Appearing in proceedings, 36th Annual Symposium on Foundations of Computer Science, November, 1995.

Efficient Algorithms for Learning to Play Repeated Games Against Computationally Bounded Adversaries

Yoav Freund
AT&T Bell Laboratories

Michael Kearns
AT&T Bell Laboratories

Yishay Mansour*
Tel-Aviv University

Dana Ron†
M.I.T.

Ronitt Rubinfeld‡
Cornell University

Robert E. Schapire
AT&T Bell Laboratories

1 Introduction

In the game theory literature, there is an intriguing line of research on the problem of playing a repeated matrix game against an adversary whose computational resources are limited in some way. Perhaps the main way in which this research differs from classical game theory lies in the fact that when our adversary is not playing the minimax optimal strategy for the game, we may be able to attain payoff that is significantly greater than the minimax optimum. In this situation, the correct measure of our performance is in comparison to the optimum achievable against the particular adversary, not to the minimax optimum.

The typical approach is to assume that the adversary's strategy is a member of some natural class of computationally bounded strategies — most often, a class of finite automata. (For a survey on the area of "bounded rationality", see the paper of Kalai [4].) Many previous papers examine how various aspects of classical game theory change in this setting; a good example is the question of whether cooperation is a stable solution for prisoner's dilemma when both players are finite automata [6, 8]. Some authors have examined the further problem of *learning* to play optimally against an adversary whose precise strategy is unknown, but is constrained to lie in some known class of strategies (for instance, see Gilboa and Samet [3]). It is this research that forms our starting point. The previous work on learning to play optimally usually does not explicitly take into account the computational efficiency of the learning algorithm, and often gives algorithms whose running time is exponential in some natural measure of the adversary's complexity; a notable recent exception is the work of Fortnow and Whang [2].

Here we examine the problem of learning to play various games optimally against resource-bounded adversaries, with an explicit emphasis on the computational efficiency of the learning algorithm. We are especially interested in providing efficient algorithms for games other than *penny-matching* (in which payoff is received for matching the adversary's action in the current round), and for adversaries other than the classically studied finite automata. In particular, we examine games and adversaries for which the learning algorithm's past actions may strongly affect the adversary's future willingness to "cooperate" (that is, permit high payoff), and therefore require carefully *planned* actions on the part of the learning algorithm. For example, in the game we call *contract*, both sides play 0 or 1 on each round, but our side receives payoff only if we play 1 in synchrony with the adversary; unlike penny-matching, playing 0 in synchrony with the adversary pays nothing. The name of the game is derived from the example of signing a contract, which becomes valid only if both parties sign (play 1). In this game, it is not enough to simply predict the adversary's actions in order to play optimally; we must also discover how to massage the adversary into his most cooperative state, in which he is willing to play 1 frequently.

As an intuitive illustration of the difference between penny-matching and contract, consider playing these two games against the same finite automaton M. For playing penny-matching, we may not need to build a detailed model of M — as shown by Fortnow and Whang, it suffices to discover a "penny-matching cycle" of M [2]. For contract, while an exact model of M may still be unnecessary, we must do enough exploration to find any regions where M plays 1 frequently. A recurrent theme of the paper is the potential utility or danger of penny-matching (and more generally, of existing computational learning theory methods) as a tool for playing contract and other games.

We defer a detailed description of our results until the main body of the paper, after we have made the necessary definitions. Here we give a brief summary. The paper and the results are divided into two main parts. In Sections 3 and 4, we introduce two new classes of adversaries. The first adver-

*Part of this research was conducted while visiting AT&T Bell Laboratories. Supported in part by the Israel Science Foundation, administered by the Israel Academy of Science and Humanities, and by a grant of the Israeli Ministry of Science and Technology.

†Part of this research was conducted while at Hebrew University and while visiting AT&T Bell Laboratories. Supported by the Eshkol Fellowship, administered by the Israeli Ministry of Science and Technology.

‡Currently visiting the MIT Lab for Computer Science. Part of this research was conducted while visiting AT&T Bell Laboratories. Supported by ONR Young Investigator Award N00014-93-1-0590 and grant No. 92-00226 from the United States-Israel Binational Science Foundation.

sary class is defined by simple boolean formulae that examine the recent history of play in order to determine their action in the current round. Adversaries in the second class base their current action on simple statistics of the entire history of play. For both classes of adversaries, we give polynomial-time algorithms for learning to play contract. In the second part of the paper in Section 5, we contribute to the literature on learning to play games against probabilistic finite automata. We give what is perhaps the most powerful positive result to date for learning to play games against automata — a polynomial-time algorithm for learning to play any game nearly optimally against any finite automaton with probabilistic actions and low cover time. If randomized transitions are allowed, we show that even approximating the optimal strategy against a given automaton is $\mathcal{PSPACE}$-complete. This improves the result of Papadimitriou and Tsitsiklis [7].

2 Models and Definitions

In this paper, games are played by two players. One will be called the *adversary* and the other the *strategy learning algorithm*. We think of the adversary as a fixed resource-bounded computational mechanism that may be playing a strategy quite different from the minimax optimal strategy for the game. The strategy learning algorithm will be a polynomial-time algorithm attempting to learn, from repeated plays, a profitable strategy for playing against the adversary.

As is typical, a game in our setting can be defined by a payoff matrix G. Here we concentrate on games in which the actions of the two players and the outcome are binary, resulting in 2×2 matrices. In game theory such matrices usually have entries of the form (a, b) where a and b represent the respective payoffs made to the two players when the chosen row and column are played. Since the strategy of the adversary is now determined by a fixed computational device, and we are interested only in the payoff made to the strategy learning algorithm, our matrices will indicate only that payoff. In a *repeated game*, play proceeds in *rounds*, with each player taking an action in each round, and the payoff to the strategy learning algorithm for that round being determined by the corresponding entry of the game matrix G. As we have indicated, in our model the adversary is chosen from a class $\mathcal{A}$ of restricted adversaries, usually defined by some resource-bounded computational device, such as finite automata, circuits of bounded size or depth, and so on. Both the strategy learning algorithm and the adversary can be regarded as (possibly probabilistic) mappings of game histories (that is, the actions made by the two players in all the rounds so far) to current actions. After N rounds, the *payoff* earned by a strategy g with respect to the adversary f is

$$PAYOFF_f^N(g) = E\left[\frac{1}{N}\sum_{i=1}^{N} G_{g(h_{i-1}),f(h_{i-1})}\right]$$

where h_i is the history of the first i moves (that is, $h_i = h_{i-1}, \langle g(h_{i-1}), f(h_{i-1})\rangle$), and $G_{a,b}$ is the entry in the matrix G which defines the payoff when player g takes action a and player f takes action b. The *optimal strategy* for N rounds against f, opt_f^N, is the strategy that maximizes $PAYOFF_f^N(g)$.

