Efficient PAC Reinforcement Learning in Regular Decision Processes

Alessandro Ronca and **Giuseppe De Giacomo**
DIAG, Sapienza University of Rome, Italy
{ronca,degiacomo}@diag.uniroma1.it

Abstract

Recently *regular decision processes* have been proposed as a well-behaved form of *non-Markov decision process*. *Regular decision processes* are characterised by a transition function and a reward function that depend on the whole history, though *regularly* (as in regular languages). In practice both the transition and the reward functions can be seen as finite transducers. We study *reinforcement learning* in regular decision processes. Our main contribution is to show that a near-optimal policy can be PAC-learned in polynomial time in a set of parameters that describe the underlying decision process. We argue that the identified set of parameters is minimal and it reasonably captures the difficulty of a regular decision process.

1 Introduction

The standard RL setting [Sutton and Barto, 2018] has long been characterised by the Markov assumption, which is arguably limiting. *Regular decision processes* (RDPs) have been proposed as a formalism with the promise to dispense with the Markov assumption while retaining good computational properties [Brafman and De Giacomo, 2019]. Complete knowledge of an RDP allows one to compute an equivalent Markov decision process (MDP), in a precise formal sense, and then to apply standard solution techniques for MDPs. This correspondence is not immediately applicable in the standard reinforcement learning (RL) setting, where an agent has no prior knowledge of the decision process. Instead, one needs to learn simultaneously a model of the dependencies on the past, and how observations and rewards evolve [Brafman and De Giacomo, 2019]. The possibility for an RL agent to learn such a model in the form of an automaton has been exploited in [Abadi and Brafman, 2020]. The work provides empirical evidence that algorithms based on automata learning can find near-optimal policies for certain RDPs. However, the proposed technique requires exponential time to converge unless only a small number of traces have high probability to be observed: it relies on statistics for single traces that may have exponentially-low probability, and hence require exponential time to be observed a number of times that is sufficient to compute accurate statistics.

In this paper, for the first time to the best of our knowledge, we introduce a technique for RL in RDPs which has formal guarantees analogous to those studied in literature for RL in MDPs [Fiechter, 1994; Brafman and Tennenholtz, 2002; Kearns and Singh, 2002; Auer *et al.*, 2002; Kakade, 2003; Strehl and Littman, 2005; Auer and Ortner, 2006; Strehl *et al.*, 2009; Audibert *et al.*, 2009; Jaksch *et al.*, 2010; Szita and Szepesvári, 2010; Lattimore and Hutter, 2014; Dann and Brunskill, 2015; Dann *et al.*, 2017; Azar *et al.*, 2017; Bai *et al.*, 2020; Wang *et al.*, 2020].

We show that RL in RDPs is feasible in *polynomial time* with respect to the *probably approximately correct* (PAC) learning criterion [Valiant, 1984; Kearns and Vazirani, 1994]. We present an algorithm that computes a near-optimal policy with high confidence, in a number of steps that is polynomial in the required accuracy and confidence, and in a set of parameters that describe the underlying RDP. The algorithm first learns a *probabilistic deterministic finite automaton* (PDFA) that represents the underlying RDP, then it maps it to an MDP, which is solved to compute an intermediate stationary policy, which is then turned into the final policy by composing it with the transition function of the learned PDFA. This process is repeated many times, yielding a better policy as more data becomes available, and converging in polynomial time.

Our results show that the Markov assumption can be largely removed while maintaining polynomial-time guarantees. This is a fundamental positive result, that should boost the confidence in finding effective RL techniques that dispense with the Markov assumption. Furthermore, our results should draw attention on RDPs, which have the potential of becoming the standard formalism for RL in the non-Markov setting, similarly to MDPs in the Markov setting.

Our work shows a strong connection between RDPs and PDFA, which allows one to take advantage of the many fundamental results and learning algorithms available for PDFA [Kearns *et al.*, 1994; Ron *et al.*, 1998; Clark and Thollard, 2004; Palmer and Goldberg, 2007; Balle *et al.*, 2013; Balle *et al.*, 2014]. On one hand, the negative results from [Kearns *et al.*, 1994] about the impossibility of polynomial-time learning when *state distinguishability* is low transfer to RDPs. On the other hand, when state distinguishability is sufficiently high, PDFA techniques allow for polynomial-time learning (cf. [Ron *et al.*, 1998; Balle *et al.*, 2013]), and we

show that such techniques can be leveraged to learn RDPs. In particular, we claim that PDFA techniques are the only known techniques so far to effectively learn states when they determine probability distributions, as it happens in RDPs.

Proofs of all results are given in an extended version of this paper [Ronca and De Giacomo, 2021]

2 Preliminaries

Transducers. We follow [Moore, 1956]. A *transducer* is a tuple $\langle Q, q_0, \Sigma, \tau, \Gamma, \theta \rangle$ where: Q is a finite set of states; $q_0 \in Q$ is the initial state; Σ is the finite input alphabet; $\tau : Q \times \Sigma \to Q$ is the deterministic transition function; Γ is the finite output alphabet; $\theta : Q \to \Gamma$ is the output function. We extend the use of τ to strings of length greater than one as $\tau(q, \sigma_1\sigma_2 \dots \sigma_n) = \tau(\tau(q, \sigma_1), \sigma_2 \dots \sigma_n)$, and to the empty string as $\tau(q, \varepsilon) = q$. We also extend the use of θ to arbitrary strings as $\theta(q, \sigma_1 \dots \sigma_n) = \theta(q)\,\theta(\tau(q, \sigma_1), \sigma_2 \dots \sigma_n)$ where the base case is $\theta(q, \varepsilon) = \theta(q)$. Furthermore, we write $\tau(\sigma_1 \dots \sigma_n)$ for $\tau(q_0, \sigma_1 \dots \sigma_n)$, and we write $\theta(\sigma_1 \dots \sigma_n)$ for $\theta(q_0, \sigma_1 \dots \sigma_n)$. Finally, we call $\theta(\sigma_1 \dots \sigma_n)$ *the output of the transducer* on $\sigma_1 \dots \sigma_n$.

