# 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
Hebrew University

Ronitt Rubinfeld
Cornell University

Robert E. Schapire
AT&T Bell Laboratories

April 1995

**Abstract**

We study the problem of efficiently learning to play a game optimally against an unknown adversary chosen from a computationally bounded class. We both contribute to the line of research on playing games against finite automata, and expand the scope of this research by considering new classes of adversaries. We introduce the natural notions of games against *recent history adversaries* (whose current action is determined by some simple boolean formula on the recent history of play), and games against *statistical adversaries* (whose current action is determined by some simple function of the statistics of the entire history of play). In both cases we give efficient algorithms for learning to play penny-matching and a more difficult game called *contract*. We also give the most powerful positive result to date for learning to play against finite automata, an efficient algorithm for learning to play any game against any finite automata with probabilistic actions and low cover time.

# 1 Introduction

In the game theory literature, there is an extensive line of interesting research on the problem of playing a repeated matrix game against an adversary whose computational resources are in some way limited. Perhaps the main way in which this line of research differs from classical work on game theory lies in the fact that when our adversary is not playing the minimax optimal mixed strategy for the game, we may be able to enjoy a payoff that is significantly greater than the minimax optimum. In such a case, 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 in this area has been 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 [5]). While most previous papers examine how various aspects of classical game theory change in this setting (for example, whether cooperation is a stable solution for prisoner's dilemma when the adversary is a finite automaton [8, 10]), some have examined the natural question of *learning* to play optimally against a computationally bounded adversary [4]. These works usually do not explicitly take into account the computational efficiency of the learning algorithm, and often give 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 [3].

In this paper, we seek to contribute to and expand the scope of this line of research. In particular, we examine the problem of efficiently learning to play optimally against classes of adversaries that are incomparable to the finite automata classes which have been primarily considered to date, and we also improve on known results for probabilistic finite automata adversaries. We are especially interested in providing algorithms for games *other* than *penny-matching* (in which payoff is received for matching the adversary's action in the current round), for which we might expect the immediate applicability of existing ideas from computational learning theory. We examine games and adversaries for which the learning algorithm's past actions may strongly affect the adversary's future willingness to cooperate, and must therefore be *planned* carefully. For example, in the game we call *contract*, we receive 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 $M$ — it suffices to discover a "penny-matching cycle" of $M$ [3]. For contract, while an exact model of $M$ may still be unnecessary, we must at least 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.

Since several of the problems we examine involve new models and new classes of adversaries, we defer a detailed description of our results until the main body of the paper, after we have defined the models. Here we give a brief summary. The paper and the results are divided into two main parts. Following the formal definitions in Section 2, in Section 3 we introduce two new classes of adversaries for contract. The first adversary 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 efficient algorithms for learning to play contract. For the first class, however, the strategy of penny-matching is fatal in terms of the contract payoff, while for the second class, it is a useful subroutine. In the second part of the paper in Section 4, we contribute to and improve upon work for probabilistic finite automata with probabilistic actions but deterministic transitions. Here we give what is perhaps the most powerful positive result to date for learning to play against such automata — an efficient algorithm for learning to play any game optimally against any finite automaton with probabilistic actions and low cover time. (For the case of randomized transitions, we improve the $\mathcal{PSPACE}$-hardness result of Papadimitriou and Tsitsiklis [9].)

## 2 Models and Definitions

In this paper, games are played by two players, one of which will be called the *adversary* and the other the *strategy learning algorithm*. We think of the adversary as being implemented by a resource-bounded computational mechanism. Thus the adversary may be playing a strategy quite different from the minimax optimal strategy for the game, for a variety of natural reasons. For instance, the computational restrictions on the adversary may actually prevent computation of the minimax optimal strategy. Alternatively, the adversary may believe that he can obtain higher payoff by cooperating with his opponent, or by exploiting weaknesses of his opponent. On the other side, the strategy learning algorithm will be a polynomial-time algorithm attempting to learn, from repeated plays, a profitable strategy for playing against the adversary. It is worth noting that once the adversary is a fixed but unknown computational device chosen from a known and restricted class, the goal of the strategy learning algorithm is essentially that of learning to *control* an unknown system. In this alternative view of our framework, the issue of the adversary's motivation for deviating from the minimax optimal strategy does not arise — the game and the adversary strategy together simply define an unknown system whose "payoff" we would like to maximize. This view will be developed further in the full paper.

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 $2 \times 2$ 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 indicated, in our model the adversary is chosen from a class $\mathcal{A}$ of adversaries, usually defined by some resource-bounded computational mechanism, 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 a current action. After $N$ rounds, the *payoff* of a strategy $g$ with respect to an 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 of the game matrix $G$ corresponding to $a$ being played by player $g$ and $b$ being played by player $f$. The *optimal strategy* for $N$ rounds against $f$, $opt_f^N$, is the strategy that maximizes $PAYOFF_f^N(g)$.

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

$$|PAYOFF_f^N(A) - PAYOFF_f^N(opt_f^N)| \leq \epsilon.$$

We call $N_0$ the *initialization time* of $A$ for the given $\epsilon$. If $A$ is $\epsilon$-good for every $\epsilon$, we say that $A$ is simply a *strategy learning algorithm* for playing $G$ against $\mathcal{A}$, and we say $A$ is an *efficient* strategy learning algorithm for playing $G$ against $\mathcal{A}$ if the initialization time is polynomial in $1/\epsilon$ and $size(f)$, and the computation time at any round $N$ is polynomial in $1/\epsilon$, $N$, and $size(f)$. As 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 automata, or the number of gates if $f$ is defined by a boolean circuit); it is natural to allow $A$ more computation to succeed against more complex adversaries. If the computation time required by $A$ at each round does not depend on the history length $N$, then $A$ is called *real-time*. If $A$ is a probabilistic algorithm then we require that it succeed only with probability $1 - \delta$ for a given confidence parameter $\delta$ (and, as usual, allow it to run in time polynomial in $\log(1/\delta)$). We will also have occasion to consider strategy learning algorithms $A$ that can only achieve a payoff that is a multiplicative factor $c < 1$ of the optimal; following the terminology of on-line algorithms, we call such algorithms *$1/c$-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. We will also often use the term penny-matching to refer not to this game itself, but to the *strategy* (regardless of the game being played) of always playing the same action as the adversary. The second game 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.

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 games types, especially in the context of resource-bounded adversaries. In penny-matching, regardless of the adversary's current action (which in the model given below 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 contrast, in contract, 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 current state, while contract may require the strategy learning algorithm to *influence* the adversary's state, possibly through careful planning over many rounds of play.