An algorithm L is called an *ϵ-good strategy learning algorithm* for playing the game G against a class of adversaries $\mathcal{A}$ if for every $f \in \mathcal{A}$ there exists an integer N_0 such that for all $N > N_0$

$$PAYOFF_f^N(L) \geq PAYOFF_f^N(opt_f^N) - \epsilon.$$

We call N_0 the *initialization time* of L for the given ϵ. If L is ϵ-good for every ϵ, we say that L is simply a *strategy learning algorithm* for playing G against $\mathcal{A}$. We say L is efficient if its initialization time is polynomial in $1/\epsilon$ and $size(f)$, and its computation time at any round N is polynomial in $1/\epsilon$, N, and $size(f)$. As is typical in learning theory, here $size(f)$ is an appropriate notion of the complexity of the adversary f (such as the number of states if f is defined by a finite automaton, or the number of gates if f is defined by a boolean circuit); it is natural to allow L more computation to succeed against more complex adversaries. If L is a probabilistic algorithm then we require that it succeed only with probability $1-\delta$ for a given confidence parameter δ (and, as usual, allow it to run in time polynomial in $\log(1/\delta)$). We will also have occasion to consider strategy learning algorithms L that can only achieve a payoff that is a multiplicative constant factor $c < 1$ of the optimal payoff; we call such algorithms *c-competitive*.

In the first part of the paper, we concentrate on two specific games. The first game, called *penny-matching*, gives payoff 1 to the strategy learning algorithm if and only its play matches that of the adversary — that is, if and only if both players play 0 or both players play 1. Otherwise, the payoff to the strategy learning algorithm is 0. Thus the game matrix is $G_{0,0} = G_{1,1} = 1, G_{0,1} = G_{1,0} = 0$. We will often use the term penny-matching to refer not only to this game itself, but also to the *strategy* (regardless of the game being played) of always playing the same action as the adversary. The second game we examine is called *contract*. Here the strategy learning algorithm receives payoff 1 if and only if both players play 1 (sign the contract), otherwise it receives payoff 0. Thus the game matrix is $G_{1,1} = 1, G_{0,0} = G_{0,1} = G_{1,0} = 0$.

Despite the fact that the penny-matching and contract game matrices differ on only a single entry (when both players play 0), they capture a fundamental distinction between game types, especially in the context of resource-bounded adversaries. In penny-matching, regardless of the adversary's current action (which may depend strongly on the history of play), there is always some action that the strategy learning algorithm may take to achieve the maximum possible payoff in the entire matrix (namely, match the adversary's play). In contract, however, if the adversary is currently playing 0, there is no action the strategy learning algorithm may make in order to achieve nonzero payoff. Thus, succeeding at penny-matching requires only that the strategy learning algorithm learn to *predict* the adversary's actions, while contract may require the strategy learning algorithm to *influence* the adversary's actions, possibly through careful planning over many rounds of play.

3 Games Against Recent History Adversaries

Much of the previous work on playing games against computationally bounded adversaries has concentrated on finite automata adversaries, and there has been little work on *learning* to play optimally except for the game of penny-matching. In this section and the next, we investigate the problem of learning to play contract against two types of adversaries that are rather different from those defined by small automata. The first type of adversary is defined by certain boolean formulae over the recent history of play, and the second type of adversary is defined by linear functions of simple statistics of the entire history of play. In both cases, exponentially many states may be required to describe the same adversaries as finite automata. We begin by introducing the general framework of recent history adversaries, then specialize to the simple class that we can analyze.

Let us introduce the *history vectors* of length ℓ at round N ($N \geq \ell$), $\vec{u}^N, \vec{v}^N \in \{0,1\}^\ell$, where the variable u_i^N contains the binary value that was played by the strategy learning algorithm at round $N-i$, and the variable v_i^N contains the binary value that was played by the adversary at round $N-i$ ($1 \leq i \leq \ell$). When the current round N is still smaller than the history length ℓ, $\vec{u}^N$ and $\vec{v}^N$ are defined to be the vectors of length N which contain the actions played by the strategy learning algorithm and the adversary, respectively, in the first N rounds. For simplicity, when no confusion will result we will drop the superscript N indicating the index of the current round. Thus $\vec{u}$ and $\vec{v}$ simply store the last ℓ rounds of play, and they are continually updated during play by shifting after each round (with the plays made $\ell+1$ rounds ago being lost). A *recent history adversary* is an adversary whose current play is defined by some boolean function $f(\vec{u}, \vec{v})$. As a simple example, consider the adversary for contract defined by the boolean formula

$$f(\vec{u}, \vec{v}) = \bigwedge_{i=1}^{\ell-1} (\neg u_i \vee \neg v_i \vee \neg u_{i+1} \vee \neg v_{i+1}),$$

The clause $(\neg u_i \vee \neg v_i \vee \neg u_{i+1} \vee \neg v_{i+1})$ forbids $u_i = v_i = u_{i+1} = v_{i+1} = 1$, which in contract means that the strategy learning algorithm enjoyed two consecutive payoffs $i+1$ rounds ago. Thus, the adversary defined by f simply plays 1 if and only if the strategy learning algorithm has not received two consecutive payoffs in the last ℓ rounds, a type of "profit-limiting" adversary.

Clearly, we would like to efficiently learn to play various games nearly optimally under the least restrictive assumptions on the recent history adversary f. Not surprisingly, however, a number of the computational limitations governing more standard learning models translate to the current setting. For example, we should not expect to efficiently learn to play penny-matching against recent history adversaries that are boolean circuits (over the input variables $\vec{u}$ and $\vec{v}$) of size polynomial in the history length ℓ, since such adversaries can compute pseudo-random functions of the history. (This claim will be made formally in the full paper). Similar arguments apply to learning to play contract.

On the positive side, suppose that the recent history adversary $f(\vec{u}, \vec{v})$ is drawn from a class $\mathcal{A}$ of boolean functions for which there is an absolute mistake-bounded algorithm [5] — that is, a prediction algorithm A that makes at most $m(\ell, size(f))$ mistakes in predicting the values of any target function chosen from $\mathcal{A}$ on *any* sequence of 2ℓ-bit inputs (history vectors), where $m(\ell, size(f))$ is a polynomial. Then it is easy to see that A is in fact an algorithm that will efficiently learn to play penny-matching nearly optimally, because since $m(\ell, size(f))$ does not depend on N, the fraction of rounds in which we have failed to match decreases at a rate of $O(1/N)$. This logic fails for contract, however, where our goal is not simply that of prediction, but of maximizing the number of rounds where we play 1 in synchrony with the adversary. We now investigate further this apparent difference between penny-matching and contract for a particular class of recent history adversaries.

