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 probabilistically.[1]

More formally, a *PTA* M, $M = (Q, q_0, \gamma, \tau)$ is defined similarly to a *PAA* except that 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 *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 [1] 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 [3] show that if such a strategy exists, then computing its payoff *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 approximating the optimal payoff for $N = poly(n)$ rounds within $\Omega(1/\sqrt{n})$ is $\mathcal{PSPACE}$-complete. As in [3], 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 A.1 *For any given non-trivial game G,[2] the problem of approximating the optimal expected payoff for playing the game G against a given PTA M for $N = poly(n)$ rounds within $\Omega(1/\sqrt{n})$, 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 ℓ. Shamir [4] 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 $q = poly(\ell)$

[1] 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.

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

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 observation has been made in previous works about interactive proof systems (e.g., [2]). Since the decision process V employs must run in polynomial time, we may describe it as a circuit C_V of size polynomial in ℓ 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 expected payoff arbitrarily close to 1 for large enough N, and if $x \notin L$ then any strategy against $M_{L,x}$ (for the same number of rounds) has expected payoff at most $1 - \epsilon$ where $\epsilon = \Omega(1/|C_V|) = \Omega(1/\sqrt{n})$. It directly follows that approximating the optimal expected payoff within ϵ 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 strategy 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 strategy learning algorithm high expected payoff in all following rounds, 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 $\Omega(1/|C_V|) = \Omega(1/\sqrt{n})$ 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 strategy 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 arbitrarily 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 jth 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 jth bit played was 1 or 0. If the jth 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 jth 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 jth bit should be random, it is chosen randomly on the $j-1$ round, but is viewed only on the jth 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, Lth 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 $\tau(Accept, 0, Accept) = \tau(Accept, 1, Accept) = 1$.

The proof that the problem is in $\mathcal{PSPACE}$ is very similar to the one given in [3] and is thus omitted. $\square$

References

[1] 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.

[2] Joe Kilian. Zero-knoledge with Log-Space Verifiers. In *Proceedings of the Twenty Ninth Annual Symposium on Foundations of Computer Science*, pages 25–35, 1988.

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

[4] Adi Shamir. IP = PSPACE. In *Proceedings of the Thirty First Annual Symposium on Foundations of Computer Science*, pages 11–15, 1990.