Probabilistic Automata. We follow [Balle *et al.*, 2013]. A *Probabilistic Deterministic Finite Automaton* (PDFA) is a tuple $\mathcal{A} = \langle Q, \Sigma, \tau, \lambda, \zeta, q_0 \rangle$ where: Q is a finite set of states; Σ is an arbitrary finite alphabet; $\tau : Q \times \Sigma \to Q$ is the transition function; $\lambda : Q \times (\Sigma \cup \{\zeta\}) \to [0, 1]$ defines the probability of emitting each symbol from each state ($\lambda(q, \sigma) = 0$ when $\sigma \in \Sigma$ and $\tau(q, \sigma)$ is not defined); ζ is a special symbol not in Σ reserved to mark the end of a string; $q_0 \in Q$ is the initial state. It is required that: (*i*) $\lambda(q, \sigma) = 0$ when $\sigma \in \Sigma$ and $\tau(q, \sigma)$ is not defined; (*ii*) for each state $q \in Q$, $\sum_{\sigma \in (\Sigma \cup \{\zeta\})} \lambda(q, \sigma) = 1$; (*iii*) for each state $q \in Q$ that can be reached from the initial state q_0 with non-zero probability, there is a state $q' \in Q$ with $\lambda(q', \zeta) > 0$ that can be reached from q. The transition function τ is extended to strings as for transducers, and the probability function is extended as $\lambda(q, \varepsilon) = 1$ and $\lambda(q, \sigma_1\sigma_2 \dots \sigma_n) = \lambda(q, \sigma_1) \cdot \lambda(q, \sigma_2 \dots \sigma_n)$. Then, the probability that $x \in \Sigma^*$ is a string generated by the automaton starting from state q is $\lambda(q, x\zeta)$, and the probability that it is a *prefix* is $\lambda(q, x)$. For $\mu > 0$, we say that $\mathcal{A}$ is μ-*distinguishable* if $\max_x |\lambda(q_1, x) - \lambda(q_2, x)| \geq \mu$ for every two distinct states q_1 and q_2.

Non-Markov Decision Processes. A *Non-Markov Decision Process* (NMDP) (cf. [Brafman and De Giacomo, 2019]) is a tuple $\mathcal{P} = \langle A, S, R, \mathbf{T}, \mathbf{R}, \gamma \rangle$ where: A is a finite set of *actions*; S is a finite set of *states* (or *observation states* to distinguish them from automata and transducer states); $R \subseteq \mathbb{R}_{\geq 0}$ is a finite set of non-negative *reward values*; $\mathbf{T} : S^* \times A \times S \to [0, 1]$ is the *transition function* which defines a probability distribution $\mathbf{T}(\cdot|h, a)$ over S for every $h \in S^*$ and every $a \in A$; $\mathbf{R} : S^* \times A \times S \to R$ is the *reward function*; $\gamma \in (0, 1)$ is the *discount factor*. The transition and reward functions can be combined into the *dynamics function* $\mathbf{D} : S^* \times A \times S \times R \to [0, 1]$ which defines a probability distribution $\mathbf{D}(\cdot|h, a)$ over $S \times R$ for every $h \in S^*$ and every $a \in A$. Namely, $\mathbf{D}(s, r|h, a)$ is $\mathbf{T}(s|h, a)$ if $r = \mathbf{R}(h, a, s)$, and zero otherwise. Every element of S^* is called a *history*. A *policy* is a function $\pi : S^* \times A \to [0, 1]$ that, for every history h, defines a probability distribution $\pi(\cdot|h)$ over the actions A. A policy π is *deterministic on a history* h if $\pi(a|h) = 1$ for some action a, in which case we write $\pi(h) = a$; and it is *deterministic* if it is deterministic on every history. A policy π is *uniform on a history* h if $\pi(a|h) = 1/|A|$ for every action $a \in A$. We call π_{u} the policy that is uniform on every history. Every element of $(ASR)^*$ is called a *trace*. The *dynamics of $\mathcal{P}$ under a policy π* describe the probability of an upcoming trace, given the history so far, when actions are chosen according to a policy π; it can be recursively computed as $\mathbf{D}_\pi(asrt|h) = \pi(a|h) \cdot \mathbf{D}(s, r|h) \cdot \mathbf{D}_\pi(t|hs)$, with base case $\mathbf{D}_\pi(\varepsilon|h) = 1$ for ε the empty trace. The *value of a policy π on a history h*, written $\mathbf{v}_\pi(h)$, is the expected discounted sum of future rewards when actions are chosen according to π given that the history so far is h; it can be recursively computed as $\mathbf{v}_\pi(h) = \sum_{asr} \pi(a|h) \cdot \mathbf{D}(s, r|h, a) \cdot (r + \gamma \cdot \mathbf{v}_\pi(hs))$. The *optimal value on a history* h is $\mathbf{v}_*(h) = \max_\pi \mathbf{v}_\pi(h)$, which can be expressed without reference to any policy as $\mathbf{v}_*(h) = \max_a \left(\sum_{sr} \mathbf{D}(s, r|h, a) \cdot (r + \gamma \cdot \mathbf{v}_*(hs))\right)$. The *value of an action a on a history h under a policy π* , written $\mathbf{q}_\pi(h, a)$, is the expected discounted sum of future rewards when the next action is a and the following actions are chosen according to π, given that the history so far is h; it is $\mathbf{q}_\pi(h, a) = \sum_{sr} \mathbf{D}(s, r|h, a) \cdot (r + \gamma \cdot \mathbf{v}_\pi(hs))$. The *optimal value of an action a on a history h* is $\mathbf{q}_*(h, a) = \max_\pi \mathbf{q}_\pi(h, a)$, and it can be expressed as $\mathbf{q}_*(h, a) = \sum_{sr} \mathbf{D}(s, r|h, a) \cdot (r + \gamma \cdot \mathbf{v}_*(hs))$. A policy π is *optimal on a history* h if $\mathbf{v}_\pi(h) = \mathbf{v}_*(h)$. For $\epsilon > 0$, a policy π is *ϵ-optimal on h* if $\mathbf{v}_\pi(h) \geq \mathbf{v}_*(h) - \epsilon$. A policy is *optimal* (resp., *ϵ-optimal*) if it is so on every history. We also say *near-optimal* to say ϵ-optimal for some ϵ.