Consider the class of recent history adversaries defined by the boolean formulae $f_S(\vec{u}, \vec{v})$, where $S \subseteq \{1, \ldots, \ell\}$ is an index set, and

$$f_S(\vec{u}^N, \vec{v}^N) = \bigvee_{i \in S,\ i \leq N} (\neg u_i \wedge v_i).$$

The restriction $i \leq N$ in the subscript is clearly redundant for $N \geq \ell$, and is added only to ensure that f_S be well defined on history vectors of length less than ℓ. We adopt the convention that for every S, $f_S(\lambda, \lambda) = 1$, where λ denotes the empty history vector. We do this because if we allow $f_S(\lambda, \lambda) = 0$, then f_S will play 0 on all future rounds, and *no* strategy can achieve payoff higher than 0. Taken together, the restriction $i \leq N$ in the subscript above and the condition $f_S(\lambda, \lambda) = 1$ are equivalent to assuming that the history vectors are initialized to contain a single sacrifice in the most recent round, and all 0 entries for both players in the $\ell - 1$ previous rounds.

Note that f_S is a 2-DNF formula over the variables $u_1, \ldots, u_\ell, v_1, \ldots, v_\ell$. Since such formulae have an efficient absolute mistake-bounded algorithm [5], our comments above imply that we can learn to play penny-matching nearly optimally against such adversaries in polynomial time. Let us now examine the more subtle problem of playing contract against this class. The term $(\neg u_i \wedge v_i)$ stipulates that i rounds ago, the adversary f_S played 1, but the strategy learning algorithm played 0. We call such a round a *sacrifice*, because since f_S played 1, the strategy learning algorithm could have obtained a payoff by playing 1 but instead played a 0. If a sacrifice occurred i rounds ago for any i in the index set S, the adversary plays 1 in the current round; otherwise the adversary plays 0. We call the set $\{f_S\}$ the *past sacrifice* adversaries for contract. The past sacrifice adversaries are interesting for the direct tension they create between obtaining payoff in the current round, and ensuring future payoff, which requires the occasional insertion of sacrifices into the history. Even a player who knows f_S and wishes to play contract optimally cannot pursue the greedy strategy of always playing 1 in synchrony with f_S — the reader can easily verify that this would quickly drain the history vectors of any sacrifices they might have contained, and no further contract payoff would be possible.

Thus, while the f_S are rather trivial adversaries for playing penny-matching, a different approach is required for playing contract against this same class. We now give an efficient strategy learning algorithm for playing contract against the past sacrifice adversaries that is $1/2$-competitive (that is, achieves payoff at least $1/2$ times the optimal payoff). The analysis shows that the optimal payoff has an interesting number-theoretic characterization.

Theorem 3.1 *There is an efficient $1/2$-competitive strategy learning algorithm for playing contract against the class of past sacrifice strategies.*

Proof (Sketch): The proposed strategy learning algorithm L operates in two simple phases. In the first phase (called the *sacrifice phase*), L always plays 0 (regardless of the actions of f_S) in order to "flood" the history vectors with as many sacrifices as possible. In the second phase (the *payoff phase*), L attempts to alternate periods of sacrifice with periods of payoff in a pattern that can be maintained indefinitely. We begin by giving a characterization of the rate at which sacrifices are obtained during the sacrifice phase of L. This rate also provides us with an upper bound on the optimal payoff.

Lemma 3.2 *Let f_S be any past sacrifice adversary for contract, and consider the strategy that always plays 0 against f_S. Then after at most ℓ^3 rounds of play, sacrifices will occur every g rounds, where $g = \gcd(S)$ is the greatest common divisor of the indices appearing in S. Furthermore, the optimal payoff that can be obtained against f_S by* any *player is at most $1/g$.*

Proof (Sketch): For the upper bound on the optimal payoff, first consider any strategy R for playing against f_S, and let us credit R with payoff 1 at the current round if f_S plays 1, regardless of whether R has played 0 or 1. Then we will only overcount the actual payoff of R, and now we can assume without loss of generality that R plays to maximize the frequency with which f_S plays 1. Since R's playing 1 in the current round can never cause f_S to play 1 in a future round, the best strategy for R under this analysis is to always play 0. If $S = \{i_1, \ldots, i_r\}$, this will cause f_S to play 1 at rounds $i_1, \ldots, i_r$, which are the rounds that the lone sacrifice in the initial configuration "passes through" the relevant indices in the history vectors. In general, if f_S plays 1 at round N, this inserts another sacrifice into the history (since R is playing 0), leading to sacrifices at the later rounds $N + i_1, \ldots, N + i_r$. Thus all the sacrifice rounds have the form $N = a_1 i_1 + \cdots a_r i_r$ for some natural number coefficients $a_1, \ldots, a_r$. This means that N is a multiple of $g = \gcd(i_1, \ldots, i_r)$, and we have proven that any strategy for playing contract against f_S can only obtain payoff at rounds that are multiples of g (call such rounds *g-rounds*), as desired for the upper bound on the optimal payoff.

We now show that after at most ℓ^3 rounds the strategy of always playing 0 against f_S will in fact result in a sacrifice on *every* g-round. To see this, recall that we can write $g = c_1 i_1 + \cdots c_r i_r$ for some integer coefficients $c_1, \ldots, c_r$; however, some of the c_i may be negative. Let $N = a_1 i_1 + \cdots a_r i_r$ be a sacrifice round, where the natural numbers $a_1, \ldots, a_r$ are sufficiently large (as determined by the analysis below). We now wish to show that $N + g, N + 2g, \ldots, N + kg$ are also sacrifice rounds, where $kg = i_1$; this suffices to show that all future g-rounds will be sacrifice rounds, since once the history vectors contain a sacrifice on every g-round between the current round and index i_1, the sacrifices passing through i_1 alone will cause new sacrifices at this same rate from then on. To see that $N + xg$ is a sacrifice round for every $1 \leq x \leq k$, we must show that it can be written as a linear combination of the indices in S with *natural number* coefficients. We have $N + xg = (a_1 i_1 + \cdots a_r i_r) + x(c_1 i_1 + \cdots c_r i_r) = (a_1 + xc_1) i_1 + \cdots (a_r + xc_r) i_r$. Provided we choose the natural numbers $a_j \geq |xc_j|$, the coefficients of $N + xg$ will also be natural numbers, and $N + xg$ will be a sacrifice round. It can be shown that the c_j can be chosen so that their absolute values are all bounded by $i_r \leq \ell$. It directly follows that the minimal value of N for which $N + xg$ is a sacrifice round for every $x \geq 0$ is at most ℓ^3, as required. □(Lemma 3.2)

Thus, the sacrifice phase of our strategy learning algorithm L continues until the history vectors are *g-flooded* with sacrifices — that is, until they contain a sacrifice every g indices. By Lemma 3.2, this occurs after at most ℓ^3 rounds. (Note that L can quickly infer the value of g from the pattern of sacrifices that emerges.) At this point, L attempts to enter its payoff phase. Since L's behavior on rounds that are not g-rounds is irrelevant, we concentrate on L's behavior on g-rounds. Of course, L cannot simply begin playing 1 indefinitely, since this would quickly drain the history vectors of *all* sacrifices, and no further payoff would be possible — a balance between payoff and preservation of sacrifices is required. To begin with, we dismiss the degenerate case in which the index set S contains only a single index i_1: it can be verified that in this case only a single payoff can ever be obtained, so the average payoff over time approaches 0. Thus without loss of generality $|S| \geq 2$.