# 3 Learning to Play Contract

As discussed in the introduction, most of the previous work on playing games against computationally bounded adversaries has concentrated on adversaries defined by finite automata of various flavors, and there has been very little work on *learning* to play optimally except for the game of penny-matching. We seek to broaden research in this area on both fronts, by considering adversaries defined by other simple but natural computational devices, and by examining games other than penny-matching. Thus, in this section we investigate the problem of learning to play contract against two new classes of adversaries. The first type of adversary is given by certain simple boolean formulae over the recent history of play, and the second is defined by linear functions of certain 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 provide what we consider to be initial but suggestive results in a potentially rich area, and the algorithms and analyses highlight the fact that penny-matching as a tool for playing contract can be alternatively fatal or quite helpful, depending on the adversary.

## 3.1 Games Against Boolean Formulae Over the Recent History

We first consider games against adversaries whose current play is determined by a simple computation on the *recent* history of play. This is in contrast to adversaries defined by finite automata, or the statistical adversaries examined in Section 3.2, where the adversary's current play may depend, however weakly, on events that occurred indefinitely far back in the history of play. We begin by introducing the general framework of recent history adversaries, then specialize to the rather simple class that we analyze.

Let us introduce the *history vectors* of length $\ell$ at round $N$, $\vec{u}^N, \vec{v}^N \in \{0,1\}^\ell$, 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 $N-i$ ($1 \leq i \leq \ell$). 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 $\ell$ 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). The adversary's current play is then defined by some boolean function $f(\vec{u},\vec{v})$. We call such an adversary a *recent history* adversary. 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 we enjoyed two consecutive payoffs $i+1$ rounds ago. Thus the adversary defined by $f$ simply plays 1 if and only if we have not experienced two consecutive payoffs in the last $\ell$ rounds, a "profit-limiting" adversary.

As usual, we would like to efficiently learn to play nearly optimally with 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 even penny-matching against recent history adversaries that are circuits of size polynomial in the history length $\ell$, since such adversaries can compute pseudo-random functions of the history (details in the full paper). Similar arguments apply to 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 [7] — 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 $2\ell$-bit inputs, 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 optimally, since for $N$ sufficiently larger than $m(\ell, size(f))$ the fraction of rounds in which we have failed to match is rapidly diminishing. 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 dichotomy between penny-matching and contract for a particular class of recent history adversaries.

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

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

Note that $f_S$ is a 2-DNF formula over the variables $u_1, \ldots, u_n, v_1, \ldots, v_n$; since such formulae have an efficient absolute mistake-bounded algorithm [7], our comments above imply that learning to play penny-matching optimally against such adversaries is already a solved problem. 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 strategy learning algorithm played 0 and the adversary played 1. If this occurs for any $i$ in the index set $S$, the adversary plays 1 in the current round, otherwise the adversary plays 0. We call the $\{f_S\}$ the *past sacrifice* adversaries for contract, because $f_S$ will permit a payoff in the current round only if the strategy learning algorithm forfeited a payoff (deliberately or otherwise) in at least one of certain rounds (defined by $S$) in the recent past. (Formally, we use the term *sacrifice* to refer to rounds in which $f_S$ plays 1 and the strategy learning algorithm plays 0.) The past sacrifice adversaries are interesting for the direct tension they create between obtaining payoff in the current round and ensuring future payoffs, 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. In other words, penny-matching against $f_S$ is a disastrous policy if the game is contract.

Thus, while the formulae $f_S$ are trivial adversaries for playing penny-matching, different arguments and strategies are 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 2-competitive (that is, achieves $1/2$ the optimal payoff); the analysis shows that the optimal payoff has an interesting number-theoretic characterization. For simplicity we assume in the following that the initial configuration of the history is $u_1 = 0$ and $v_1 = 1$ (a sacrifice), and $u_i = 0$ and $v_i = 0$ for $i = 2, \ldots, \ell$; various complications arise for other initial configurations that we address in the full paper.

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

**Proof (Sketch):** The proposed strategy learning algorithm $A$ operates in two simple phases. In the first phase (called the *sacrifice phase*), $A$ 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*), $A$ 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 $A$. 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 sufficiently many 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 $A$ for playing against $f_S$, and let us credit $A$ with payoff 1 at the current round if $f_S$ plays 1, regardless of whether $A$ has played 0 or 1. Then we will only overcount the actual payoff of $A$, and now we can assume without loss of generality that $A$ plays to maximize the frequency with which $f_S$ plays 1. Since $A$'s playing 1 in the current round can never cause $f_S$ to play 1 in a future round, the best strategy for $A$ 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. In general, if $f_S$ plays 1 at round $N$, this inserts another sacrifice into the history (since $A$ 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 any natural number coefficients $a_1, \ldots, a_r$. Thus $N$ 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 the strategy of always playing 0 against $f_S$ will in fact eventually 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). We now wish to show that $N + g, N + 2g, \ldots, N + xg$ are also sacrifice rounds, where $i_1 = xg$; 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, 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 $a_i \geq |xc_i|$, the coefficients of $N + xg$ will also be natural numbers, and $N + xg$ will be a sacrifice round, as will $N + g, N + 2g, \ldots$. □

Thus, the sacrifice phase of our strategy learning algorithm $A$ continues until the history vectors are *$g$-flooded* with sacrifices — that is, they contain a sacrifice every $g$ indices ($A$ can quickly infer the value of $g$ from the pattern of sacrifices that emerges). At this point, $A$ attempts to enter its payoff phase. Since $A$'s behavior on rounds that are not $g$-rounds is irrelevant, we concentrate on $A$'s behavior on $g$-rounds. Of course, $A$ 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 $i_1$: we claim that in this case only a single payoff can ever be obtained, so the average payoff over time approaches 0 (the reader can verify this point). Thus without loss of generality $|S| \geq 2$.

First suppose that we know the first two indices $i_1, i_2$, and let $i_2 - i_1 = kg$ for a natural number $k \geq 1$. Consider the strategy of restoring sacrifices for the next $k$ $g$-rounds, then obtaining payoff for the following $k$ $g$-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 $k$ $g$-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.

The only remaining problem is that we do not know $i_1$ and $i_2$ (or more to the point, we do not know $k$). This is handled by trying values of $k$ in the above alternating sacrifice-payoff pattern, starting with $k = 1$. If we are able to maintain the pattern using block size $k$, we are obtaining payoff $1/(2g)$ and we do not care if we have the correct value for $k$. The only way we can fail is if $f_S$ plays 0 at some $g$-round, in which case we know we have a value for $k$ that is too small. However, it is easily verified that at this point there must be a sacrifice in the history vectors at an index smaller than $i_2$ (because we are approaching the correct value of $k$ from below). Thus $A$ will reflood the history vectors with sacrifices and try the next value of $k$. □

It would be nice to generalize this result to obtain an efficient contract strategy learning algorithm for the entire class of 2-DNF recent history adversaries, but the past sacrifice special case already suffices to demonstrate the significant differences with penny-matching, and also that number theoretic analyses must arise in the more general case. We now turn our attention to another class of contract adversaries — one for which penny-matching is sometimes a useful subroutine, but for somewhat novel reasons.

