the Noisy IPD. Strategies that do quite well in the ordinary (non-noisy) IPD may do quite badly in the Noisy IPD [4, 6, 7, 18, 19, 20]. For example, if two players both use the well-known Tit-For-Tat (TFT) strategy, then an accidental defection may cause a long series of defections by both players as each of them punishes the other for defecting.

In this paper, we describe a technique called *symbolic noise detection* for dealing with noise. The key idea is to build a symbolic model of how the other agent behaves, and watch for any deviation from this model. If the other agent's next move is inconsistent with their past behavior, this inconsistency can be due either to noise or to a genuine change in their behavior; and we can often distinguish between these two cases by waiting to see if this inconsistency persists or not in the next few iterations of the game.

To test our technique, we wrote several programs that used it, and entered them as contestants in Category 2 (the Noisy IPD) of the 2005 Iterated Prisoner's Dilemma competition [14]. In the competition, the noise level was 10%. The result of the competition showed an impressive performance for symbolic noise detection (Table 1). Seven of our nine symbolic noise detection programs placed among top ten according to their overall average score. Our best program, DBSz, placed third, and it was the best-performing non-master-slave program in Category 2.[1]

[1]Each participant in the competition was allowed to submit up to 20 programs as contestants. Some participants took advantage of this to submit collections of programs that worked together in a conspiracy in which 19 of their 20 programs (the "slaves") worked to give as many points as possible to the 20th program (the "master"). In contrast, DBS is not a master-slave program. Instead, DBS builds behavioral models of the other contestants in order to coop-

This paper describes the design of DBS, particularly the implementation of the symbolic noise detection in DBS. Section 2 presents the motivation for the approach, Section 3 gives the basic definitions, and Section 4 gives an overview of how DBS works. Section 5 how the opponent modeling is done and in particular the symbolic noise detection works. Section 6 discusses DBS's performance in the 2005 Iterated Prisoner's Dilemma competition, and contrasts DBS to the notorious master-and-slaves strategy. Section 7 discusses related work, and Section 8 is the conclusion.

2. MOTIVATION AND APPROACH

The techniques used in DBS are motivated by a British army officer's story that was quoted in [3, page 40]:

> I was having tea with A Company when we heard a lot of shouting and went out to investigate. We found our men and the Germans standing on their respective parapets. Suddenly a salvo arrived but did no damage. Naturally both sides got down and our men started swearing at the Germans, when all at once a brave German got onto his parapet and shouted out: "We are very sorry about that; we hope no one was hurt. It is not our fault. It is that damned Prussian artillery." (Rutter 1934, 29)

Such an apology was an effective way of resolving the conflict and preventing a retaliation because it told the British that the salvo was not the intention of the German infantry, but instead was an unfortunate accident that the German infantry did not expect nor desire. The reason why the apology was convincing was because it was consistent with the German infantry's past behavior. The British had was ample evidence to believe that the German infantry wanted to keep the peace just as much as the British infantry did.

More generally, an important question for conflict prevention in noisy environments is *whether a misconduct is intentional or accidental.* A deviation from the usual course of action in a noisy environment can be explained in either way. If we form the wrong belief about which explanation is correct, our response may potentially destroy our long-term relationship with the other player. If we ground our belief on evidence accumulated before and after the incident, we should be in a better position to identify the true cause and prescribe an appropriate solution. To accomplish this, DBS uses the following key techniques:

1. **Learning about the other player's strategy.** DBS uses an induction technique to identify a set of rules that model the other player's recent behavior. The rules give the probability that the player will cooperate under different situations. As DBS learns these probabilities during the game, it identifies a set of *deterministic* rules that have either 0 or 1 as the probability of cooperation.

2. **Detecting noise**. DBS uses the above rules to detect anomalies that may be due either to noise or a genuine change in the other player's behavior. If a move is different from what the deterministic rules predict, this inconsistency triggers an *evidence collection process* that will monitor the persistence of the inconsistency in the next few iterations of the game. The purpose of the evidence-collection process is to determine whether the violation is likely to be due to noise or to a change in the other player's policy.

3. **Temporarily tolerating possible misbehaviors by the other player.** Until the evidence-collection process finishes, DBS assumes that the other player's behavior is still as described by the deterministic rules. Once the evidence collection process has finished, DBS decides whether to believe the other player's behavior has changed, and updates the deterministic rules accordingly.

Since DBS emphasizes the use of deterministic behaviors to distinguish noise from the change of the other player's behavior, it works well when the other player uses a pure (i.e., deterministic) strategy or a strategy that makes decisions deterministically most of the time. Fortunately, deterministic behaviors are abundant in the Iterated Prisoner's Dilemma. Many well-known strategies, such as TFT and GRIM, are pure strategies. Some strategies such as Pavlov or Win-Stay, Lose-Shift strategy (WSLS) [15, 16, 17, 21] are not pure strategies, but a large part of their behavior is still deterministic. The reason for the prevalence of determinism is discussed by Axelrod in [2]: clarity of behavior is an important ingredient of long-term cooperation. A strategy such as TFT benefits from its clarity of behavior, because it allows other players to make credible predictions of TFT's responses to their actions. We believe the success of our strategy in the competition is because this clarity of behavior also helps us to fend off noise.