Suppose first that we knew the first two indices i_1, i_2, and let $i_2 - i_1 = dg$ for a natural number $d \geq 1$. Consider the strategy of restoring sacrifices for the next d g-rounds, then obtaining payoff for the following d g-rounds. We claim that this pattern can be maintained indefinitely, because at every g-round either i_1 or i_2 contains a past sacrifice, and thus f_S will play 1. The reason is that a single payoff block of d g-rounds is too short to cover both i_1 and i_2, but these payoff blocks are too long for i_1 and i_2 to be covered by different blocks of payoff.

It remains to find i_1 and i_2. This can be done on the first rounds of play (prior to the sacrifice phase) in the following simple manner. The strategy learning algorithm plays 1 until the adversary first plays 1 (in round i_1) and then switches to playing 0 until the adversary next plays 1 (at round i_2). Note that if r is known, then all the indices in S can be found in a similar manner, and the strategy defined above when only the first two indices are known can be easily generalized to get higher average payoff. However, if r is not known, it is not clear how the algorithm can aquire this information without risking the loss of all sacrifices in its recent history. $\square$(Theorem 3.1)

4 Games Against Statistical Adversaries

In this section we study the problem of playing contract against an adversary whose current move depends on the *entire* (unbounded) history of play so far (in contrast to the recent history adversaries of the last section), but only through the *statistics* of the history. Here we consider only the simplest statistics of the history, the values $p_{00}^N, p_{11}^N, p_{01}^N, p_{10}^N$, where for $a, b \in \{0,1\}$ the value p_{ab}^N is the fraction of the first N rounds in which the strategy learning algorithm played a and the adversary played b. Where no confusion will result, we drop the superscript N indicating the current round. As a simple example, we might consider a contract adversary that plays 1 in the current round if and only if $p_{11} < \theta$ for some threshold $\theta \in [0,1]$. This adversary is willing to let its opponent receive a payoff only if the opponent is not too wealthy already. In general, one might consider adversaries whose current play is a complicated function of not only the p_{ab} but also of other statistics of the history. Here we make a modest start by giving an efficient strategy learning algorithm for playing contract against adversaries that play 1 if and only if the statistics $\{p_{ab}\}$ obey a linear inequality; we refer to such adversaries as *linear statistical adversaries*. We thus revisit the classical linear separator, but this time in the context of learning to play games. For simplicity we concentrate on the two-dimensional case in which the adversary is determined by a linear constraint over only the statistics p_{00} and p_{11}; the general three-dimensional case[1] is similar and will be considered in the full paper. We state our theorems below for contract, but we point out that since absolute mistake bounded algorithms do not exist for linear separators in $\Re^2$ (a point elaborated upon below), even learning to play penny-matching against linear statistical adversaries nearly optimally is a nontrivial problem. Our algorithms for playing contract contain penny-matching subroutines, and thus solve the penny-matching problem as well. The reason that penny-matching is a useful subroutine here is that for certain $f_{\alpha,\beta}$, penny-matching moves us on a trajectory through the $p_{11} - p_{00}$ plane that rapidly approaches the optimal point, and remains there stably. Thus in these cases penny-matching can be thought of as an effective means of motion in the $p_{11} - p_{00}$ plane.

[1]Note that there are really only three dimensions here since $\sum_{a,b\in\{0,1\}} p_{ab} = 1$ always holds.

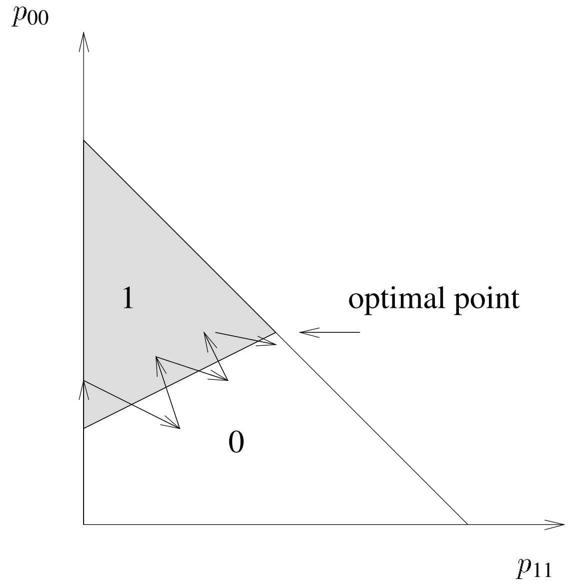

Figure 1: Example of a statistical adversary $f_{\alpha,\beta}(p_{00}, p_{11})$, showing the regions in which the adversary plays 0 and 1, the diagonal line representing the constraint $p_{00} + p_{11} \leq 1$, and the optimal point of play lying on this line. Also shown is a typical trajectory taken by the strategy of penny-matching when playing contract against $f_{\alpha,\beta}$. Matching ones results in an increase to p_{11} and a decrease to p_{00}, while matching zeros has the reverse effect, resulting in the alternating pattern shown. The distance traveled with each successive match decreases with time.

Theorem 4.1 *For any $\alpha, \beta \in \Re$, let $f_{\alpha,\beta}(p_{00}, p_{11}) = 1$ if and only if $\alpha p_{11} + \beta < p_{00}$. Then there is an efficient strategy learning algorithm for playing contract against the linear statistical adversary class $\{f_{\alpha,\beta} : \alpha, \beta \in \Re\}$.*

Proof (Sketch): We restrict our attention to the case $\alpha > 0$, which is the most interesting, and for which a representative picture is shown in Figure 1. The general α case will be considered in the full paper; the ideas are quite similar but some care is required in the handling of initial conditions. The main points of the proof are: (1) For $\alpha > 0$ (implying that the linear separator has positive slope in the $p_{11} - p_{00}$ plane) the strategy of penny-matching (that is, always playing identically to the adversary $f_{\alpha,\beta}$) is in fact an optimal contract strategy [2]; (2) Despite the fact that the well-known halving algorithm can be forced to make many prediction mistakes on an *arbitrary* sequence of trials, on the sequence actually generated by the halving algorithm playing contract against $f_{\alpha,\beta}$, only a logarithmic number of mistakes will be made; and (3) The logarithmic number of prediction mistakes have a small effect on the contract payoff.

We assume in the following analysis that the initial conditions are $p_{00} = p_{11} = 0$. Other initial conditions can be treated similarly. We begin by considering the payoff obtained when the halving algorithm plays contract against a simple one-dimensional adversary $f_\theta(p_{11})$, where $f_\theta(p_{11}) = 1$ if and only if $p_{11} < \theta$. The reader can easily verify that against such an adversary, the optimal contract payoff at round N is within $1/N$ of θ, and that this optimum can be achieved by always playing 1, but also by (perfectly) penny-matching against f_θ. Here we are interested in analyzing the contract payoff of the one-dimensional halving algorithm L_1, since this analysis will include many of the elements required to solve the two-dimensional case.