## 3.2 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, but in a simple and natural way: namely, the adversary's move depends only on the *statistics* of the history. Here we consider the most basic 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 rounds (in the first $N$) in which the strategy learning algorithm played $a$ and the adversary played $b$. These fractions are updated in the obvious way to obtain the $\{p_{ab}^{N+1}\}$ following each round. 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 the strategy learning algorithm receive a payoff only if it is not too wealthy already, a rather natural strategy.

In general, one can consider adversaries whose current play is a complicated function of not only the $p_{ab}$ but of more elaborate statistics of the history. As a start, here we give 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. Thus we are revisiting 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 involving only $p_{00}$ 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 for linear separators in $\Re^2$ do not exist (a point elaborated upon in the proof), even playing penny-matching is nontrivial here. Since our algorithms for contract in fact involve penny-matching subroutines, we solve the penny-matching problem as well. As we shall demonstrate, the reason that penny-matching is helpful here (in contrast to playing contract against the past sacrifice strategies) is that for some $f_{\alpha,\beta}$, penny-matching moves us on a trajectory through the $p_{00} - p_{11}$ plane that rapidly approaches the optimal point in that plane, and furthermore remains stably at the optimal point. Thus in some cases penny matching, without explicitly attempting to maximize the contract payoff, can be regarded as an effective means of motion in the plane.

**Theorem 3.3** *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 contract against the adversary class $\{f_{\alpha,\beta} : \alpha, \beta \in \Re\}$.*

**Proof (Sketch):** For simplicity, we restrict our attention to the case $\alpha > 0$, for which a representative picture is shown in Figure 1. (The general $\alpha$ case will be considered in the full paper; the ideas are similar but some care is required in the handling of initial conditions.) The main ideas of the proof are: (1) For $\alpha > 0$, penny-matching (that is, always playing identically to the adversary $f_{\alpha,\beta}$) is an optimal contract strategy [2]; (2) Despite the fact that the well-known halving algorithm can be forced to make many mistakes of prediction 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, due to the limited rational complexity (defined below) of the inputs $p_{00}$ and $p_{11}$; and (3) An analysis of the contract payoff of the halving algorithm shows that the logarithmic number of predictions mistakes have a small effect.

For concreteness we assume in the following analysis that the initial conditions are $p_{00} = p_{11} = 0$. We begin by considering the contract payoff when playing 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 $\Theta$, and that this optimal can be achieved by always playing 1, and also by penny-matching against $f_\Theta$. Here we are interested in analyzing the contract payoff of the halving algorithm $A_1$ that *attempts* to penny-match against $f_\Theta$ (imperfectly, since $\Theta$ is unknown), since the analysis of $A_1$ includes most of the elements required to solve the two-dimensional case.

At every round $N$, $A_1$ keeps track of the *version space* $[\Theta_L, \Theta_R)$ for $\Theta$. Here $\Theta_L$ is the largest past value of $p_{11}$ for which the adversary played 1, and $\Theta_R$ is the smallest past value of $p_{11}$ for which the adversary played 0. Thus at all times $\Theta$ lies in the version space, which represents $A_1$'s range of uncertainty about $\Theta$. At the current round, $A_1$ plays 1 if and only if $p_{11} < (\Theta_L + \Theta_R)/2$. If $A_1$'s play fails to match that of $f_\Theta$, then the width of the version space has been at least halved.

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

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

In the usual models of on-line prediction [7], the sequence of inputs $x_i$ to $f_\Theta$ to be classified (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). Here, however, the situation is considerably more favorable. Let us say that $p \in [0,1]$ 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 $N$, $\Theta_L$ and $\Theta_R$ 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 $\Theta_R - \Theta_L > 1/N^2$ — in other words, there is a nice lower bound on the version space width. Thus if $m$ is the number of prediction mistakes made by $A_1$ on the first $N$ rounds, we must have $1/2^m > 1/N^2$, or $m < 2\log N$. We now argue that the few prediction mistakes made by $A_1$ have a rather small effect on the contract payoff that $A_1$ receives in comparison to penny-matching.

**Lemma 3.4** *For any $\epsilon > 0$, the contract payoff of algorithm $A_1$ when playing against $f_\Theta$ is at least $\Theta - \epsilon$ within $O((1/\epsilon)\log(1/\epsilon))$ rounds.*

**Proof:**We analyze the behavior of $p_{11}$ when $A_1$ plays against $f_\Theta$. First, for any small $\epsilon > 0$, we bound the number of rounds required for $p_{11}$ to exceed $\Theta - \epsilon/2$ for the first time. Since $A_1$ makes at most $2\log N$ prediction mistakes in the first $N$ rounds, and since $f_\Theta$ 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 $p_{11} \geq 1 - (2\log N)/N \geq \Theta - \epsilon/2$ will certainly hold for $N = O((1/\epsilon)\log(1/\epsilon))$.

Next we argue that if $p_{11}$ falls below $\Theta - \epsilon/2$ immediately following some round $N$, then by 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 $2N$, $A_1$ 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 $A_1$ increases $p_{11}$ by at least $(1 - (\Theta - \epsilon/2))/2N \geq \epsilon/4N$; this is because an increase to $p_{11}$ after round $N$ looks like $p_{11} \leftarrow (N/(N+1))p_{11} + 1/(N+1)$, or $p_{11} \leftarrow p_{11} + (1 - p_{11})/(N+1)$. Thus the total increase to $p_{11}$ between rounds $N$ and $2N$ is at least $(N - 2\log(2N))(\epsilon/4N)$. If we satisfy $(N - 2\log(2N))(\epsilon/4N) > (2\log(2N))/N$ 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 $\epsilon$ of $\Theta$ for all rounds in between $N$ and $2N$. Again, $N = O((1/\epsilon)\log(1/\epsilon))$ suffices. □

Armed with the analysis of algorithm $A_1$, we now return to the more general 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 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 optimal (see Figure 2 for a typical trajectory taken by penny-matching), and also that the naive strategy of always playing 1 (which *did* achieve the optimal against the one-dimensional $f_\Theta$) will *not* succeed (similarly for always playing 0).