Regular Decision Processes. A *Regular Decision Process (RDP)* [Brafman and De Giacomo, 2019][1] is an NMDP $\mathcal{P} = \langle A, S, R, \mathbf{T}, \mathbf{R}, \gamma \rangle$ whose transition and reward functions can be represented by *finite transducers*. Specifically, there is a finite transducer that, on every history h, outputs the function $\mathbf{T}_h : A \times S \to [0, 1]$ induced by $\mathbf{T}$ when its first argument is h; and there is a finite transducer that, on every history h, outputs the function $\mathbf{R}_h : A \times S \times R \to [0, 1]$ induced by $\mathbf{R}$ when its first argument is h. Note that the cross-product of such transducers yields a finite transducer for the dynamics function $\mathbf{D}$ of $\mathcal{P}$.

Markov Decision Processes. A *Markov Decision Process (MDP)* [Bellman, 1957; Puterman, 1994] is an NMDP $\langle A, S, R, \mathbf{T}, \mathbf{R}, \gamma \rangle$ where both the transition function and the reward function (and hence the dynamics function) depend only on the last state in the history, which is never empty since every history starts with a state that is chosen arbitrarily. Specifically, for every pair of non-empty histories $h_1 s$ and $h_2 s$, it holds that $\mathbf{T}(h_1 s, \cdot) = \mathbf{T}(h_2 s, \cdot)$ and $\mathbf{R}(h_1 s, \cdot) = \mathbf{R}(h_2 s, \cdot)$. Similarly, we say that a policy π is *stationary* if $\pi(h_1 s, \cdot) = \pi(h_2 s, \cdot)$ for every pair of non-empty histories $h_1 s$ and $h_2 s$. In the case of MDPs, we can see all history-

[1] In [Brafman and De Giacomo, 2019] the functions $\mathbf{T}$ and $\mathbf{R}$ are represented using the temporal logics on finite traces LDL_f. Here instead we use directly finite transducers to express them. Note that all $\mathbf{T}$ and $\mathbf{R}$ representable in LDL_f are indeed expressible through finite transducers.

dependent functions—e.g., transition and reward functions, value functions, policies—as taking a single state in place of a history.

3 PAC-RL in RDPs

We consider *reinforcement learning (RL)* as the problem of an agent that has to learn an optimal policy for an unknown RDP $\mathcal{P}$, by acting and receiving observation states and rewards according to the transition and reward functions of $\mathcal{P}$. The agent performs a sequence of actions $a_1 \dots a_n$, receiving an observation state s_i and a reward r_i after each action a_i. The agent has the opportunity to *stop* and possibly start over. This process generates strings of the form $a_1 s_1 r_1 \dots a_n s_n r_n$ which we call *episodes*. They form the *experience* of the agent, which is the basis to learn an optimal policy.

Example 1. *As a running example, we consider an agent that has to cross a $2 \times m$ grid while avoiding enemies. Every enemy guards a two-cell column: enemy i guards cells $(0, i)$ and $(1, i)$, and it is found in cell $(0, i)$ with probability p_i^0 or p_i^1. Initially the probability is p_i^0, and the two probabilities are swapped every time the agent hits an enemy. At every step, say i, the agent has to decide whether to go through cell $(0, i)$ or $(1, i)$, taking action a_0 or a_1, respectively. Specifically, when in cell $(0, i)$ or $(1, i)$, action a_0 leads to $(0, i + 1)$ and action a_1 to $(1, i + 1)$. From the last column, the agent is brought back to the first one. The agent receives reward one every time it avoids an enemy. Each observation state s is a triple $\langle i, j, e\rangle$ where $i \in [0, 1]$ and $j \in [0, m-1]$ are the coordinates of the agent and $e \in \{enemy, clear\}$ denotes whether an enemy is in the agent's current cell. To maximise rewards, the agent has to learn to predict the enemies' positions, which depend on the history of observation states in a regular manner. Such a dependency is described by a transducer with $2m$ states of the form $\langle q_1, q_2\rangle$ where $q_1 \in [0, m - 1]$ stores the agent's current column and q_2 is a bit that keeps track of whether probabilities p_i^0 or p_i^1 are being used.*

Our goal is to understand how fast an agent can learn a near-optimal policy. In particular, we want the agent to find near-optimal policies in the shortest possible time, and we are not interested in the agent maximising the collected rewards during learning. Thus, we require the agent to return policies $\pi_1, \pi_2, \dots$ that improve over time. The idea is that these policies can be passed to an executing agent, living in such an environment, whose behaviour will improve as it receives better policies. Then, the central question is how fast an agent can reach a point after which it outputs only good policies.

Now we make our discussion more precise. To measure the learning performance of an agent, we consider the number of *steps* it performs.

Definition 1. *An* action step *consists in performing an action and collecting the resulting observation-state and reward. A* step *is an action step or an elementary computation step.*

Then, to have a reasonable notion of learning, we borrow from the *Probably Approximately Correct (PAC)* framework [Valiant, 1984; Kearns and Vazirani, 1994], similarly to what is done in previous work on RL in MDPs (cf. [Fiechter, 1994; Kearns and Singh, 2002; Strehl *et al.*, 2009]). The PAC framework is based on the observation that exact learning is infeasible. Thus, it introduces two parameters $\epsilon > 0$ and $\delta \in (0, 1)$ that describe the required accuracy and the confidence of success, respectively. In the RL setting it translates into looking for policies that are ϵ-optimal with probability at least $1 - \delta$. Then, RL is considered feasible if these policies can be found in a number of steps that is polynomial in $1/\epsilon$ and $\ln(1/\delta)$, and in other parameters that describe the underlying RDP.