The results of the competition show that the techniques used in DBS are indeed an effective way to fend off noise and maintain cooperation in noisy environments. When DBS defers judgment about whether the other player's behavior has changed, the potential cost is that DBS may not be able to respond to a genuine change of the other player's behavior as quickly as possible, thus losing a few points by not retaliating immediately. But this delay is only temporary, and after it DBS will adapt to the new behavior. What is more important is that the techniques used in DBS greatly reduce the probability that noise will cause it to end a cooperation and fall into a mutual-defect situation. Our experience has been that it is hard to re-establish cooperation from a mutual-defection situation, so it is better avoid getting into mutual defection situations in the first place. When compared with the potential cost of ending an cooperation, the cost of temporarily tolerating some defections is worthwhile.

Temporary tolerance also benefits us in another way. In the noisy Iterated Prisoner's Dilemma, there are two types of noise: one that affects the other player's move, and the other affects our move. While our method effectively handles the first type of noise, it is the other player's job to deal with the second type of noise. Some players such as TFT are easily provoked by the second type of noise and retaliate immediately. Fortunately, if the retaliation is not a permanent one, our method will treat the retaliation in the same way as the first type of noise, thus minimizing its effect.

3. DEFINITIONS

3.1 Iterated Prisoner's Dilemma with Noise

In the Iterated Prisoner's Dilemma, two players play a finite sequence of classical prisoner's dilemma games, whose

erate with them as effectively as possible.

payoff matrix is:

		Player 2	
		Cooperate	Defect
Player 1	Cooperate	(u_{CC}, u_{CC})	(u_{CD}, u_{DC})
	Defect	(u_{DC}, u_{CD})	(u_{DD}, u_{DD})

where $u_{DC} > u_{CC} > u_{DD} > u_{CD}$ and $2u_{CC} > u_{DC} + u_{CD}$. In the competition, u_{DC}, u_{CC}, u_{DD} and u_{CD} are 5, 3, 1 and 0, respectively.

At the beginning of the game, each player knows nothing about the other player and does not know how many iterations it will play. In each iteration, each player chooses either to cooperate (C) or defect (D), and their payoffs in that iteration are as shown in the payoff matrix. We call this decision a *move*, hence $\{C, D\}$ is the set of possible moves. After both players choose a move, they will each be informed of the other player's move before the next iteration begins.

If $a_k, b_k \in \{C, D\}$ are the moves of Player 1 and Player 2 in iteration k, then we say that (a_k, b_k) the *outcome* of iteration k. If there are N iterations in a game, then the total scores for Player 1 and Player 2 are $\sum_{1 \le k \le N} u_{a_k b_k}$ and $\sum_{1 \le k \le N} u_{b_k a_k}$, respectively.

The *Noisy Iterated Prisoner's Dilemma* is a variant of the Iterated Prisoner's Dilemma in which there is a small probability called the *noise level*[2] that a player's moves will be mis-implemented. The noise level is the probability of executing C when D was the intended move, or vice versa. The incorrect move is recorded as the player's move, and determines the outcome of the iteration.[3] Furthermore, neither player has any way of knowing whether the other player's move was executed correctly or incorrectly.[4]

3.2 Strategies and Policies

A *history* H of length k is the sequence of outcomes of all iterations up to and including iteration k. We write $H = \langle (a_1, b_1), (a_2, b_2), \ldots, (a_k, b_k) \rangle$. We let $\mathcal{H} = \{(C, C), (C, D), (D, C), (D, D)\}^*$ be the set of all possible histories. A *strategy* $M : \mathcal{H} \to [0, 1]$ associates with each history H a number $M(H) \in [0, 1]$ called the *degree of cooperation*. $M(H)$ is the probability that M chooses to cooperate at iteration $k + 1$, where k is H's length.

As an example, suppose the Player 2 is TFT. We can model this as a function M_{TFT} such that $M_{TFT}(H) = 1$ if H's length is zero or $a_k = C$, and $M_{TFT}(H) = 0$ otherwise.

A *condition* $Cond : \mathcal{H} \to \{\mathsf{True}, \mathsf{False}\}$ is a mapping from histories to boolean values. A history H *satisfies* a condition $Cond$ if $Cond(H) = \mathsf{True}$. A *policy schema* is a set of conditions such that each history in $\mathcal{H}$ satisfies exactly one of the conditions. A *rule* is a pair $(Cond, p)$ which we will write as $Cond \to p$, where $Cond$ is a condition and p is a degree of cooperation. A *policy* is a set of rules whose conditions constitute a policy schema. We will use a policy to approximate the behavior exhibited by a strategy.

[2]The noise level in the competition was 0.1.

[3]Hence, a mis-implementation is different from a misperception, which would not change the outcome of the iteration. The competition included mis-implementations but no misperceptions.