Thus, our goal will again be that of approximately penny matching, this time using the halving algorithm $A_2$ 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 $A_2$. However, the basic proof outline is similar. It can be shown that the version space, which is the region in the $\alpha - \beta$ plane of parameters consistent with the history so far, is always a convex polygon (see Figure 3). Furthermore, the vertices of this polygon, while no longer equal to past values of $p_{00}$ and $p_{11}$, are nevertheless determined by past values, and have rational complexity at most $N$; this implies a $1/p(N)$ lower bound on the version space area at round $N$ for some polynomial $p(N)$, leading again to an $O(\log N)$ bound on the number of prediction mistakes by $A_2$ (details in the full paper). For the efficiency of $A_2$, note that computing the area of a convex polygon in the plane can be done in time polynomial in the number of vertices.

This analysis implies that when $A_2$ 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 $A_2$ 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 3.4 shows that $A_2$ will converge rapidly to this optimal configuration. □

As a final example of learning to play contract against a statistical adversary, we consider the more complicated case in which the the adversary's 1-region is an intersection of $d$ half-spaces, rather than a single half-space as in the functions $f_{\alpha,\beta}$. Again for simplicity, we assume here that all of the half-spaces are defined by lines with positive slope in the $p_{00} - p_{11}$ plane; a representative adversary is shown in Figure 4.

**Theorem 3.5** *There is an efficient strategy learning algorithm for contract against any intersection of $d$ half-spaces defined by lines of positive slope in the $p_{00} - p_{11}$ plane.*

**Proof (Sketch):** 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 penny-matching will rapidly converge to this optimal. However, now the version space has $2d$ parameters and the volume computation required for implementing the halving algorithm directly has no known polynomial time solution (in particular, the version space is no longer 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 attempts to maintain a "local" version space for a *line* using only those examples $\langle (p_{00}, p_{11}), f(p_{00}, p_{11}) \rangle$ seen so far for which $p_{11} \in [a, b)$ (see Figure 5). Notice that if the strip of the $p_{00} - p_{11}$ plane defined by the constraint $p_{11} \in [a, b)$ is intersected by only one of the lines defining $f$, we can always maintain such a version space for $[a, b)$. Otherwise, at some point the examples falling into the subinterval may become inconsistent with any linear classifier (that is, the version space for $[a, b)$ 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 attempt to define the version spaces for these two new subintervals using the examples seen so far that fall into each. For at least one of the new subintervals, the version space must be non-empty, since only the subinterval containing the last example can have an empty version space (recall that prior to the last example, the entire subinterval $[a, b)$ had a non-empty version space). As long as a new subinterval has an empty 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 non-empty version space, in the worst case we divide $[a, b)$ into three new subintervals $[a, c), [c, d), [d, b)$ with non-empty version spaces.

The main point is that each time a 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 this version space became empty, and now this corner lies in a subinterval of length at most $(b - a)/2$). Furthermore, since the past 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 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 reach the optimal point for playing against $f$ rapidly. □

# 4 Games Against Probabilistic Finite Automata

In this section we consider algorithms for playing against adversaries defined by *probabilistic finite automata*. This continues a line of research investigated by Gilboa and Samet [4], and most recently by Fortnow and Whang [3]. We describe an efficient algorithm for learning to play any game optimally against any finite automaton with probabilistic actions and low cover time. In Appendix A we show that if the adversary automaton has probabilistic transitions then the problem of approximating the optimal strategy is $\mathcal{PSPACE}$-complete even if the automaton is known to the algorithm, thus improving on a result of Papadimitriou and Tsitsiklis [9].

In the setting studied here, the action played by the adversary automaton is determined according to a probability distribution over the actions $\{0, 1\}$ associated with the current state — thus, each state of the adversary automaton can be thought of as containing a different biased coin. Given the action of the strategy learning algorithm, the next state of the automaton is determined by its (deterministic) state transition function. More formally, a *Probabilistic Action Automaton* (*PAA*) $M$ is a tuple $(Q, q_0, \gamma, \tau)$, where $Q$ is a finite set of $n$ *state*, $q_0 \in Q$ is the designated *starting state*, $\gamma : Q \to [0, 1]$, is the *state action probability function*, and $\tau : Q \times \{0, 1\} \to Q$ is the *transition function*. The adversary automaton $M$ is initially at its

starting state, $q_0$. If its state in round $N$ is $q_N$, then the action it plays is 1 with probability $\gamma(q_N)$, and 0 with probability $1-\gamma(q_N)$. If the action of the learning algorithm in the $N$th round is $\alpha_N \in \{0,1\}$, then $M$'s state in round $N+1$ is $\tau(q_N, \alpha_N)$.

An important property of $PAA$'s is that similarly to deterministic finite automata ($DFA$s) they have a well-defined notion of *cycles*. A *cycle* on a $PAA$ $M$ is determined by a state $q \in Q$, and a sequence of actions $\alpha = \alpha_1\alpha_2\cdots\alpha_\ell$ such that $\tau(q, \alpha_1\alpha_2\ldots\alpha_\ell) = q$. A cycle is *simple* if no state is visited more than once on the cycle. As in the case of $DFA$s, for every $PAA$ there exists an optimal strategy that is a repeated simple cycle [4]. Moreover, if the automaton is given, such a cycle can be found efficiently using dynamic programming. In the learning setting, Gilboa and Samet [4] note that their algorithm for learning an optimal cycle strategy against $DFA$s can be easily adapted to $PAA$'s. However, as in their original strategy for $DFA$s it might take an exponential number of rounds (in the number of states of the adversary automaton) to achieve payoff which is close to optimal. Fortnow and Whang [3] show that 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. They use a subclass of $DFA$s, which are sometimes referred to as *combination lock* automata, to show that there exists a constant $\epsilon$ such that for every strategy learning algorithm there exists a deterministic automaton for which the strategy learning algorithm achieves payoff $\epsilon$-close to optimal only after an exponential number of rounds.

It is interesting to note that if the underlying game is penny-matching then this last statement is not true. Fortnow and Whang give an efficient strategy learning algorithm for penny-matching against $DFA$s. Unfortunately, it does not seem that penny-matching is any easier than contract when playing against $PAA$'s. The combination lock argument given by Fortnow and Whang can easily be modified to give an exponential lower bound for learning to penny-match against $PAA$'s. The key property of combination lock automata used in this argument is that they have a state which can be reached only if a specific sequence of $n-1$ actions is played by the learning algorithm. For this hard-to-reach 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 if we can do better if we restrict our attention to automata which do not have states that are hard to reach — namely, to automata whose underlying graphs have small (polynomial) cover time[3]. We answer this question in the affirmative in the theorem below.