Definition 2. *An RL agent (or algorithm) is said to reach accuracy ϵ and confidence δ in the moment it returns the first policy π_* such that π_* is ϵ-optimal with probability at least $1 - \delta$ and the same holds for every policy returned after π_*.*

Definition 3. *Let $\mathbf{d}_\mathcal{P}$ be a list of parameters describing $\mathcal{P}$, let A be its actions, and let γ be its discount factor. An RL agent (or algorithm) is PAC-RL with respect to $\mathbf{d}_\mathcal{P}$ if, for every $\epsilon > 0$ and $\delta \in (0, 1)$, given $(A, \gamma, \epsilon, \delta)$ as input, it reaches accuracy ϵ and confidence δ in $\mathrm{poly}(1/\epsilon, \ln(1/\delta), \mathbf{d}_\mathcal{P})$ steps.*

We propose to describe an RDP $\mathcal{P} = \langle A, S, R, \mathbf{T}, \mathbf{R}, \gamma\rangle$ using the following parameters.

$$\mathbf{d}_\mathcal{P} = \left(|A|, \frac{1}{1-\gamma}, R_{\max}, n, \frac{1}{\rho}, \frac{1}{\mu}, \frac{1}{\eta}\right) \tag{1}$$

The first three parameters take into account the number of actions $|A|$, the discount factor γ, the maximum reward value $R_{\max}$, and the number of states n of the minimum transducer for the dynamics of $\mathcal{P}$. Then, the reachability ρ of an RDP measures how easy it is to reach states of the dynamics transducer T when actions are taken uniformly at random.

Definition 4. *The* reachability *of $\mathcal{P}$ is the minimum non-zero probability ρ that a given state of the dynamics transducer T is reached from the initial state within n steps (with n the number of states) when actions are chosen uniformly at random.*

The distinguishability μ of an RDP measures how easy it is to distinguish states of the dynamics transducer T when actions are taken uniformly at random. It is a parameter inherited from the PDFA literature [Ron *et al.*, 1998; Balle *et al.*, 2013].

Definition 5. *The* distinguishability *of $\mathcal{P}$ is the minimum $\mu \in (0, 1)$ such that one of the following conditions holds for every two histories h_1, h_2, where π_u is the uniform policy:*

- $\mathbf{D}_{\pi_\mathrm{u}}(t|h_1) = \mathbf{D}_{\pi_\mathrm{u}}(t|h_2)$ *for every trace t;*
- $|\mathbf{D}_{\pi_\mathrm{u}}(t|h_1) - \mathbf{D}_{\pi_\mathrm{u}}(t|h_2)| > \mu$ *for some trace t.*

The degree of determinism η measures how easy it is to discover transitions. If η is small, then there is some transition that is possible, but unlikely to be observed. Note that this parameter takes value one when the RDP is deterministic.

Definition 6. *The* degree of determinism *η of $\mathcal{P}$ is the minimum non-zero probability that $\mathbf{T}(\cdot|h, a)$ assigns to an observation state.*

Example 2. *Consider the RDP of Example 1. Its reachability ρ is $1/2$, since, by an inductive argument, every state reachable in i steps has probability at least $1/2$ to be visited at step i and, from there, each of the two next states is visited*

with probability $1/2$*. The degree of determinism* η *is the minimum, for any* i*, among the probabilities* p_i^0 *and* p_i^1*, and their complements. The distinguishability* μ *is given by the uniform probability of an action* $1/2$ *times the minimum among* η *and the values* $|p_i^0 - p_i^1|$*.*

One can show that all the above parameters are necessary. For the number of actions, discount, and maximum reward parameters (and also for the accuracy and confidence parameters) the result follows from a known lower bound for the MAB problem [Mannor and Tsitsiklis, 2004]. Then, the number of transducer states, reachability, and degree of determinism can be shown to increase the number of action steps required to visit the relevant parts of the dynamics transducer a sufficient number of times. Finally, for the distinguishability parameter, it is straighforward to adapt the hardness proof for PDFA given in [Kearns *et al.*, 1994], which assumes hardness of learning noisy parity function, a standard cryptographic assumption.

Theorem 1. *There is no algorithm that is PAC-RL with respect to a strict subset of the parameters* $\mathbf{d}_{\mathcal{P}}$ *given in* (1)*, if it is hard to learn noisy parity functions.*

4 PAC-RL via Probabilistic Automata

We present an RL algorithm for RDPs that relies on probabilistic automata. We consider an RDP $\mathcal{P} = \langle A, S, R, \mathbf{T}, \mathbf{R}, \gamma \rangle$, its dynamics function $\mathbf{D}$, and the minimum transducer $T = \langle Q, q_0, S, \tau, \Gamma, \theta \rangle$ that represents $\mathbf{D}$ in the sense that $\theta(h)(a, s, r) = \mathbf{D}(s, r|h, a)$.

Representing RDPs as PDFA. When learning, the agent has the possibility to experience multiple episodes. In particular, the agent has an extra action, called *stop action*, that ends the current episode and starts a new one. To take advantage of that, the agent generates episodes following a stationary policy that chooses to stop with a non-zero probability p, and it chooses an action from A uniformly if it does not stop.

Definition 7. *The* stop action *is a special action* ζ *that allows the agent to terminate the current episode and start a new one. The* exploration policy with stop probability $p > 0$*, written* π_p*, is the policy that selects the stop action* ζ *with probability* p*, and each action from* A *with probability* $(1-p)/|A|$*.*

Under an exploration policy, an RDP determines a probability distribution on traces, and hence can be seen as a PDFA. Specifically, the dynamics of $\mathcal{P}$ under π_p are captured by the PDFA $\mathcal{A} = \langle Q, \Sigma, \tau', \lambda, \zeta, q_0 \rangle$ where:

- alphabet $\Sigma = \{asr \in ASR \mid \exists h.\ \mathbf{D}(s, r|h, a) > 0\}$,
- transitions $\tau'(q, asr) = \tau(q, s)$,
- probability function:
  - $\lambda(q, asr) = ((1-p)/|A|) \cdot \theta(q)(a, s, r)$,
  - $\lambda(q, \zeta) = p$.