At every round N, L_1 keeps track of the *version space* $[\theta_L, \theta_R)$ for θ. Here θ_L is the largest past value of p_{11} for which the adversary played 1, and θ_R is the smallest past value of p_{11} for which the adversary played 0. At all times θ lies in the version space. At the current round, L_1 plays 1 if and only if $p_{11} < (\theta_L + \theta_R)/2$. If L_1's play fails to match that of f_θ, then the width of the version space has been at least halved.

In the usual models of on-line prediction [5], the sequence of inputs x_i to f_θ to be classified (that is, penny-matched) is arbitrary, and the halving algorithm may be forced to make N prediction mistakes in N trials due to the arbitrary precision of the x_i (an unfavorable sequence would always arrange the next x_i to fall in the current version space). To see that the present situation is considerably more favorable, we make the following definition: $p \in [0, 1]$ has *rational complexity* at most N if p is a rational number whose numerator and denominator are both at most N. Then at every round N, θ_L and θ_R have rational complexity at most N, since they represent frequencies of certain events over at most N rounds of play. It is easy to verify that this implies $\theta_R - \theta_L > 1/N^2$. Thus if m is the number of prediction mistakes made by L_1 on the first N rounds, we must have $1/2^m > 1/N^2$, or $m < 2\log N$. We now argue that these few prediction mistakes made by L_1 have a rather small effect on the contract payoff that L_1 receives in comparison to perfect penny-matching, which is an optimal contract strategy.

Lemma 4.2 *For any $\epsilon > 0$, the contract payoff of algorithm L_1 when playing against f_θ is at least $\theta - \epsilon$ within $O((1/\epsilon)\log(1/\epsilon))$ rounds.*

Proof: First, for any small $\epsilon > 0$, we upper bound N, the number of rounds required for p_{11} to exceed $\theta - \epsilon/2$ for the first time when L_1 plays contract against f_θ. Since L_1 makes at most $2\log N$ prediction mistakes in the first N rounds, and since f_θ is playing 1 as long as $p_{11} < \theta$, we must have $p_{11} \geq 1 - (2\log N)/N$ at round N. The desired inequality $p_{11} \geq 1-(2\log N)/N \geq \theta - \epsilon/2$ holds for $N = O((1/\epsilon)\log(1/\epsilon))$.

We next argue that if p_{11} falls below $\theta - \epsilon/2$ immediately following some round N, then on or before round $2N$, p_{11} exceeds $\theta - \epsilon/2$ again, and for all rounds in between N and $2N$, p_{11} is at least $\theta - \epsilon$. We know that between rounds N and $2N$, L_1 will make at most $2\log(2N)$ prediction mistakes, and that each such mistake decreases p_{11} by at most $1/N$. Thus the total decrease to p_{11} between rounds N and $2N$ is at most $2\log(2N)/N$. On the other hand, as long as $p_{11} < \theta - \epsilon/2$, each correct prediction by L_1 increases p_{11} by at least $(1 - (\theta - \epsilon/2))/2N \geq \epsilon/4N$. This is because an increase to p_{11} following round N is always given by $p_{11} \leftarrow (N/(N+1))p_{11} + 1/(N+1)$, or equivalently $p_{11} \leftarrow p_{11} + (1-p_{11})/(N+1)$. Thus the total increase to p_{11} between rounds N and $2N$ is at least $(N - 2\log(2N))(\epsilon/4N)$. If N satisfies $(N-2\log(2N))(\epsilon/4N) > (2\log(2N))/N$ then we know that p_{11} has increased from rounds N to $2N$, and hence p_{11} again exceeds $\theta - \epsilon/2$ by round $2N$. The stated inequality holds for $N = O((1/\epsilon)\log(1/\epsilon))$. Finally, for all rounds in between N and $2N$ we know that p_{11} must exceed $\theta - \epsilon/2 - (2\log(2N))/N$. Thus if $(2\log(2N))/N \leq \epsilon/2$, p_{11} is within ϵ of θ for all rounds in between N and $2N$. Again, $N = O((1/\epsilon)\log(1/\epsilon))$ suffices. □(Lemma 4.2)

Armed with the analysis of algorithm L_1, we can now return to the two-dimensional adversary $f_{\alpha,\beta}(p_{00}, p_{11})$. In this case, the optimal contract payoff is the largest value $p \in [0, 1]$ such that $f_{\alpha,\beta}(1-p, p) = 1$. In other words, an optimal contract strategy tries to maintain $p_{00} = 1 - p$ and $p_{11} = p$ for p as large as possible; see Figure 1. The reader may verify that penny-matching against $f_{\alpha,\beta}$ will again achieve this optimum (see Figure 1 for a typical trajectory taken by penny-matching), and also that the naive strategy of always playing 1 (which *did* achieves the optimum against the one-dimensional f_θ) will *not* succeed.

Thus, our goal will again be that of approximately penny-matching, this time using the two-dimensional halving algorithm L_2 for the $f_{\alpha,\beta}$. The version space is now more complicated than in the one-dimensional case, as is the analysis bounding the number of prediction mistakes made by L_2. However, the basic proof outline is similar. It can be shown that the version space, which is the region in the $\alpha - \beta$ plane consistent with the history so far, is always a convex polygon. Furthermore, the vertices of this polygon, while no longer equal to past values of p_{11}, are nevertheless determined by past values of p_{00} and p_{11}, and have rational complexity polynomial in N. This implies a $1/poly(N)$ lower bound on the version space area at round N, leading again to an $O(\log N)$ bound on the number of prediction mistakes by L_2 (details in

[2]This statement does not hold for $\alpha < 0$, but this case can be handled by separate methods.

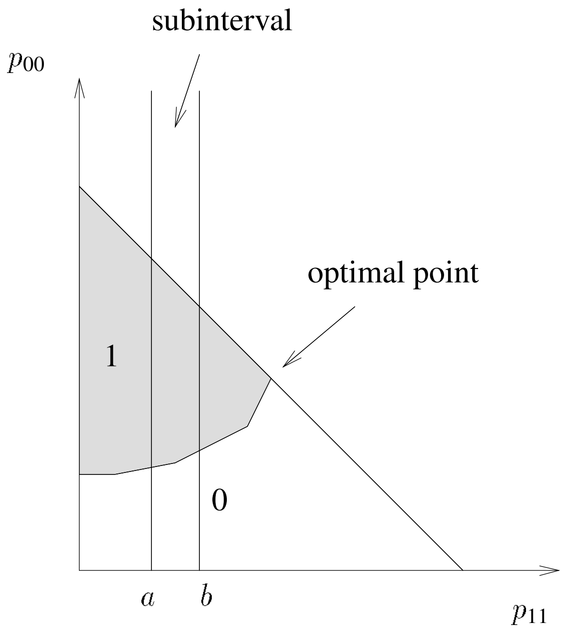

Figure 2: Example of a statistical adversary defined by an intersection of 4 lines of increasing slope, showing the regions in which the adversary plays 0 and 1, the line representing the constraint $p_{00} + p_{11} \leq 1$, and the optimal point of play lying on this line. Also shown is the vertical strip defined by the subinterval $[a, b)$. The examples falling in this subinterval are used to compute a local version space by the algorithm of Theorem 4.3. Since this particular subinterval includes a corner point of the adversary, at some point the local version space may become empty, resulting in the splitting of the subinterval.

the full paper). To see that L_2 is efficient, note that computing the area of a convex polygon in the plane can be done in time polynomial in the number of vertices.