**Theorem 4.1** *For any game $G$, there is an efficient strategy learning algorithm for playing $G$ against the class of probabilistic action automata with polynomial cover time.*

Due to space limitations we shall not give the full details of the algorithm and its proof. Instead, we sketch the main ideas of the algorithm, referencing standard methods from the literature where needed and highlighting the points where new ideas are required.

## 4.1 The Strategy Learning Algorithm for $PAA$'s: Sketch of Main Ideas

Let $M$ denote the adversary $PAA$ with low cover time. In what follows we describe an efficient learning algorithm that with high probability constructs a *good approximation* $\widehat{M}$ of $M$ after playing a polynomial number of rounds against $M$. The automaton $\widehat{M}$ is a good approximation of $M$ in the sense that for any sequence of actions, if the sequence is played against $\widehat{M}$, then the probability distribution on $\widehat{M}$'s next action is very similar to the probability distribution on $M$'s next action had the sequence been played against $M$. Thus, if we play against $M$ according to an optimal cycle computed for $\widehat{M}$, then our payoff is close to optimal. It should be noted that the quality of approximation needed may depend on the game $G$ being played (in our description below, we assume for simplicity it is either penny-matching or contract). For example, if some game gives extremely high payoff $H$ for some pair of actions $\alpha_1$ of the adversary and $\alpha_2$ of the strategy learning algorithm, then an optimal cycle might include a single state which plays $\alpha_1$ with extremely small probability, and thus this probability should be approximated to within $1/H$.

In order to simplify the presentation we start by making two assumptions which can be removed using a technique described in [13] (details in the full paper). (1) There exists some value $\Delta = poly(\epsilon/n)$, such

[3] The cover time of a graph is defined to be the smallest integer $\ell$ such that for every vertex $v$, a random walk of length $\ell$ starting from $v$, will pass through every other vertex with probability greater than one half.

that for every state $q$ in the adversary automaton $M$, $\gamma(q) = k\Delta$, for some $1 \leq k \leq 1/\Delta$ (Thus, there might exist states which have the exact same biased coin, but any two that differ, differ significantly); (2) $M$ is irreducible in the sense that there is no smaller automaton equivalent to it.

A central and typical problem that arises when trying to learn automata from one long sequence of experimentation is the need for a procedure that *orients* the learner by providing evidence that it has returned to a state it has previously visited. Given such an orientation procedure, the *PAA* learning problem reduces [11] to what we call *PAA learning with reset*. Under assumption (1) above (as well as the low cover time assumption) the latter problem can be solved using a variant of Angluin's algorithm [1] (for learning *DFA*s with reset) that is similar to the one described in [12]; thus we concentrate here on the issue of orientation. A well-studied procedure [11] for orientation when learning *DFA*s uses a *homing sequence* [6]. A homing sequence 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 by executing $h$.

The obvious problem that arises when trying to adapt the notion of a homing sequence to the *PAA* $M$ is that now the state labels (adversary actions) observed are no longer deterministic, so that for *no* action sequence performed by the strategy learning algorithm can the sequence of adversary actions observed uniquely determine the final state reached. 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.

**Definition 1** *A sequence $h = h_1 \ldots h_k$, $h_i \in \{0,1\}$ is a* homing sequence *for the PAA $M$ 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 $\langle\gamma(q_i), \gamma(\tau(q_i,h_1)), \ldots \gamma(\tau(q_i,h_1\ldots h_{k-1}))\rangle$.*

It is not hard to verify that (under assumptions (1) and (2)) every *PAA* has a homing sequence of length at most $n^2$. Unfortunately, the algorithm can not directly observe the biases in the states. However, we now describe a procedure that, given an arbitrary sequence of actions $\alpha$, executes $\alpha$ a polynomial number of times, and computes (with high probability) a good approximation of the sequence of probabilities (biases) corresponding to the states passed on *the last* execution of $\alpha$. This provides us with a sufficient orientation procedure for *PAA*s. We hope that this procedure may find other applications as well.

The idea we employ follows a technique used by Dean *et. al.* [2] in the related setting of learning *DFA*s with noisy outputs. However, in order for their technique to work, their algorithm must be given a *distinguishing sequence* for the automaton.[4] Assume we execute $\alpha$ for $m$ consecutive times, and we are interested in computing the sequence of state probabilities passed on the $m$th execution of $\alpha$. Let the state reached after the $i$th execution of $\alpha$ be denoted by $q^i$. If $m > n$, then it is not hard to verify that the states $q^{m-n}, \ldots, q^m$ follow a cycle. More precisely, there exists some minimal *period* $\pi$, where $1 \leq \pi \leq n$, such that for every $m - \pi + 1 \leq i \leq m$, $q^i = q^{i-k\pi}$ for every $1 \leq k \leq \lfloor (i-n)/\pi \rfloor$. In particular, every $\pi$ executions of $\alpha$ there is an execution of $\alpha$ that starts from $q_s = q^{m-1}$ and ends at $q_f = q^m$. Let $o^i = o^i_1 \ldots o^i_{|\alpha|}$ be the action sequence played by $M$ on the $i$th execution of $\alpha$. Had we known the correct period $\pi$ we could estimate with high precision the sequence of probabilities corresponding to the execution of $\alpha$ starting from $q_s$ (and ending at $q_f$), simply by computing for every $1 \leq j \leq |\alpha|$ the average of the $o^i_j$'s for the corresponding indices $i$.

In order to find a good approximation for the sequence of probabilities $q_s\langle\alpha\rangle$ *without* knowing the correct period, we make the following observation. Let $1 \leq \ell, \ell' \leq n$, and for every pair, $\ell, \ell'$, let $\psi^{\ell,\ell'} = \psi^{\ell,\ell'}_1 \ldots \psi^{\ell,\ell'}_{|\alpha|}$ be the sequence of averages computed under the assumption that $\ell \cdot \ell'$ is a period. If $\ell$ 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 $\ell$ is *not* a period, then one possibility is that we shall discover it is incorrect since for some $\ell' a$, $\psi^{\ell,\ell'}$ 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 $q_s\langle a\rangle$ even though $\ell$ is not a period: If for every $\ell'$, $\psi^{\ell,\ell'} \approx \psi^{\ell,1}$, then, in particular this is true for the *minimal* period $\ell' = \pi$. But we know that for the minimal period $\pi$, $\psi^{\ell,\pi} \approx \psi^{\pi,1}$, 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 (1).

[4] A distinguishing sequence 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, and this remains true if we restrict our attention to automata with small cover time.

We finally must deal with the fact that the strategy learning algorithm does not have a homing sequence. This can be handled by some technical variations on known methods. The main idea is to 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, or are able to improve our candidate homing sequence. This latter event can happen only $n-1$ times before $h$ is a true homing sequence. (Details will be provided in the full paper.)

# 5 Conclusions and Open Problems

# References


[1] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, November 1987.