The dynamics of $\mathcal{P}$ are captured in the sense that the following holds for every history h, trace t, and string h' such that the projection of h' on observation states coincides with h:

$$\mathbf{D}_{\pi_p}(t|h) = \lambda(\tau'(q_0, h'), t).$$

Algorithm 1 Reinforcement Learning $\texttt{RL}(A, \gamma, \epsilon, \delta)$

Input: Actions A, discount factor γ, required precision ϵ, confidence parameter δ.
Output: Policies.

1: **for** $\ell = 1, 2, \ldots$ **do**
2: $p \leftarrow 1/(10\ell + 1)$; $k \leftarrow (2/p) \cdot \ell^2 \cdot (\ell + 5 \ln \ell)$
3: $X \leftarrow \emptyset$; $i \leftarrow 0$; $hardStop \leftarrow false$
4: **while** $\neg hardStop$ **do**
5: $x, hardStop \leftarrow$ generate an episode under policy π_p with a hard stop after $k - i$ actions
6: $i \leftarrow$ increase i by the number of actions in x
7: **if** $\neg hardStop$ **then**
8: $X \leftarrow X \cup \{x\}$
9: **end if**
10: **end while**
11: $\hat{\Sigma} \leftarrow$ symbols in X; $\hat{R}_{\max} \leftarrow$ max reward in X
12: $\hat{\mathcal{A}} \leftarrow$ learn PDFA by calling $\texttt{AdaCT}(\ell, |\hat{\Sigma}|, \delta/2, X)$
13: $\hat{M} \leftarrow$ compute the MDP induced by $\hat{\mathcal{A}}$ and γ
14: $m \leftarrow \left\lceil \frac{1}{1-\gamma} \cdot \ln\left(\frac{2 \cdot \hat{R}_{\max}}{\epsilon \cdot (1-\gamma)^2}\right) \right\rceil$
15: $\pi \leftarrow$ solve $\hat{M}$ by calling $\texttt{ValueIteration}(\hat{M}, m)$
16: **return** transducer for the composition of π with the projection-on-states of the transition function of $\hat{\mathcal{A}}$
17: **end for**

PAC-RL Algorithm. Algorithm 1 provides the pseudocode of our RL algoritm. The algorithm repeats the following operations for increasing values of an integer variable ℓ, starting from 1. (Line 2) It computes the stop probability p, and the maximum number k of actions to perform during the current iteration. (Line 3) It initialises the set of episodes X to the empty set, a counter i for the actions to zero, and a flag $hardStop$ to know when a hard stop has occurred. (Lines 4–10) It generates episodes following the exploration policy with stop probability p, and with a hard stop if the number of performed actions reaches k. Episodes are stored in variable X. (Line 11) It reads the set $\hat{\Sigma}$ of action-state-reward symbols from X, and the maximum reward $\hat{R}_{\max}$ occurring in X. (Line 12) It learns a PDFA $\hat{\mathcal{A}} = \langle \hat{Q}, \hat{\Sigma}, \hat{\tau}', \hat{\lambda}, \zeta, \hat{q}_0 \rangle$ via the AdaCT algorithm [Balle *et al.*, 2013] instantiated with ℓ as an upper bound on the number of automaton states, $|\hat{\Sigma}|$ as an estimate of the size of the alphabet, confidence parameter $\delta/2$, and set of strings X. (Line 13) Starting from $\hat{\mathcal{A}}$, the algorithm computes the MDP $\hat{M} = \langle A, \hat{Q}, \hat{R}, \mathbf{D}^{\hat{M}}, \gamma, \hat{q}_0 \rangle$ where $\hat{R}$ consists of each reward value occurring as the third component of an element of $\hat{\Sigma}$, and the dynamics function is as follows:

$$\mathbf{D}^{\hat{M}}(q_2, r|q_1, a) = (|A|/(1-p)) \sum_{s:\hat{\tau}'(q_1, asr) = q_2} \hat{\lambda}(q_1, asr).$$

(Lines 14–15) The algorithm then solves $\hat{M}$ via m iterations of the classic *value iteration* algorithm with action-value estimates initialised to zero. Specifically, value iteration computes an approximation $\hat{\mathbf{q}}_*^{\hat{M}}$ of the optimal action-value function $\mathbf{q}_*^{\hat{M}}$ of $\hat{M}$, and then computes the greedy policy π with respect $\hat{\mathbf{q}}_*^{\hat{M}}$, which is a stationary policy on state space $\hat{Q}$.

	Definition
ϵ'	$\frac{(1-\gamma)^3 \cdot \epsilon}{3 \cdot R_{\max}}$
ϵ''	$\frac{(1-p) \cdot \epsilon'}{\lvert A \rvert \cdot n \cdot \lvert \Sigma \rvert}$
δ_0	$\frac{\delta}{2 \cdot \hat{n} \cdot (\hat{n} \cdot \lvert \Sigma \rvert + \lvert \Sigma \rvert + 1)}$
N'	$\frac{22 \cdot \mathrm{e} \cdot \lvert A \rvert \cdot \hat{n} \cdot \lvert \Sigma \rvert}{\rho \cdot \eta \cdot (1-p) \cdot \mu^2} \cdot \ln \frac{704 \cdot \lvert A \rvert \cdot \hat{n} \cdot \lvert \Sigma \rvert}{\rho \cdot \eta \cdot \mu^2 \cdot p \cdot \delta_0^2}$
N''	$\frac{\hat{n} \cdot \lvert \Sigma \rvert \cdot \lvert A \rvert}{0.9 \cdot \rho \cdot \eta \cdot (1-p) \cdot (\epsilon'')^2} \cdot \ln \frac{2 \cdot (\lvert \Sigma \rvert + 1)}{\delta_0}$

Table 1: Quantities used in the PAC analysis.