The preceding analysis implies that when L_2 plays against $f_{\alpha,\beta}$, we must have $p_{00} + p_{11} \geq 1 - O((\log N)/N)$ for round N and all later rounds. At this point, we have almost reduced the analysis to the one-dimensional case, since now L_2 is approximately penny-matching within a narrow ribbon of the line $p_{00} + p_{11} = 1$, which includes the optimal configuration. An analysis similar to that given in Lemma 4.2 shows that L_2 will converge rapidly to this optimal configuration. □(Theorem 4.1)

We conclude this section by considering the more complicated case in which the adversary's 1-region is an *intersection* of d half-spaces, rather than just a single half-space. We seek a strategy learning algorithm that is polynomial in d. We again assume that all of the half-spaces are defined by lines with positive slope in the $p_{11} - p_{00}$ plane; a representative adversary is shown in Figure 2.

Theorem 4.3 *There is an efficient strategy learning algorithm for playing contract against any intersection of d half-spaces defined by lines of positive slope in the $p_{11} - p_{00}$ plane.*

Proof (Sketch): Let f be the adversary function. Again it is easily verified that in the case of positive slopes, the optimal configuration is the dividing point between the 0 and 1 regions on the line $p_{00} + p_{11} = 1$, and that penny-matching will rapidly converge to this optimum. However, now the version space has $2d$ parameters and the volume computation required for directly implementing the halving algorithm has no known polynomial time solution (in particular, since histories for which the adversary plays 0 result in a disjunction of linear constraints on the $2d$ parameters, the version space is no longer necessarily a convex body). Our algorithm will still attempt to penny-match, but using a different strategy.

The algorithm partitions the p_{11} axis into disjoint subintervals. Initially, there is only the single subinterval $[0, 1]$. At any time, within each subinterval $[a, b)$, the algorithm maintains a *local version space* for a *single line* by using only the past examples $\langle (p_{00}, p_{11}), f(p_{00}, p_{11}) \rangle$ for which $p_{11} \in [a, b)$ (see Figure 2). Notice that if the vertical strip of the $p_{11} - p_{00}$ plane defined by the constraint $p_{11} \in [a, b)$ does not contain a "corner" of the adversary f (meaning that the boundary of between the 0 and 1 regions of f is a straight line when restricted to this vertical strip), then the local version space for $[a, b)$ will always be non-empty. Otherwise, at some point the examples falling into the subinterval $[a, b)$ may become inconsistent with any linear classifier — that is, the local version space for $[a, b)$ may becomes empty. When this happens, we split the subinterval $[a, b)$ into the two subintervals $[a, (a + b)/2)$ and $[(a + b)/2, b)$ and compute the local version spaces for these two new subintervals using the past examples that fall into each subinterval. For at least one of the new subintervals, the local version space must be non-empty, since only the subinterval containing the last example can have an empty local version space (recall that prior to the last example, the entire subinterval $[a, b)$ had a non-empty local version space, so any subinterval has a non-empty local version space if the last example is omitted). As long as a new subinterval has an empty local version space, we divide it again. However, since newly created subintervals that do not contain the last example and are adjacent can be merged together again to get a larger subinterval with a non-empty local version space, in the worst case we divide $[a, b)$ into three new subintervals $[a, c), [c, d), [d, b)$, each with non-empty local version spaces.

The main point is that each time a local version space becomes empty, we create at most three new subintervals, while reducing our uncertainty about the location of a corner of f by at least $1/2$ (since there must have been a corner in $[a, b)$ if its local version space became empty, and now this corner lies in a subinterval of length at most $(b - a)/2$). Furthermore, since the values of p_{00} and p_{11} have low rational complexity, the subintervals always have length at least $1/N^2$ at round N.

This allows us to guarantee that there are at most $O(d \log N)$ prediction mistakes caused by empty local version spaces. In each subinterval we make at most $2 \log N$ prediction mistakes, so the total number of mistakes is bounded by $O(d \log^2 N)$. This can be shown to imply that the algorithm will rapidly reach the optimal point in the $p_{11} - p_{00}$ plane for playing contract against f. $\square$(Theorem 4.3)

5 Games Against Probabilistic Automata

We now turn to the problem of playing games against adversaries defined by *probabilistic finite automata*. This continues a line of research investigated by Gilboa and Samet [3], and more recently by Fortnow and Whang [2] and Vovk [12]. Our main result is a polynomial-time algorithm for learning to play *any* game nearly optimally against any finite automaton with probabilistic states (defined below) and small cover time. We also prove that if the adversary automaton has probabilistic transitions (defined below) then even the problem of approximating the optimal payoff when the automaton is given to the algorithm is $\mathcal{PSPACE}$-complete (improving on a result of Papadimitriou and Tsitsiklis [7]), which precludes a positive result for learning to play optimally against such machines.

5.1 Probabilistic State Automata

A probabilistic state automaton (PSA) can be thought of as a generalization of a DFA in which each state contains a different biased coin (with the usual definition of a DFA being obtained by restricting all the biases to be either 0 or 1). In state q, the PSA generates a single output bit (action) by flipping the biased coin at q. As in a DFA, the next state is still determined solely by the next input bit (action).

More formally, a *Probabilistic State Automaton* M is a tuple (Q, q_0, γ, τ), where Q is a finite set of n *states*, $q_0 \in Q$ is the designated *starting state*, $\gamma : Q \rightarrow [0, 1]$, is the *state bias function*, and $\tau : Q \times \{0, 1\} \rightarrow Q$ is the *transition function*. The adversary PSA M is initially in its starting state q_0. If the state in round N is q, then M plays action 1 with probability $\gamma(q)$, and action 0 with probability $1 - \gamma(q)$. If the action of the strategy learning algorithm in the Nth round is $b \in \{0, 1\}$, then M's state in round $N + 1$ is $\tau(q, b)$. Thus, the strategy learning algorithm observes the outcome of the adversary's biased coin flip for the current round only *after* choosing its own action for the current round. Note that the outcome of the biased coin flip is the only information the strategy learning algorithm receives about the current state; this is in contrast to much of the research on Markov decision processes, where the learning algorithm is told the identity of the current state.