[4]As far as we know, the definitions of "mis-implementation" used in the existing literature are ambiguous about whether either of the players should know that an action has been mis-executed.

```
Procedure DerivedBeliefStrategy()
1.  R_d ← R_TFT      // the default rule set
2.  R_c ← ∅          // the current rule set
3.  a_0 ← C ; b_0 ← C ; H ← ⟨(a_0, b_0)⟩; π = R_d
4.  a_1 ← MoveGen(π, H) ; k ← 1 ; v ← 0
5.  Loop until the end of the game
6.     Output a_k and obtain the other player's move b_k
7.     r+ ← ((a_{k-1}, b_{k-1}) → b_k)
8.     r− ← ((a_{k-1}, b_{k-1}) → ({C, D} \ {b_k}))
9.     If r+, r− ∉ R_c
10.       If ShouldPromote(r+) = true, insert r+ into R_c.
11.    If r+ ∈ R_c, set the violation count of r+ to zero
12.    If r− ∈ R_c and ShouldDemote(r−) = true
13.       R_d ← R_c ∪ R_d; R_c ← ∅; v ← 0
14.    If r− ∈ R_d, then v ← v + 1
15.    If v > RejectThreshold, or (r+ ∈ R_c and r− ∈ R_d)
16.       R_d ← ∅; v ← 0
17.    R_p ← {(C → p') ∈ ψ_{k+1} : C not appear in R_c or R_d}
18.    π ← R_c ∪ R_d ∪ R_p   // R_p is the probabilistic rule set
19.    H ← ⟨H, (a_k, b_k)⟩; a_{k+1} ← MoveGen(π, H) ; k ← k + 1
```

Figure 1: An outline of the DBS strategy. ShouldPromote first increases r^+'s promotion count, and then if r^+'s promotion count exceeds the promotion threshold, ShouldPromote returns true and resets r^+'s promotion count. Likewise, ShouldDemote first increases r^-'s violation count, and then if r^-'s violation count exceeds the violation threshold, ShouldPromote returns true and resets r^-'s violation count. ψ_{k+1} in Line 17 is calculated from Equation 1.

We can model M_{TFT} as a policy as follows: we write $Cond = (a, b)$ as a condition about the outcomes of the last iteration of a history, such that $Cond(H) = \mathsf{True}$ if and only if $a_k = a$ and $b_k = b$. For simplicity, if the length of H is zero, then $Cond(H) = \mathsf{True}$ if and only if $a = C$ and $b = C$. The policy for M_{TFT} is $R_{TFT} = \{(C, C) \to 1.0, (C, D) \to 1.0, (D, C) \to 0.0, (D, D) \to 0.0\}$.

4. DERIVED BELIEF STRATEGY

Figure 1 outlines the DBS strategy. DBS maintains a *hypothesized policy* to model the other player's strategy. DBS initially assumes that the other player's policy is TFT, but at each iteration of the game, DBS updates the hypothesized policy according to how the player has actually played. Section 5 describes how the updating works.

DBS uses the current hypothesized policy to decide what move to make. For this, it uses the MoveGen procedure on line 4 of the figure. MoveGen is bascially a game-tree search procedure for maximizing DBS's utility, assuming that the hypothesized policy π does not change. We omit the details due to lack of space; for details see [1].

An important question is how large a policy schema to use for the hypothesized policy. If the other player's strategy is complicated and the policy schema is too small, the policy schema won't provide enough detail to give useful predictions of the other player's behavior. But if the policy schema is too large, DBS will be unable to compute an accurate approximation of each rule's degree of cooperation, because the number of iterations in the game will be too small. In the competition, we used a policy schema of size 4: $\{(C, C), (C, D), (D, C), (D, D)\}$. We have found this to be good enough for modeling a large number of strategies.

5. LEARNING HYPOTHESIZED POLICIES IN NOISY ENVIRONMENTS

We now describe how DBS learns another player's strategy. Section 5.1 describes how DBS maintains a *discounted frequency* for each behavior: instead of keeping an ordinary frequency count, DBS applies discount factors based on how recent each occurrence of the behavior was. Sections 5.2–5.5 describe why discounted frequencies alone are not sufficient to deal with noise, and how DBS uses symbolic noise detection and temporary tolerance to help overcome the difficulty.

5.1 Learning by Discounted Frequencies

Given a history $H = \{(a_1, b_1), (a_2, b_2), \ldots, (a_k, b_k)\}$, a real number α between 0 and 1 (called the *discount factor*), and an initial hypothesized policy $\pi_0 = \{Cond_1 \rightarrow p_1^0, Cond_2 \rightarrow p_2^0, \ldots, Cond_n \rightarrow p_n^0\}$ whose policy schema is $\mathcal{C} = \{Cond_1, Cond_2, \ldots, Cond_n\}$, the *probabilistic policy* at iteration $k+1$ is $\psi_{k+1} = \{Cond_1 \rightarrow p_1^{k+1}, Cond_2 \rightarrow p_2^{k+1}, Cond_n \rightarrow p_n^{k+1}\}$, where p_i^{k+1} is computed by the following equation:

$$p_i^{k+1} = \frac{\sum_{0 \le j \le k} (\alpha^{k-j} g_j)}{\sum_{0 \le j \le k} (\alpha^{k-j} f_j)} \quad (1)$$

and where

$$g_j = \begin{cases} p_i^0 & \text{if } j = 0, \\ 1 & \text{if } 1 \le j \le k,\ Cond_i(H_{j-1}) = \mathsf{True} \text{ and } b_j = C, \\ 0 & \text{otherwise;} \end{cases}$$

$$f_j = \begin{cases} p_i^0 & \text{if } j = 0, \\ 1 & \text{if } 1 \le j \le k,\ Cond_i(H_{j-1}) = \mathsf{True}, \\ 0 & \text{otherwise;} \end{cases}$$

$$H_{j-1} = \begin{cases} \emptyset & \text{if } j = 1, \\ \{(a_1, b_1), (a_2, b_2), \ldots, (a_{j-1}, b_{j-1})\} & \text{otherwise.} \end{cases}$$

In short, the current history H has $k+1$ possible prefixes, and f_j is bascially a boolean function indicating whether the prefix of H up to the $j-1$'th iteration satisfies $Cond_i$. g_j is a restricted version of f_j.