(Line 16) The algorithm returns a transducer that represents the policy function $\pi(\hat{\tau}(\cdot))$ obtained by composing the stationary policy π and the 'projection' transition function $\hat{\tau}$ defined as $\hat{\tau}(q, s) = \hat{\tau}'(q, asr)$ for an arbitrary choice of a and r such that $\hat{\lambda}(q, asr) > 0$; note that every choice yields the same result and a choice always exists. Specifically, the returned transducer is $\langle \hat{Q}, \hat{q}_0, \hat{S}, \hat{\tau}, A, \theta' \rangle$ where $\theta'(\hat{q}) = \pi(\hat{q})$ and $\hat{S}$ consists of each observation-state occurring as the second component of an element of $\hat{\Sigma}$.

5 PAC Analysis

Algorithm 1 shows that RL in RDPs is feasible in polynomial time.

Theorem 2. *Algorithm 1 is PAC-RL with respect to the parameters $\mathbf{d}_{\mathcal{P}}$ given in* (1)*.*

We analyse Algorithm 1 to show that the theorem holds. We first show that we can compute a near-optimal policy for $\mathcal{P}$ starting from a near-optimal policy for the MDP induced by the dynamics transducer T, that is defined as $M = \langle A, Q, R, \mathbf{D}^M, \gamma \rangle$ where the dynamics function is as follows:

$$\mathbf{D}^M(q_2, r | q_1, a) = \textstyle\sum_{s:\tau(q_1,s)=q_2} \theta(q_1)(a, s, r).$$

We call it the *ideal MDP*, since it is the MDP that the algorithm would build if it learned the automaton $\hat{\mathcal{A}}$ perfectly. Its key property is that an ϵ-optimal policy for $\mathcal{P}$ can be obtained from an ϵ-optimal policy for M by composing it with the transition function of T.

Lemma 1. *If π is an ϵ-optimal policy for M, then $\pi(\tau(\cdot))$ is an ϵ-optimal policy for $\mathcal{P}$.*

Now that we know that it suffices to compute an ϵ-optimal policy for the ideal MDP M, we establish the accuracy required for the MDP $\hat{M}$ that is actually computed by the algorithm (Line 14) in order to be the basis for computing an ϵ-optimal policy for M.

Definition 8. *We say that $\hat{M}$ is an α-approximation of M if the two MDPs have the same actions, rewards, and discount factor, and there is a bijection ϕ between their states such that $\|\mathbf{D}^M(\cdot|q,a) - \mathbf{D}^{\hat{M}}(\cdot|\phi(q),a)\|_1 \leq \alpha$ for every q and a.*

The following lemma is an application of known results for MDPs. First, if $\hat{M}$ approximates M, then the value functions of $\hat{M}$ approximate the value functions of M [Strehl and Littman, 2005]. Then, the value functions of $\hat{M}$ can be computed in polynomial time with high accuracy via the value iteration algorithm [Strehl *et al.*, 2009]. Finally, any policy that greedily chooses the best action according to an accurate estimate of the optimal action-value function is near-optimal [Singh and Yee, 1994].

Lemma 2. *If $\hat{M}$ is an ϵ'-approximation of M (with ϵ' as in Table 1), then the greedy policy obtained via*

$$\left\lceil \frac{1}{1-\gamma} \cdot \ln \left(\frac{2 \cdot R_{\max}}{\epsilon \cdot (1-\gamma)^2} \right) \right\rceil$$

iterations of the value iteration *algorithm, with action-value estimates initialised to zero, is an ϵ-optimal policy for M.*

We establish the accuracy required for the learned PDFA.

Definition 9. *We say that $\hat{\mathcal{A}}$ is an α-approximation of $\mathcal{A}$ if the two automata have the same alphabet, there is a transition-preserving isomorphism ϕ between their states, and their probability functions satisfy $|\lambda(q,\sigma) - \hat{\lambda}(\phi(q),\sigma)| < \alpha$ for every state q and symbol σ, with the additional condition that $\lambda(q,\sigma) = 0$ implies $\hat{\lambda}(\phi(q),\sigma) = 0$.*

The bound on the error for the learned automaton transfers to the MDP it induces, amplified by the number of transducer states, alphabet size, and the inverse of the minimum probability assigned to an action by the exploration policy π_p.

Lemma 3. *If $\hat{\mathcal{A}}$ is an ϵ''-approximation of $\mathcal{A}$, then $\hat{M}$ is an ϵ'-approximation of M (with ϵ'' as in Table 1).*

We next derive the number of episodes and the stop probability (which determines the length of episodes) under which the `AdaCT` algorithm guarantees to learn ϵ''-approximation of $\mathcal{A}$. The accuracy of the algorithm depends on the distinguishability of the automaton to learn, on the minimum probability that a state is visited during a run, and on the minimum transition probability. Thus, we need establish lower bounds for the former three quantities. First, the distinguishability $\mathcal{A}$ is close to the distinguishability μ of $\mathcal{P}$ if p is sufficiently small, since states can be distinguished by looking at strings of length at most the number of states n. In fact, for any pair of longer strings witnessing the distinguishability, we can take their substrings of length at most n obtained by removing 'cycles', similarly to what is done in the *pumping lemma* for regular languages. The bound then follows by taking into account the probability $(1-p)^n$ of generating a string of length at least n. Second, the probability of visiting a state of $\mathcal{A}$ is at least the probability of generating a string of length at least n times the probability of visting the corresponding state in T while generating the string, which is at least the reachability ρ of $\mathcal{P}$. Third, the minimum transition probability in $\mathcal{A}$ is at least the degree of determinism η of $\mathcal{P}$ times the probability of picking an action uniformly when not stopping.