Notice that although PSA's flip coins at each state, the fact that their transitions remain deterministic preserves the usual notion of cycles — namely, a state q and a sequence of actions (of the strategy learning algorithm) from q that returns to q. As in the case of DFA's, for every PSA there exists an optimal strategy that is a repeated simple cycle [3]. Moreover, if the automaton is given, the optimal cycle can be found efficiently using dynamic programming. Gilboa and Samet [3] note that their algorithm for learning an optimal cycle strategy against DFA's can be easily adapted to PSA's. However, as in their original strategy for DFA's, it may take an exponential number of rounds (exponential in the number of states of the adversary PSA) to achieve payoff which is close to optimal. Fortnow and Whang [2] showed that in the case of DFA's, if the underlying game played has a certain property (held by contract but not by penny-matching), then there is also an exponential lower bound for the number of rounds needed before approaching optimality.

It is interesting to note that if the underlying game is penny-matching then the exponential lower bound does not hold. Fortnow and Whang give an efficient strategy learning algorithm for playing penny-matching against DFA's. Unfortunately, it does not seem that penny-matching is any easier than contract when playing against PSA's. The "combination lock" lower bound argument given by Fortnow and Whang for contract can easily be modified to give an exponential lower bound for learning to play penny-matching against PSA's. The important property of combination lock automata is the existence of a state which can be reached only when the strategy learning algorithm plays a specific sequence of $n - 1$ actions. For this remote state, there exists an action which achieves high expected payoff, and which allows the algorithm to remain in the same state. For every other state, the expected payoff is low no matter what action is chosen by the learning algorithm. A natural question is whether we can obtain efficient algorithms when we restrict our attention to automata that do not have states that are hard to reach — that is, automata whose underlying graphs have small (polynomial) cover time[3] We answer this question in the affirmative in the theorem below. Our result improves on the work of Ron and Rubinfeld[10], who examine the special case of state bias functions that take only two possible values η and $1 - \eta$ for $0 \leq \eta \leq 1$.

Theorem 5.1 *For any game G, there is an efficient strategy learning algorithm for playing G against the class of probabilistic state automata with polynomial cover time.*

Proof (Sketch): Let M denote the adversary PSA with small cover time. In what follows we describe an efficient strategy learning algorithm that attempts to construct a good approximation $\widehat{M}$ to M after playing a polynomial number of rounds against M. The automaton $\widehat{M}$ will be a good approximation to M in the sense that for any sequence of actions, if the sequence is played against both M and $\widehat{M}$, then the bias of the coin in the state reached in $\widehat{M}$ will be close to the bias of the coin in the state reached in M. We refer to this as *PSA learning*, since

[3]The *cover* time of a directed graph is defined to be the smallest integer ℓ such that for every vertex v, the probability that a random walk of length ℓ from v will pass through all the vertices of the graph is greater than one half.

it is similar to standard models of automaton learning in its construction of a complete model of M, and does not explicitly account for the fact that the goal is to play a game against M. However, if we play against M according to an optimal cycle computed for $\widehat{M}$, then our payoff will be close to optimal. It should be noted that the quality of approximation needed may depend on the game G being played — for instance, if some entry of the game matrix is much larger than the other entries, then high expected payoff may be achieved by hitting this entry only very rarely, in which case we will need better approximations of the state coin biases. For simplicity in the description below, we will assume the game matrix has only entries from $\{0, 1\}$.

In order to simplify the presentation we start by making two assumptions; the first can be made without loss of generality, and the second can be removed using a technique due to Ron and Rubinfeld[11]. Assumption (1): M is irreducible in the sense that there is no smaller automaton equivalent to it. Assumption (2): There exists some value $\Delta = poly(\epsilon/n)$ such that for every state q in the adversary automaton M, the state coin bias $\gamma(q)$ satisfies $\gamma(q) = k\Delta$ for some integer k. Thus, any pair of states may contain coins with either the same bias or significantly different biases.

The standard problem that arises when trying to learn automata of various types from a single continuous sequence of actions is the need for a procedure that *orients* the learner by providing evidence that it has returned to a state it has previously visited. In fact, it is possible to show [10] that in the case of small cover time, the PSA learning problem reduces to the problem of orientation (details omitted), and thus we concentrate on this latter problem. A well-studied procedure for orientation when learning DFA's uses *homing sequences* [9]. A homing sequence for a DFA is an action sequence h such that regardless of the starting state, the sequence of state labels observed during the execution of h uniquely determines the state reached.

The problem that arises when trying to adapt the notion of a homing sequence to a PSA M is that now the state labels (adversary actions) observed are no longer deterministic, but are generated by biased coin flips. Hence, given any action sequence h and state q of M, there is not a unique sequence of actions played by M when h is executed starting from q, but rather there is a *distribution* on such sequences. In order to overcome this obstacle we first modify the definition of a homing sequence as if we were actually able to see the bias of the coin in each state as we passed through it. We say that a sequence $h = h_1 \ldots h_k$, $h_i \in \{0, 1\}$ is a homing sequence for the PSA M if for every pair of states q_1 and q_2 in M, if $q_1\langle h\rangle = q_2\langle h\rangle$, then $\tau(q_1, h) = \tau(q_2, h)$, where $q_i\langle h\rangle$ denotes the sequence of probabilities $\gamma(q_i), \gamma(\tau(q_i, h_1)), \ldots \gamma(\tau(q_i, h_1 \ldots h_{k-1}))$. It is not hard to verify (under Assumptions (1) and (2) above) that every PSA has a homing sequence of length at most n^2. Unfortunately, the algorithm can not directly observe the coin biases in the states. However, we now describe a procedure that, given an arbitrary sequence of actions α, executes α a polynomial number of times, and computes (with high probability) a good approximation to the sequence of coin biases in the states passed on *the last* execution of α. This provides us with a sufficient orientation procedure for PSA's, since if we know a homing sequence h, we can run the procedure with $\alpha = h$. If we do not know a homing sequence, we initially assume that some single-action sequence h is already a homing sequence. By testing the payoff obtained from the algorithm by using h, we either succeed in obtaining near-optimal payoff (in which case we may not discover a good approximation of M in the sense described above, but it does not matter in terms of the game payoff), or we are able to improve our candidate homing sequence (details omitted). This latter event can happen only $n - 1$ times before h is a true homing sequence.

To estimate the coin biases encountered on the last execution of α, we use an idea of Dean et al. [1] in the related setting of learning DFA's with noisy outputs [4]. Assume we execute α, m consecutive times, and we are interested in computing the sequence of state coin biases passed on the mth execution of α. Let the state reached after the ith execution of α be denoted by q^i. If $m > n$, then it is not hard to verify that the states $q^{m-n}, \ldots, q^m$ lie a cycle (separated by executions of α). More precisely, there exists some minimal period $1 \leq \pi \leq n$ such that every π executions of α there is an execution of α that starts from q^{m-1} (let us call this state q_s) and ends at q^m (let us call this state q_f). If we knew the correct period π, we could estimate $q_s\langle\alpha\rangle$ (the sequence of coin biases from q_s to q_f reached by executing α from q_s) by simply keeping statistics only every π executions of α, with the intervening $\pi - 1$ executions returning us to q_s.