[2] 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.

[3] 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.

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

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

[6] Zvi Kohavi. *Switching and Finite Automata Theory*. McGraw-Hill, second edition, 1978.

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

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

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

[10] 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.

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

[12] Dana Ron and Ronitt Rubinfeld. Exactly learning automata with small cover time. 1995. To appear in Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory.

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

[14] Adi Shamir. IP = PSPACE. In *Proceedings of the Thirty First Annual Symposium on Foundations of Computer Science*, 1990.

# A Probabilistic Transitions: A Hardness Result

In this appendix we consider the problem of playing against *Probabilistic Transitions Automata* ($PTA$). In these automata the state transition function is probabilistic and thus the path taken through the automaton on a sequence of actions is randomly distributed according to some distribution. For simplicity we assume that the actions associated with the states are also chosen probabilisticly.[5]

More formally, A $PTA$ $M$, $M = (Q, q_0, \gamma, \tau)$ is defined similarly to a $PAA$ except that the transition function, $\tau$, is defined as follows: $\tau : Q \times \{0,1\} \times Q \to [0,1]$, where for every $q \in Q$, and for every $b \in \{0,1\}$, $\sum_{q' \in Q} \tau_2(q, b, q') = 1$. As is the case when $M$ is a $PAA$, $M$ is initially at its starting state, $q_0$. If its state in round $N$ is $q_N$, then the action it plays is 1 with probability $\gamma(q_N)$, and 0 with probability $1 - \gamma(q_N)$. If the action of the strategy learning algorithm at round $N$ is $\alpha_N \in \{0,1\}$ then the next state is chosen randomly according to the probability distribution defined by $\tau(q_N, \alpha_N, \cdot)$. As before let $n = |Q|$.

Fortnow and Whang [3] show that there exist $PTA$s for which there is *no* optimal strategy, even for penny-matching. Papadimitriou and Tsitsiklis [9] show that if such a strategy exists, then computing it *exactly* given the automaton is $\mathcal{PSPACE}$-complete. Unfortunately this result is not sufficient to dismiss the possibility of an efficient strategy learning algorithm, which must only discover an approximately optimal strategy. In the following theorem we claim that even finding a strategy whose expected payoff is $1/poly(n)$ smaller than the optimal payoff is $\mathcal{PSPACE}$-complete. As in [9], 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 is at least as hard.

**Theorem A.1** *For any given non-trivial game $G$,[6] the problem of finding a strategy whose expected payoff after every $N \geq n$ rounds is $1/poly(n)$ close to the optimal expected payoff for playing the game $G$ against a given $PTA$ $M$ is $\mathcal{PSPACE}$-complete.*

The proof of the theorem, which is given below, 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).

**Proof of Theorem A.1:** Let $L$ be a language in $\mathcal{PSPACE}$ defined over binary strings of length $\ell$. Shamir [14] proved that every language in $\mathcal{PSPACE}$ has an Arthur-Merlin proof protocol. An Arthur-Merlin proof protocol is a public coin interactive proof protocol between a randomized polynomial time Verifier, $V$, and an all-powerful Prover $P$, where for a given $x$, $x \in \{0,1\}^{\ell}$, $P$ tries to convince $V$ that $x \in L$. The protocol consists of $q = poly(n)$ rounds, where in each round $V$ sends a single random bit $r_i$ to $P$, and $P$ returns a single bit $y_i$ to $V$. Following this interaction, $V$ decides if $x$ belongs to $L$ or not. If $x \in L$, then there exists a prover which will cause $V$ to accept with probability at least $2/3$, and if $x \notin L$, then for *any* prover, $V$ will accept with probability at most $1/3$. The advantage of $V$ can easily be amplified to make his uncertainly exponentially small by repeating the protocol and choosing the majority answer.

The following observations has been made in previous works about interactive proof systems (e.g., [**?**]). Since the decision process $V$ employs must run in polynomial time, we may describe it as a circuit $C_V$ of polynomial size whose inputs are $x, r_1, \ldots, r_q$ and $y_1, \ldots, y_q$. Consider a slightly modified protocol in which, following the interaction between $V$ and $P$, $P$ also send $V$ the setting of *all* the internal lines in $C_V$. $V$ can then verify that both the final output is *accept*, and that for each of the gates in $C_V$, the input lines are consistent with the output lines. If any inconsistency is found, it outputs *reject*. The verifier thus performs a polynomial number of tests, each of constant complexity. Clearly, the new protocol has the same properties as the original protocol.

For any given $L \in \mathcal{PSPACE}$ with a verification circuit $C_V$, and for any $x \in \{0,1\}^{\ell}$, we next show how to construct a $PTA$ $M_{L,x}$, such that if $x \in L$ then there exists an optimal strategy against $M_{L,x}$ for penny matching with payoff exponentially close to 1, and if $x \notin L$ then any strategy against $M_{L,x}$ has payoff at

[5] It is easily verified 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.

[6] we say a game is *trivial* if all the entries in its game matrix have the same value

most $1-\epsilon$ where $\epsilon = \Omega(1/|C_V|)$. It directly follows that finding a strategy whose expected payoff is within $\epsilon$ of the optimal is $\mathcal{PSPACE}$ hard. If the game is not penny-matching then the construction can be modified in a straightforward manner.

The automaton we construct can be viewed as simulating the verifier's side of the protocol (i.e., tossing public coins) while the learning algorithm plays the role of the prover. If $m$ is the total number of bits sent in the protocol, then the first $m$ rounds of play basically constitute a transcript of the protocol. Following these $m$ rounds the automaton decides if to accept and allow the learning algorithm high expected payoff, or to reject and play so that the expected payoff is minimized. Since the automaton is very restricted computationally, it cannot "remember" the complete transcript of the protocol and verify that it should lead to acceptance. However, it can randomly (and privately) choose a gate in $C_V$ and verify that the bits in the protocol corresponding to lines connected to this gate are consistent with the function computed by the gate. Though the automaton may accept an input $x$ not in $L$ with quite high probability, there is still a gap of $\Omega(1/|C_V|)$ between the probability that it accepts if $x \in L$ and the probability that it accepts if $x \notin L$.