Lemma 4. *The distinguishability of $\mathcal{A}$ is at least $\mu \cdot (1-p)^n$. The minimum non-zero probability that a state of $\mathcal{A}$ is visited during a run is at least $\rho \cdot (1-p)^n$. The minimum non-zero probability of a non-ζ transition of $\mathcal{A}$ is at least $\eta \cdot (1-p)/|A|$.*

Given the bounds above, the next lemma follows from the guarantees for the `AdaCT` algorithm [Balle *et al.*, 2013]. In

particular, the guarantees require a number of episodes that depends on the specified upper bound on the states, on the specified alphabet size, on the required accuracy and confidence, and on the three quantities of the previous lemma, which applies considering that $(1-p)^n \geq 0.9$ because of the condition on p.

Lemma 5. *If $p \leq 1/(10n+1)$, $\hat{n}$ is an upper bound on the number n of states of $\mathcal{A}$, and X are strings generated by $\mathcal{A}$ with $|X| \geq \max(N', N'')$, then* $\texttt{AdaCT}(\hat{n}, |\Sigma|, \delta/2, X)$ *returns an ϵ''-approximation of $\mathcal{A}$ with probability $1-\delta/2$.*

The required sample size and stop probability are achieved in a polynomial number ℓ of iterations of the algorithm. The sample size also guarantees $\hat{\Sigma} = \Sigma$ and $\hat{R}_{\max} = R_{\max}$. Furthermore, introducing a logarithmic dependency on δ ensures that a hard stop occurs with probability at most $\delta/2$, which combined with the probability of failure of `AdaCT` is still less than δ. The ℓ-th iteration of the algorithm performs the number of actions k specified in Line 2, which is $\widetilde{O}(\ell^4)$, and hence the algorithm performs $\widetilde{O}(\ell^5)$ action steps in ℓ iterations, which is:

$$\widetilde{O}\left(n^5 + \frac{|\Sigma|^5 \cdot |A|^5}{\rho^5 \cdot \eta^5} \cdot \left(\frac{1}{\mu^{10}} + \frac{n^{10} \cdot |\Sigma|^{10} \cdot |A|^{10} \cdot R_{\max}^{10}}{\epsilon^{10} \cdot (1-\gamma)^{30}}\right)\right)$$

The bound mentions $|\Sigma|$ which is not in $\mathbf{d}_{\mathcal{P}}$, but satisfies $|\Sigma| \leq n \cdot |A| \cdot \lceil 1/\eta \rceil$. Then, a polynomial number of action steps immediately implies a polynomial number of steps overall. In particular, `AdaCT` runs in time polynomial in the size of the input sample and in the specified values for the number of states and alphabet size, and also value iteration `ValueIteration` runs in time polynomial in the size of the input MDP and in the number of iterations. We conclude that Algorithm 1 reaches accuracy ϵ and confidence δ in a polynomial number of steps. Therefore, Algorithm 1 is PAC-RL.

6 Exploiting Prior Knowledge

We observe that if we know (or we can estimate) the number of states in the dynamics transducer, we can devise a simpler algorithm, Algorithm 2, with better performance bounds. Algorithm 2 is simpler than Algorithm 1, since it does need to search for the number of transducer states. The stop probability is constant throughout a run, and computed based on the upper bound on the states. Since the stop probability is uniform across iterations, all episodes can be seen as generated by the same automaton, and hence we can accumulate episodes instead of deleting previous ones. Furthermore, we can now call `AdaCT` directly with the given parameter $\hat{n}$. A simplified analysis, again based on Lemmas 1–5, yields the following polynomial bound on the expected number of action steps, which immediately implies a polynomial bound on the expected number of steps overall.

Theorem 3. *Algorithm 2 on input $(A, \gamma, \epsilon, \delta, \hat{n})$ reaches accuracy ϵ and confidence δ within an expected number of action steps that is:*

$$\widetilde{O}\left(\frac{|A| \cdot \hat{n} \cdot |\Sigma|}{\rho \cdot \eta} \cdot \left(\frac{1}{\mu^2} + \frac{\hat{n}^2 \cdot |\Sigma|^2 \cdot |A|^2 \cdot R_{\max}^2}{(1-\gamma)^6 \cdot \epsilon^2}\right)\right).$$

Algorithm 2 Reinforcement Learning $\texttt{RL}(A, \gamma, \epsilon, \delta, \hat{n})$

Input: Actions A, discount factor γ, required precision ϵ, confidence parameter δ, upper bound $\hat{n}$ on transducer states.
Output: Policies.

1: $p \leftarrow 1/(10 \cdot \hat{n} + 1)$
2: $X \leftarrow \emptyset$
3: **loop**
4: $\quad x \leftarrow$ generate an episode under exploration policy π_p
5: $\quad X \leftarrow X \cup \{x\}$
6: $\quad \hat{\Sigma} \leftarrow$ symbols in X; $\hat{R}_{\max} \leftarrow$ max reward in X
7: $\quad \hat{\mathcal{A}} \leftarrow$ learn PDFA by calling $\texttt{AdaCT}(\hat{n}, |\hat{\Sigma}|, \delta, X)$
8: $\quad \hat{M} \leftarrow$ compute the MDP induced by $\hat{\mathcal{A}}$ and γ
9: $\quad m \leftarrow \left\lceil \frac{1}{1-\gamma} \cdot \ln\left(\frac{2 \cdot \hat{R}_{\max}}{\epsilon \cdot (1-\gamma)^2}\right)\right\rceil$
10: $\quad \pi \leftarrow$ solve $\hat{M}$ by calling $\texttt{ValueIteration}(\hat{M}, m)$
11: $\quad$ **return** transducer for the composition of π with the projection-on-states of the transition function of $\hat{\mathcal{A}}$
12: **end loop**

7 Discussion

We have presented RL algorithms that can learn near-optimal policies for RDPs in polynomially-many steps, in the parameters that describe the underlying RDP.