In order to find a good approximation to $q_s\langle\alpha\rangle$ *without* knowing the correct period, for every pair $1 \leq \ell, \ell' \leq n$ we compute the statistics obtained by $assuming$ that $\ell \cdot \ell'$ is a period of α. More precisely, we first perform $\ell \cdot \ell' - 1$ executions of α simply to reach some state q_s. We then execute α again, but for each action we record whether M played 0 or 1. We then repeat this process, again making $\ell \cdot \ell' - 1$ executions of α and *assuming* we have returned to q_s, then executing α to record further statistics for the state coin biases following q_s. Let us denote the sequence of $|\alpha|$ averages obtained in this way by $\psi^{\ell,\ell'}$. If ℓ is a minimal period, then for every $1 \leq \ell' \leq n$, $\ell \cdot \ell'$ is also a period (though not minimal) and $\psi^{\ell,\ell'}$ should be approximately the same as $\psi^{\ell,1}$. If ℓ is *not* a period, then one possibility is that we shall discover it is incorrect since for some ℓ', $\psi^{\ell,\ell'}$ differs substantially from $\psi^{\ell,1}$. We next claim that if we do not discover such an inconsistency, then $\psi^{\ell,1}$ is a good approximation for $q_s\langle a\rangle$

[4] The algorithm of Dean et al. requires a *distinguishing sequence* for the automaton, which is a sequence of actions whose execution determines the *starting* state of the execution, rather than the destination state. It is known that some automata do *not* have a distinguishing sequence, even if we restrict to automata with small cover time.

even though ℓ is not a period: If for every ℓ', $\psi^{\ell,\ell'} \approx \psi^{\ell,1}$, then, in particular this is true for the *minimal* period $\ell' = \pi$. But we know that for the minimal period π, $\psi^{\ell,\pi} \approx \psi^{\pi,1}$, where $\psi^{\pi,1}$ is a good approximation of $q_s\langle\alpha\rangle$. The above discussion gives us a simple procedure for computing a good approximation of $q_s\langle\psi\rangle$, from which we can derive with high probability the exact sequence of corresponding probabilities under Assumption (2). □(Theorem 5.1)

5.2 Probabilistic Transition Automata

In this section we give a hardness results for the problem of playing games against *Probabilistic Transition Automata* (PTA's). In these automata, the state transition function is probabilistic, and thus the *path* taken through the automaton in response to a sequence of strategy learning algorithm actions is randomly distributed.

More formally, a PTA M, $M = (Q, q_0, \gamma, \tau)$ is the same as a PSA except that now the transition function τ is defined as follows: $\tau : Q \times \{0,1\} \times Q \to [0,1]$, where for every $q \in Q$, and for every $b \in \{0,1\}$, $\sum_{q' \in Q} \tau(q,b,q') = 1$. As is the case when M is a PSA, M is initially in its starting state q_0. If its state in round N is q, then the action it plays is 1 with probability $\gamma(q)$, and 0 with probability $1 - \gamma(q)$. If the action of the strategy learning algorithm at round N is $b \in \{0,1\}$ then the next state is chosen randomly according to the probability distribution defined by $\tau(q,b,\cdot)$. As before we let $n = |Q|$. It is worth noting that every automaton which has both probabilistic transitions and probabilistic actions can be transformed into an automaton of approximately the same size which has only probabilistic transitions.

Fortnow and Whang [2] show that there exist PTA's for which there is *no* optimal strategy, even for penny-matching (where a strategy g is *optimal* with respect to an adversary f if $\liminf_{N\to\infty} PAYOFF_f^N(g)$ is maximized). Papadimitriou and Tsitsiklis [7] show that computing the *exact* payoff of the optimal strategy (given that it exists) is $\mathcal{PSPACE}$-complete. Unfortunately, this result is not sufficient to dismiss the possibility of an efficient strategy learning algorithm, because such an algorithm must only discover an *approximately* optimal strategy. In the following theorem we claim that even approximating the optimal payoff for $N = poly(n)$ rounds within $\Omega(1/\sqrt{n})$ is $\mathcal{PSPACE}$-complete. As in the work of Papadimitriou and Tsitsiklis[7], we assume the automaton is given as input to the algorithm. Clearly, if the automaton is not given (as in the learning setting), the problem of learning an approximately optimal strategy is at least as hard.

Theorem 5.2 *For any given non-trivial game[5], the problem of approximating, within $\Omega(1/\sqrt{n})$, the optimal expected payoff for playing the game G against a given PTA M for $N = poly(n)$ rounds is $\mathcal{PSPACE}$-complete.*

[5] We say a game is *trivial* G if all the entries in its game matrix have the same value.

The proof of the theorem, which is ommited for lack of space, is based on a reduction from Arthur-Merlin games. The size of the automaton in the reduction is proportional to the number of bits sent between Arthur and Merlin. This raises the question of whether we can do better for small PTA's — namely, whether a good strategy can be found in time exponential in the size of the automaton but polynomial in $1/\epsilon$. We show in the full version of the paper that, given an automata, if the game is played for N rounds then such a strategy can be found efficiently (where the dependence of the running time on N is linear).

References

[1] Thomas Dean, Dana Angluin, Kenneth Basye, Sean Engelson, Leslie Kaelbling, Evangelos Kokkevis, and Oded Maron. Inferring finite automata with stochastic output functions and an application to map learning. *Machine Learning*, 18(1):81–108, January 1995.

[2] Lance Fortnow and Duke Whang. Optimality and domination in repeated games with bounded players. In *The 25th Annual ACM Symposium on Theory of Computing*, pages 741–749, 1994.

[3] Itzhak Gilboa and Dov Samet. Bounded versus unbounded rationality: The tyranny of the weak. *Games and Economic Behavior*, 1(3):213–221, 1989.

[4] Ehud Kalai. Bounded rationality and strategic complexity in repeated games. *Game Theory and Applications*, pages 131–157, 1990.

[5] N. Littlestone. Learning when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[6] Abraham Neyman. Bounded complexity justifies cooperation in the finitely repeated prisoners' dilemma. *Economics Letters*, 19:227–229, 1985.

[7] Christos H. Papadimitriou and John N. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441 – 450, 1987.

[8] Christos H. Papadimitriou and Mihalis Yannakakis. On complexity as bounded rationality. In *Proceedings of the Twenty Sixth Annual ACM Symposium on the Theory of Computing*, pages 726–733, 1994.

[9] Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103(2):299–347, 1993.

[10] Dana Ron and Ronitt Rubinfeld. Exactly learning automata with small cover time. In *Proceedings of the Eighth Annual ACM Conference on Computational Learning Theory*, pages 98–109, 1995.

[11] Dana Ron and Ronitt Rubinfeld. Learning fallible finite state automata. *Machine Learning*, 18:149–185, 1995.

[12] V. G. Vovk. An optimal-control application of two paradigms of on-line learning. In *Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory*, pages 98–109, 1994.