We start with the rough idea of the construction. In the first round played against $M_{L,x}$, independently of the action played, a random gate in $C_V$ is chosen, but its identity is not revealed to the learning algorithm. For any chosen gate $g_i \in C_V$, the automaton simulates the complete protocol between $V$ and $P$ (in a sense described below), but "remembers" only the final output bit of $C_V$, and the 3 bits in the protocol that correspond to the lines connected to $g_i$. If the 3 bits are consistent with the function computed by $g_i$ and the final output of $C_V$ is 1, then the automaton enters an *Accept* state. The action played by the automaton in this state is always 1, and its outgoing transitions are deterministic and form a self-loop. If the 3 bits are not consistent with the function computed by $g_i$ then the automaton enters a *Reject* state whose action function is completely random (and its outgoing transitions also form a self loop). Clearly, the only way to get payoff exponentially close to 1 is to reach an *Accept* state and penny match forever. It will not be hard to verify that the construction is such that there exists a one-to-one mapping between play strategies and prover strategies. If $x \in L$ then there exists a play strategy based on an the prover's strategy in an Arthur-Merlin protocol that with extremely high probability reaches an *Accept* state no matter which gate is chosen, and thus its expected payoff is as required. However, if $x \notin L$, then any play strategy that sets all circuit lines consistently has expected payoff $\approx 1/2$, and any play strategy that sends inconsistent values for some gate has expected payoff $1 - \Omega(1/|C_V|)$.

More formally, let $q_0$ be the starting state of $M_{L,x}$, and let $\gamma(q_0) = 1/2$. For each gate $g_i$ let there be a state $A_i^0$, where for each $A_i^0$, $\gamma(A_i^0) = 1/2$, and $\tau(q_0, 0, A_i^0) = \tau(q_0, 1, A_i^0) = 1/|C_V|$. We next concentrate our attention on some single gate $g_i$. Assume for now that we only want to simulate the protocol between $V$ and $P$. We later describe how the automaton "remembers" the relevant bits and verifies consistency. For the $j$th bit in the protocol, we have two states $A_i^{j,1}$ and $A_i^{j,0}$. where $\gamma(A_i^{j,1}) = 1$, and $\gamma(A_i^{j,0}) = 0$. These states hold the information if the $j$th bit played was 1 or 0. If the $j$th bit in the protocol is sent by $V$, then for $b, b' \in \{0,1\}$, $\tau(A_i^{j-1,b}, b', A_i^{j,1}) = \tau(A_i^{j-1,b}, b', A_i^{j,0}) = 1/2$, and if the $j$th bit is sent by $P$ then for $b, b' \in \{0,1\}$ $\tau(A_i^{j-1,b}, b', A_i^{j,b'}) = 1$. Thus, if the $j$th bit should be random, it is chosen randomly on the $j-1$ round, but is viewed only on the $j$th round, and if it is determined by $P$, then it is chosen in round $j-1$ and viewed again in round $j$. The seemingly lack of synchronization is due to the restriction that the action played by a state can be viewed only after leaving the state.

In the description above, the automaton only remembers the last bit of the protocol. Since it needs to remember 3 bits, (in addition to the last, answer bit) it does the following. Let the 3 bits be $j_1 < j_2 < j_3$, and consider first $j_1$. For every index $j < j_1$, all transitions (and action probability functions) are as described above. At this point, instead of "forgetting" the value of the $j_1$ bit, the automaton splits into two separate *paths*, one going out of $A_i^{j_1,0}$ and "remembering" that the bit was 0, and one going out of $A_i^{j_1,1}$ and "remembering" that the bit was 1. In the 0 path, and until $j_2$ is reached, we have copies $A_i^{j,1}(0)$ and $A_i^{j,0}(0)$ of $A_i^{j,1}$ and $A_i^{j,0}$, respectively, and in the 1 path we have copies $A_i^{j,1}(1)$ and $A_i^{j,0}(1)$. Transitions are only inside the paths (from $A_i^{j-1,1}(b)$ and $A_i^{j-1,0}(b)$ to $A_i^{j,1}(b)$ and $A_i^{j,0}(b)$) and are as described above for the"memoryless" transitions. When $j_2$ is reached we split again – forming 4 paths, one for each of the 4 possible settings of the 2 bits, and when reaching $j_3$ one more split occurs and the setting of the 3 bits is determined. When the final, $L$th bit is reached, if the path corresponds to an inconsistent setting then we enter the *Reject* state where $\gamma(Reject) = 1/2$ and $\tau(Reject, 0, Reject) = \tau(Reject, 1, Reject) = 1$, and if the setting is consistent and the last (output) bit is 1, then we enter an *Accept* state, where $\gamma(Accept) = 1$ and

$\tau(Accept, 0, Accept) = \tau(Accept, 1, Accept) = 1.$

# B Figures for Section 3.2

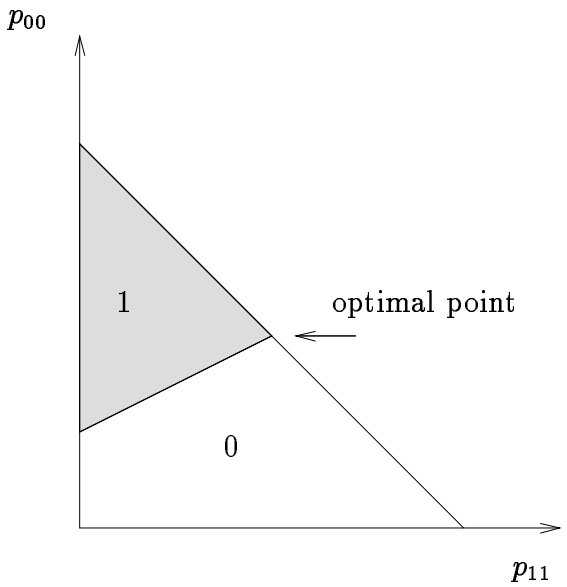


Figure 1: Example of a statistical adversary $f_{\alpha,\beta}(p_{00}, p_{11})$, showing the regions in which the adversary plays 0 and 1, the diagonal line representing the constraint $p_{00} + p_{11} \leq 1$, and the optimal point of play lying on this line.

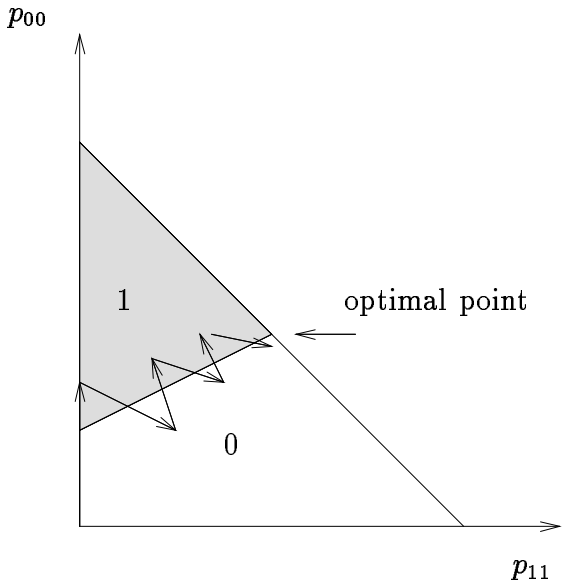


Figure 2: Typical trajectory towards the optimal point of the strategy of penny-matching when playing contract against $f_{\alpha,\beta}$. Matching ones results in an increase to $p_{11}$ and a decrease to $p_{00}$, while matching zeros has the reverse effect, resulting in the alternating pattern shown. Furthermore, the distance traveled with each successive match decreases with time.