When $\alpha = 1$, p_i is approximately equal to the frequency of the occurrence of $Cond_i \rightarrow p_i$. When α is less than 1, p_i becomes a weighted sum of the frequencies that gives more weight to recent events than earlier ones. For our purposes, it is important to use $\alpha < 1$, because it may happen that the other player changes its behavior suddenly, and therefore we should forget about its past behavior and adapt to its new behavior (for instance, when GRIM is triggered). In the competition, we used $\alpha = 0.75$.

It is important to have a good initial hypothesized strategy because at the beginning of the game the history is not long enough for us to derive any meaningful information about the other player's strategy. In the competition, the initial hypothesized policy is TFT: $\{(C, C) \rightarrow 1.0, (C, D) \rightarrow 1.0, (D, C) \rightarrow 0.0, (D, D) \rightarrow 0.0\}$.

5.2 The Deficiency of using Discounted Frequency to Deal with Noise

It may appear that the probabilistic policy learnt by the discounted-frequency learning technique should be inherently capable of tolerating noise, because it takes many, if not all, moves in the history into account: if the number of terms in the calculation of the average or weighted average is large enough, the effect of noise should be small. However, there is a problem with this reasoning: it neglects the effect of multiple occurences of noise within a small time interval.

A mis-implementation that alters the move of one player would distort an established pattern of behavior observed by the other player. The general effect of such distortion to the Equation 1 is hard to tell—it varies with the value of the paramters and the history. But if several distortions occur within a small time interval, the distortion may be big enough to alter the probabilistic policy and hence change our decision about what move to make. This change of decision may potentially destroy an established pattern of mutual cooperation between the players.

At first glance, it might seem rare for several noise events to occur at nearly the same time. But if the game is long enough, the probability of it happening can be quite high. The probability of getting two noise events in two consecutive iterations out of a sequence of i iterations can be computed recursively as $X_i = p(p + qX_{i-2}) + qX_{i-1}$, providing that $X_0 = X_1 = 0$, where p is the probability of a noise event and $q = 1 - p$. In the competition, the noise level was $p = 0.1$ and $i = 200$, which gives $X_{200} = 0.84$. Similarly, the probabilities of getting three and four noises in consecutive iterations are 0.16 and 0.018, respectively.

In the 2005 competition, there were 165 players, and each player played each of the other players five times. This means every player played 825 games. On average, there were 693 games having two noises in two consecutive iterations, 132 games having three noises in three consecutive iterations, and 15 games having four noises in four consecutive iterations. Clearly, we did not want to ignore situations in which several noises occur nearly at the same time.

The *symbolic noise detection* and temporary tolerance outlined in Section 2 provide a way to reduce the amount of susceptibility to multiple occurrences of noise in a small time interval. The basic idea is to partition the set of rules in the current hypothesized policy into two sets: the set of *deterministic rules* (whose degrees of cooperation are either zero or one) and the set of *probabilistic rules* (whose degrees of cooperation are between zero and one). The deterministic rules are then used to detect noise in the way that we mentioned in Section 2.

In the following subsections, we first present the induction technique we used to identify the deterministic rules in the other player's behavior. Then we describe the evidence-collection process in the symbolic noise detection. After that, we discuss how to cope with the ignorance of the other player's new behavior when the evidence-collection process decides that there is a change in the other player's behavior.

5.3 Identifying Deterministic Rules Using Induction

As we discussed in Section 2, deterministic behaviors are abundant in the Iterated Prisoner's Dilemma. Deterministic behaviors can be modeled by deterministic rules, whereas random behavior would require probabilistic rules.

Deterministic rules have only two possible degrees of cooperation: zero or one. Therefore, there should be ways to learn deterministic rules that are much faster than the discounted frequency method described earlier. For example, if we knew at the outset which rules were deterministic, it would take only one occurrence to learn each of them: each time the condition of a deterministic rule was satisfied, we could assign a degree of cooperation of 1 or 0 depending on

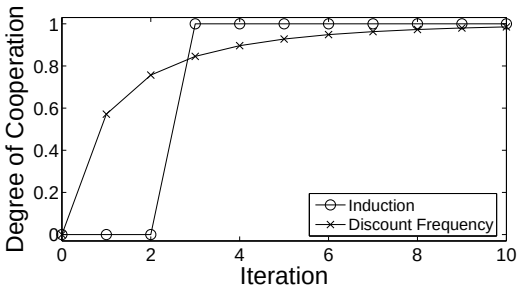

Figure 2: Learning speeds of the induction method and the discounted frequency method when the other player always cooperates. The initial degree of cooperation is zero, the discounted rate is 0.75, and the promotion threshold is 3.

whether the player's move was C or D.

The trick, of course, is to determine which rules are deterministic. We have developed an inductive-reasoning method to distinguish deterministic rules from probabilistic rules during learning and to learn the correct degree of cooperation for the deterministic rules.