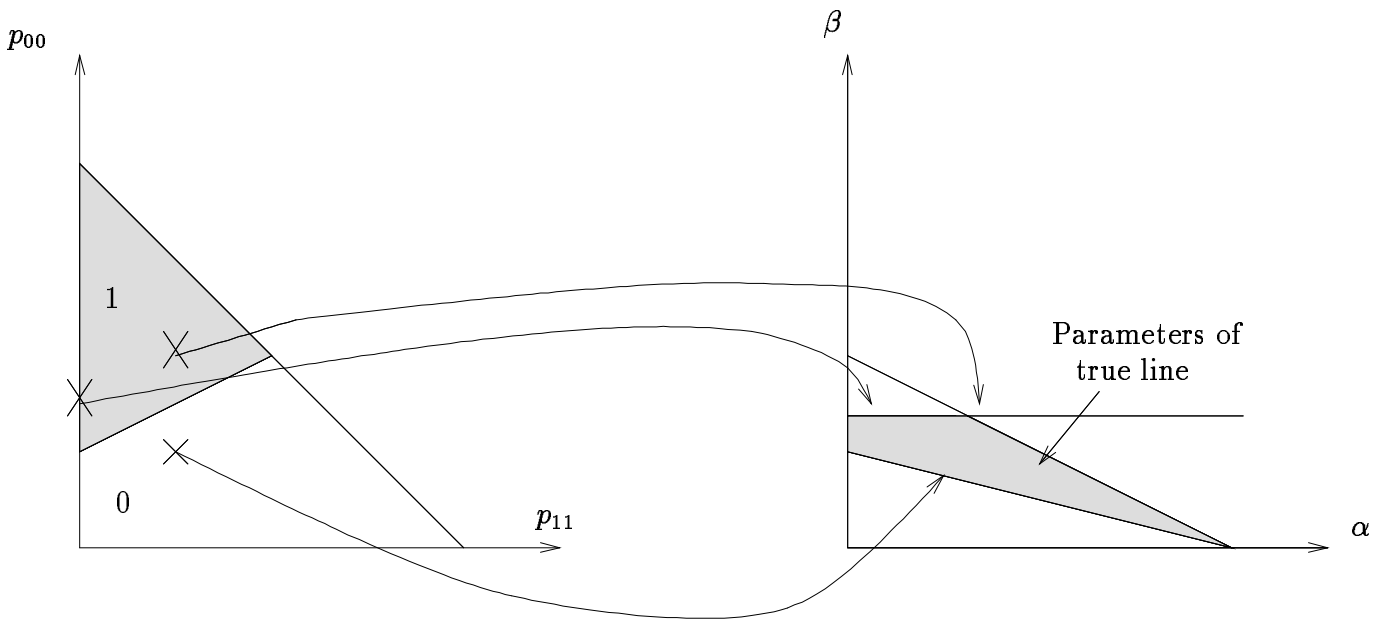


Figure 3: Construction of the version space following three rounds of play of contract against $f_{\alpha,\beta}$ in which the values of $(p_{00}, p_{11})$ were at the locations indicated in the figure on the left. Each example $\langle (p_{00}, p_{11}), f_{\alpha,\beta}(p_{00}, p_{11}) \rangle$ provides a linear constraint on the unknown parameters, shown in the figure on the right. The simultaneous satisfaction of all the constraints results in a convex polygon in the $\alpha - \beta$ plane that always contains the true parameters of the adversary.

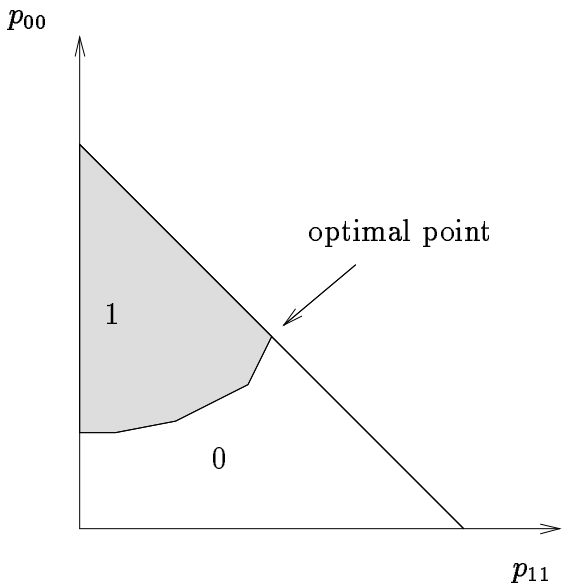


Figure 4: Example of an adversary defined by an intersection of lines of increasing slope, showing the regions in which the adversary plays 0 and 1, the line representing the constraint $p_{00} + p_{11} \leq 1$, and the optimal point of play lying on this line.

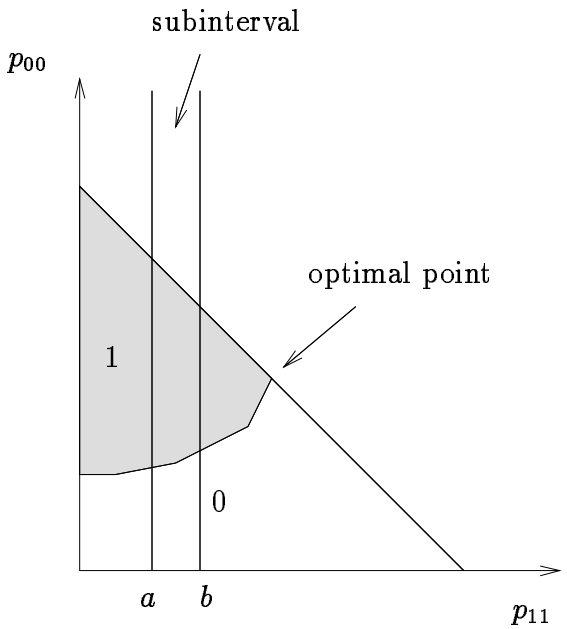


Figure 5: A subinterval $[a, b)$ defined by the algorithm of Theorem 3.5. The examples falling in this subinterval are used to define the version space for a line. Since this particular subinterval includes a corner point of the adversary, at some point this version space may become empty, resulting in the splitting of the subinterval.