In general, induction is the process of deriving general principles from particular facts or instances. To learn deterministic rules, the idea of induction can be used as follows. If a certain kind of behavior occurs repeatedly several times, and during this period of time there is no other behavior that contradicts to this kind of behavior, then we will hypothesize that the chance of the same kind of behavior occurring in the next few iterations is pretty high, regardless of how the other player behaved in the remote past.

More precisely, let $K \geq 1$ be a number which we will call the *promotion threshold*. Let $H = \langle (a_1, b_1), (a_2, b_2), \ldots, (a_k, b_k) \rangle$ be the current history. For each condition $Cond_j \in \mathcal{C}$, let I_j be the set of indexes such that for all $i \in I_j$, $i < k$ and $Cond_j(\langle (a_1, b_1), (a_2, b_2), \ldots, (a_i, b_i) \rangle) = \mathsf{True}$. Let $\hat{I}_j$ be the set of the largest K indexes in I_j. If $|I_j| \geq K$ and for all $i \in \hat{I}_j$, $b_{i+1} = C$ (i.e., the other player chose C when the previous history up to the i'th iteration satisties $Cond_j$), then we will hypothesize that the other player will choose C whenever $Cond_j$ is satisfied; hence we will use $Cond_j \rightarrow 1$ as a deterministic rule. Likewise, if $|I_j| \geq K$ and for all $i \in \hat{I}_j$, $b_{i+1} = D$, we will use $Cond_j \rightarrow 0$ as a deterministic rule. See Line 7 to Line 10 in Figure 1 for an outline of the induction method we use in DBS.

The induction method can be faster at learning deterministic rules than the discounted frequency technique (see Figure 2). A faster learning speed allows us to infer the deterministic rules with a shorter history. The promotion threshold K controls the speed of the identification of deterministic rules. The larger the value of K, the slower the speed of identification, but the less likely we will mistakenly hypothesize that the other player's behavior is deterministic.

5.4 Symbolic Noise Detection and Temporary Tolerance in DBS

Once DBS has identified the set of deterministic rules, it can readily use them to detect noise. As we said earlier, if the other player's move violate a deterministic rule, it can be caused either by noise or by a change in the other player's behavior, and DBS uses an evidence collection process to figure out which is the case. More precisely, once a deterministic rule $Cond_i \rightarrow o_i$ is violated (i.e., the history up to the previous iteration satisfies $Cond_i$ but the other player's move in the current iteration is different from o_i), DBS keeps the violated rule but marks it as violated. Then DBS starts an *evidence collection process* that in the implementation of our competition entries is a violation counting: for each violated probabilistic rule DBS maintains a counter called the *violation count* to record how many violations of the rule have occurred (Line 12). In the subsequent iterations, DBS increases the violation count by one every time a violation of the rule occurs. However, if DBS encounters a positive example of the rule, DBS resets the violation count to zero and unmark the rule (Line 11). If any violation count excesses a threshold called the *violation threshold*, DBS concludes that the violation is not due to noise; it is due to a change of the other player's behavior. In this case, DBS invokes a special procedure (described in Section 5.5) to handle this situation (Line 13).

The drawback of the discount frequency method is that it fails to deal with several misinterpretations caused by noise within a small time interval. The way to avoid this drawback is to set a larger violation threshold. But if the threshold is too large, it will slow down the speed of adaptation to changes in the other player's behavior. In the competition, we entered DBS several times with several different violation thresholds; and in the one that performed the best, the violation threshold was 4.

5.5 Coping with Ignorance of the Other Player's New Behavior

When the evidence collection process detects a change in the other player's behavior, DBS knows little about the other player's new behavior. How DBS copes with this ignorance is critical to its success.

DBS maintains two sets of deterministic rules: the *current rule set* R_c and the *default rule set* R_d. Symbolic noise detection and temporary tolerance makes use of the rules in R_c but not the rules in R_d. However, DBS makes use of the rules in both R_c and R_d when DBS decides the next move (Line 18). More precisely, when DBS constructs a hypothesized policy π for move generation, it uses every rule in R_c and R_d. In addition, for any missing rule (i.e., the rule those condition are different from any rule's condition in R_c or R_d), we regard it as a probabilistic rule and approximate its degree of cooperation by Equation 1 (Line 17). These probabilistic rules form the probabilistic rule set $R_p \subseteq \psi_{k+1}$.

The default rule set is designed to be *rejected*: we maintain a violation count to record the number of violations to any rule in R_d. Every time any rule in R_d is violated, the violation count increased by 1 (Line 14). Once the violation count exceeds a *rejection threshold*, we drop the default rule set entirely (set it to an empty set) and reset the violation count (Line 15 and Line 16). We also reject R_d whenever any rule in R_c contradicts any rule in R_d (Line 15).

At the beginning of a game, R_d is R_{TFT} and R_c is empty (Line 1 and Line 2). While DBS can insert any newly found deterministic rule in R_c, it insert rules into R_d *only when* the evidence collection process detects a change of the other player's behavior. When it happens, DBS copies all the rules in R_c to R_d, and then set R_c to an empty set (Line 13).

Table 1: Scores of the top 20 programs, averaged over the five runs in Competition 2.

Rank	Program	Avg. score
1	BWIN	433.8
2	IMM01	414.1
3	DBSz	408.0
4	DBSy	408.0
5	DBSpl	407.5
6	DBSx	406.6
7	DBSf	402.0
8	DBStft	401.8
9	DBSd	400.9
10	lowESTFT_classic	397.2
11	TFTIm	397.0
12	Mod	396.9
13	TFTIz	395.5
14	TFTIc	393.7
15	DBSe	393.7
16	TTFT	393.4
17	TFTIa	393.3
18	TFTIb	393.1
19	TFTIx	393.0
20	mediumESTFT_classic	392.9

We preserve the rules in R_c mainly for sake of providing a smooth transition: we don't want to convert all deterministic rules to probabilistic rules at once, as it might suddenly alter the course of our moves and lead to uncertain outcomes. Furthermore, some of these rules may still hold, and we don't want to learn them from scratch.

6. COMPETITION RESULTS

The 2005 IPD Competition was actually a set of four competitions, each for a different version of the IPD. The one for the Noisy IPD was Competition 2, which used a noise level of 0.1.

Within Competition 2 there were five runs of 200 iterations each. Table 1 shows the average scores of the top twenty programs, averaged over the five runs. The competition website at http://www.prisoners-dilemma.com gives a more extensive set of tables showing each program's ranking for each of the five runs.

We entered nine different versions of DBS into the competition, each with a different set of parameters or different implementation. The one that performed best was DBSz, which makes use of the set of features we mentioned in this paper. Versions of DBS that had fewer features or additional features than DBSz did not do as well, but all nine of versions of DBS were in the top 25, and they dominated the top ten places. This implies that DBS's performance is insensitive to the parameters in the programs and the implementation details of an individual program.

DBSz is ranked third in the table; it lost only to BWIN and IMM01. BWIN and IMM01 both used the *master-and-slaves* strategy, in which a collection of *slave* programs sacrifice their own performance in order to give as many points as possible to another program called the *master* program.

Each participant was allowed to submit 20 programs. Each of the three master-and-slaves strategies (the third one, CNGF, was not good enough to make it into our table) has one master and 19 slaves. At the start of a run, the master and slaves would each make a series of moves using a predefined protocol, in order to identify themselves to each other. From then on, the master program would always play "defect" when playing with the slaves and the slave programs would always play "cooperate" when playing with the master, so that the master would gain the highest possible payoff at each iteration. Furthermore, a slave would alway plays "defect" when playing with a program other than the master, in order to try to minimize that player's score.

DBS does not use a master-slave strategy, nor does it conspire with other programs in any other way. Nonetheless, DBS remained competitive with the master-slave strategies in the competition, and performed much better than the master-slave strategies if the score of each master is averaged with the scores of its slaves. In particular, the average score for BWIN and its slaves was 379.9, and the average score for IMM01 and its slaves was 351.7. Furthermore, a more extensive analysis [1] shows that if the size of each master-and-slaves team had been limited to less than 10, DBSz would have outperformed the best master-slave strategy in the competition, even without averaging the score of each master with its slaves.

To show the destructive effect of the master-and-slaves strategies, Table 2 gives the percentages of each of the four possible outcomes when any program from one group plays with any program from another group. Note that:

- When BWIN and IMM01 play with their slaves, about 64% and 47% of the outcomes are (D, C), but when non-master-and-slave strategies play with each other, only 19% of the outcomes are (D, C).
- When the slave programs play with non-master-and-slaves programs, over 60% of outcomes are (D, D), but when non-master-and-slaves programs play with other non-master-and-slaves programs, only 31% of the outcomes are (D, D).
- The master-and-slaves strategies decrease the overall percentage of (C, C) from 31% to 13%, and increase the overall percentage of (D, D) from 31% to 55%.

Next, we consider how DBSz performs against TFT and TFTT, (the latter, Tit-For-Two-Tats, is like TFT except that it retaliates only when the other player defected in both of the last two iterations).

- Table 2 shows that when playing with another cooperative

Table 2: Percentages of different outcomes. *"All but M&S"* means all 105 programs that did not use master-and-slaves strategies, and *"all"* means all 165 programs in the competition.

Player 1	Player 2	(C, C)	(C, D)	(D, C)	(D, D)
BWIN	BLOS's slaves	12%	5%	64%	20%
IMM01	IMS'slaves	10%	6%	47%	38%
CNGF	CNHM's slaves	2%	10%	10%	77%
BLOS's slaves	*all but M&S*	5%	9%	24%	62%
IMS's slaves	*all but M&S*	7%	9%	23%	61%
CNHM's slaves	*all but M&S*	4%	8%	24%	64%
TFT	*all but M&S*	33%	20%	20%	27%
DBSz	*all but M&S*	54%	15%	13%	19%
TFTT	*all but M&S*	55%	20%	11%	14%
TFT	*all*	23%	19%	16%	42%
DBSz	*all*	36%	14%	11%	39%
TFTT	*all*	38%	21%	10%	31%
all but M&S	*all but M&S*	31%	19%	19%	31%
all	*all*	13%	16%	16%	55%

player, TFT cooperates ((C, C) in the table) 33% of the time, DBSz does so 54% of the time, and TFTT does so 55% of the time. Furthermore, when playing with a player who defects, TFT defects ((D, D) in the table) 27% of the time, DBSz does so 19% of the time, and TFTT does so 14% of the time. From this, one might think that DBSz's behavior is somewhere between TFT's and TFTT's.

- But on the other hand, when playing with a player who defects, DBSz cooperates ((C, D) in the table) only 15% of the time, which is a lower percentage than for TFT and TFTT (both 20%). Since cooperating with a defector generates no payoff, this makes TFT and TFTT perform worse than DBSz overall. DBSz's average score was 408 and it ranked 3rd, but TFTT's and TFT's average scores were 388.4 and 388.2 and they ranked 30th and 33rd.

7. RELATED WORK

Early studies of the effect of noise in the Iterated Prisoner's Dilemma focused on how TFT, a highly successful strategy in noise-free environments, would do in the presence of noise. TFT is known to be vulnerable to noise; for instance, if two players use TFT at the same time, noise would trigger long sequences of mutual defections [18]. A number of people confirmed the negative effects of noise to TFT [4, 6, 7, 18, 19, 20]. Axelrod found that TFT was still the best decision rule in the rerun of his first tournament with a one percent chance of misperception [2, page 183], but TFT finished sixth out of 21 in the rerun of Axelrod's second tournament with a 10 percent chance of misperception [10]. In Competition 2 of the 2005 IPD competition, the noise level was 0.1, and TFT's overall average score placed it 33rd out of 165.

The oldest approach to remedy TFT's deficiency in dealing with noise is to be more forgiving in the face of defections. A number of studies found that more forgiveness promotes cooperation in noisy environments [7, 19]. For instance, Tit-For-Two-Tats (TFTT) retaliates only when it receives two defections in two previous iterations. TFTT can tolerate isolated instances of defections caused by noise and is more readily to avoid long sequences of mutual defections. However, TFTT is susceptible to exploitation of its generosity and was beaten in Axelrod's second tournament by TESTER, a strategy that may defect every other move. In Competition 2 of the 2005 IPD Competition, TFTT ranked 30—a slightly better ranking than TFT's. In contrast to TFTT, DBS can tolerate not only an isolated defection but also a sequence of defections caused by noise, and at the same time DBS monitors the other player's behavior and retaliates when exploitation behavior is detected (i.e., when the exploitation causes a change of the hypothesized policy, which initially is TFT).

[18] proposed to mix TFT with ALLC to form a new strategy which is now called Generous Tit-For-Tat (GTFT) [22]. Like TFTT, GTFT avoids an infinite echo of defections by cooperating when it receives a defection in certain iterations. The difference is that GTFT forgives randomly: for each defection GTFT receives it randomly choose to cooperate with a small probability (say 10%) and defect otherwise. DBS, however, does not make use of forgiveness explicitly as in GTFT; its decisions are based entirely on the hypothesized policy that it learned. But temporary tolerance can be deemed as a form of forgiveness, since DBS does not retaliate immediately when a defection occurs in a mutual cooperation situation. This form of forgiveness is carefully planned and there is no randomness in it.

Another way to improve TFT in noisy environments is to use contrition: unilaterally cooperate after making mistakes. However, this is less useful in the Noisy IPD since a program does not know whether its action is affected by noise or not.

A family of strategies called "Pavlovian" strategies, or simply called Pavlov, was found to be more successful than TFT in noisy environments [15, 16, 17, 21]. When an accidental defection occurs, Pavlov can resume mutual cooperation in a smaller number of iterations than TFT [15, 16]. Pavlov learns by conditioned response through rewards and punishments; it adjusts its probability of cooperation according to the previous outcome. Like Pavlov, DBS learns from its past experience. DBS, however, has an intermediate step between learning from experience and decision making: it maintains a model of the other player's behavior, and uses this model to reason about nosie.

The use of opponent modeling is common in games of imperfect information such as Poker [5, 8, 9] and RoShamBo [12]. There have been many works on learning the opponent's strategy in the non-noisy IPD [11, 13, 23]. By assuming the opponent's next move depends only on the outcomes of the last few iterations, these works model the opponent's strategy as probabilistic finite automata, and then use various learning methods to learn the probabilities in the automata. In contrast, DBS does not aim at learning the other player's strategy completely; instead, it learns the other player's recent behavior, which is subject to change. We believe that the other player may alter its strategy at the middle of a game, and therefore it is difficult for any learning method to converge. It is essentially true in noisy IPD, since noise can provoke the other player. Moreover, the length of a game is usually too short for us to learn any non-trivial strategy completely.

To the best of our knowledge, ours is the first work on using opponent models in the IPD to detect errors in the execution of another agent's actions.

8. SUMMARY AND FUTURE WORK

For conflict prevention in noisy environments, a critical problem is to distinguish between situations where another player has misbehaved intentionally and situations where the misbehavior was accidental. That is the problem that DBS was formulated to deal with.

DBS's impressive performance in the 2005 Iterated Prisoner's Dilemma competition occurred because DBS was better able to maintain cooperation in spite of noise than any other program in the competition.

To distinguish between intentional and unintentional misbehaviors, DBS uses a combination of symbolic noise detection plus temporary tolerance: if an action of the other player is inconsistent with the player's past behavior, we continue as if the player's behavior has not changed, until we gather sufficient evidence to see whether the inconsistency was caused by noise or by a genuine change in the other player's behavior.

Since clarity of behavior is an important ingredient of long-term cooperation in the IPD, most IPD programs have behavior that follows clear deterministic patterns. The clarity of these patterns made it possible for DBS to construct policies that were good approximations of the other players'

strategies, and to use these policies to fend off noise.

We believe that clarity of behavior is also likely to be important in other multi-agent environments in which agents have to cooperate with each other. Thus it seems plausible that techniques similar to those used in DBS may be useful in those domains.

In the future, we are interested in studying the following issues:

- The evidence collection process takes time, and the delay may invite exploitation. For example, the policy of temporary tolerance in DBS may be exploited by a "hypocrite" strategy that behaves like TFT most of the time but occasionally defects even though DBS did not defect in the previous iteration. DBS cannot distinguish this kind of intentional defection from noise, even though DBS has built-in mechanism to monitor exploitation. We are interested to seeing how to avoid this kind of exploitation.

- In multi-agent environments where agents can communicate with each other, there is a probability for the agents to detect noise by a predefined communication protocol. However, we believe there is no protocol that can guarantee to tell which action has been affected by noise, as long as the agents cannot completely trust each other. It would be interested to compare these alternative approaches with symbolic noise detection and see how symbolic noise detection could enhance these methods or vice versa.

- The type of noise in the competition assumes that no agent know whether an execution of an action has been affected by noise or not. Perhaps there are situations in which some agents may be able to obtain partial information about the occurrence of noise. For example, some agents may obtain a plan of the malicious third party by counter-espionage. We are interested to see how to utilitize these information into symbolic noise detection.

9. ACKNOWLEDGMENTS

This work was supported in part by ISLE contract 0508268818 (subcontract to DARPA's Transfer Learning program), UC Berkeley contract SA451832441 (subcontract to DARPA's REAL program), and NSF grant IIS0412812. The opinions in this paper are those of the authors and do not necessarily reflect the opinions of the funders.

10. REFERENCES

[1] T.-C. Au and D. Nau. An Analysis of Derived Belief Strategy's Performance in the 2005 Iterated Prisoner's Dilemma Competition. Technical Report CSTR-4756/UMIACS-TR-2005-59, University of Maryland, College Park, 2005.

[2] R. Axelrod. *The Evolution of Cooperation*. Basic Books, 1984.

[3] R. Axelrod. *The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration*. Princeton University Press, 1997.

[4] R. Axelrod and D. Dion. The further evolution of cooperation. *Science*, 242(4884):1385–1390, 1988.

[5] L. Barone and L. While. Adaptive learning for poker. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 566–573, 2000.

[6] J. Bendor. In good times and bad: Reciprocity in an uncertain world. *American Journal of Politicial Science*, 31(3):531–558, 1987.

[7] J. Bendor, R. M. Kramer, and S. Stout. When in doubt... cooperation in a noisy prisoner's dilemma. *The Journal of Conflict Resolution*, 35(4):691–719, 1991.

[8] D. Billings, N. Burch, A. Davidson, R. Holte, and J. Schaeffer. Approximating game-theoretic optimal strategies for full-scale poker. In *IJCAI*, pages 661–668, 2003.

[9] A. Davidson, D. Billings, J. Schaeffer, and D. Szafron. Improved opponent modeling in poker. In *Proceedings of the 2000 International Conference on Artificial Intelligence (ICAI'2000)*, pages 1467–1473, 2000.

[10] C. Donninger. *Paradoxical Effects of Social Behavior*, chapter Is it always efficient to be nice?, pages 123–134. Heidelberg: Physica Verlag, 1986.

[11] D. W. Dyer. Opponent modelling and strategy evolution in the iterated prisoner's dilemma. Master's thesis, School of Computer Science and Software Engineering, The Univ. of Western Australia, 2004.

[12] D. Egnor. Iocaine powder explained. *ICGA Journal*, 23(1):33–35, 2000.

[13] P. Hingston and G. Kendall. Learning versus evolution in iterated prisoner's dilemma. In *Proceedings of the Congress on Evolutionary Computation*, 2004.

[14] G. Kendall, P. Darwen, and X. Yao. The iterated prisoner's dilemma competition. http://www.prisoners-dilemma.com, 2005.

[15] D. Kraines and V. Kraines. Pavlov and the prisoner's dilemma. *Theory and Decision*, 26:47–79, 1989.

[16] D. Kraines and V. Kraines. Learning to cooperate with pavlov an adaptive strategy for the iterated prisoner's dilemma with noise. *Theory and Decision*, 35:107–150, 1993.

[17] D. Kraines and V. Kraines. Evolution of learning among pavlov strategies in a competitive environment with noise. *The Journal of Conflict Resolution*, 39(3):439–466, 1995.

[18] P. Molander. The optimal level of generosity in a selfish, uncertain environment. *The Journal of Conflict Resolution*, 29(4):611–618, 1985.

[19] U. Mueller. Optimal retaliation for optimal cooperation. *The Journal of Conflict Resolution*, 31(4):692–724, 1987.

[20] M. Nowak and K. Sigmund. The evolution of stochastic strategies in the prisoner's dilemma. *Acta Applicandae Mathematicae*, 20:247–265, 1990.

[21] M. Nowak and K. Sigmund. A strategy of win-stay, lose-shift that outperforms tit-for-tat in the prisoner's dilemma game. *Nature*, 364:56–58, 1993.

[22] M. A. Nowak and K. Sigmund. Tit for tat in heterogeneous populations. *Nature*, 355:250–253, 1992.

[23] R. Powers and Y. Shoham. Learning against opponents with bounded memory. In *IJCAI*, 2005.