Example 3. *Algorithm 1 and Algorithm 2 (with a bound on the number of states that is polynomial in the grid length) compute a near-optimal policy in each of the RDPs introduced in Example 1 in a number of steps that is polynomial in the following quantities: (i) the grid length m, (ii) the inverse of the minimum among p_i^0, p_i^1, $1-p_i^0$, $1-p_i^1$, (iii) the inverse of the minimum value $|p_i^0 - p_i^1|$.*

Adopting PDFA techniques takes us into a different direction from existing approaches based on a direct clustering of histories such as [Abadi and Brafman, 2020]. There, histories are clustered according to the probability of the following observation state. Since the algorithm compares single histories, the accuracy of the algorithm depends on the probability of single histories, which can be exponentially-low in their length. In turn, the required length of histories can grow with the number of transducer states, and hence the approach can require exponentially-many episodes in order to achieve high accuracy. For instance, in our running example, a history of length m has probability at most g^m under any policy, with g the maximum probability among $p_i^0, p_u^1, 1-p_i^0, 1-p_i^1$. Histories of length $m-1$ are necessary to determine the best action at the m-th step. To address the issue, PDFA algorithms build states incrementally while relying on their distinguishability. This way, each state gathers the probability of all the histories it represents. In light of our results, we believe that these PDFA techniques will be instrumental in developing the next generation of tools for RL in RDPs.

Acknowledgments

This work has been partially supported by the ERC Advanced Grant WhiteMech (No. 834228) and by the EU ICT-48 2020 project TAILOR (No. 952215).

References

[Abadi and Brafman, 2020] Eden Abadi and Ronen I. Brafman. Learning and solving regular decision processes. In *IJCAI*, 2020.

[Audibert *et al.*, 2009] Jean-Yves Audibert, Rémi Munos, and Csaba Szepesvári. Exploration-exploitation tradeoff using variance estimates in multi-armed bandits. *Theor. Comput. Sci.*, 410(19), 2009.

[Auer and Ortner, 2006] Peter Auer and Ronald Ortner. Logarithmic online regret bounds for undiscounted reinforcement learning. In *NeurIPS*, 2006.

[Auer *et al.*, 2002] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, 47(2–3), 2002.

[Azar *et al.*, 2017] Mohammad Gheshlaghi Azar, Ian Osband, and Rémi Munos. Minimax regret bounds for reinforcement learning. In *ICML*, volume 70, 2017.

[Bai *et al.*, 2020] Yu Bai, Chi Jin, and Tiancheng Yu. Near-optimal reinforcement learning with self-play. In *NeurIPS*, 2020.

[Balle *et al.*, 2013] Borja Balle, Jorge Castro, and Ricard Gavaldà. Learning probabilistic automata: A study in state distinguishability. *Theor. Comput. Sci.*, 473, 2013.

[Balle *et al.*, 2014] Borja Balle, Jorge Castro, and Ricard Gavaldà. Adaptively learning probabilistic deterministic automata from data streams. *Mach. Learn.*, 96(1-2), 2014.

[Bellman, 1957] Richard Bellman. A Markovian decision process. *J. Math. Mech.*, 1957.

[Brafman and De Giacomo, 2019] Ronen I. Brafman and Giuseppe De Giacomo. Regular decision processes: A model for non-Markovian domains. In *IJCAI*, 2019.

[Brafman and Tennenholtz, 2002] Ronen I. Brafman and Moshe Tennenholtz. R-MAX: A general polynomial time algorithm for near-optimal reinforcement learning. *J. Mach. Learn. Res.*, 3, 2002.

[Clark and Thollard, 2004] Alexander Clark and Franck Thollard. PAC-learnability of probabilistic deterministic finite state automata. *J. Mach. Learn. Res.*, 5, 2004.

[Dann and Brunskill, 2015] Christoph Dann and Emma Brunskill. Sample complexity of episodic fixed-horizon reinforcement learning. In *NeurIPS*, 2015.

[Dann *et al.*, 2017] Christoph Dann, Tor Lattimore, and Emma Brunskill. Unifying PAC and regret: Uniform PAC bounds for episodic reinforcement learning. In *NeurIPS*, 2017.

[Fiechter, 1994] Claude-Nicolas Fiechter. Efficient reinforcement learning. In *COLT*, 1994.

[Jaksch *et al.*, 2010] Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *J. Mach. Learn. Res.*, 11, 2010.

[Kakade, 2003] Sham M. Kakade. *On the Sample Complexity of Reinforcement Learning*. PhD thesis, Gatsby Unit, University College London, 2003.

[Kearns and Singh, 2002] Michael J. Kearns and Satinder P. Singh. Near-optimal reinforcement learning in polynomial time. *Mach. Learn.*, 49(2–3), 2002.

[Kearns and Vazirani, 1994] Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994.

[Kearns *et al.*, 1994] Michael J. Kearns, Yishay Mansour, Dana Ron, Ronitt Rubinfeld, Robert E. Schapire, and Linda Sellie. On the learnability of discrete distributions. In *STOC*, 1994.

[Lattimore and Hutter, 2014] Tor Lattimore and Marcus Hutter. Near-optimal PAC bounds for discounted MDPs. *Theor. Comput. Sci.*, 558, 2014.

[Mannor and Tsitsiklis, 2004] Shie Mannor and John N. Tsitsiklis. The sample complexity of exploration in the multi-armed bandit problem. *J. Mach. Learn. Res.*, 5, 2004.

[Moore, 1956] Edward F. Moore. Gedanken-experiments on sequential machines. *Automata Studies*, 34, 1956.

[Palmer and Goldberg, 2007] Nick Palmer and Paul W. Goldberg. PAC-learnability of probabilistic deterministic finite state automata in terms of variation distance. *Theor. Comput. Sci.*, 387(1), 2007.

[Puterman, 1994] Martin L. Puterman. *Markov Decision Processes: Discrete stochastic dynamic programming*. Wiley, 1994.

[Ron *et al.*, 1998] Dana Ron, Yoram Singer, and Naftali Tishby. On the learnability and usage of acyclic probabilistic finite automata. *J. Comput. Syst. Sci.*, 56(2), 1998.

[Ronca and De Giacomo, 2021] Alessandro Ronca and Giuseppe De Giacomo. Efficient PAC reinforcement learning in regular decision processes. *CoRR*, abs/2105.06784, 2021.

[Singh and Yee, 1994] Satinder P. Singh and Richard C. Yee. An upper bound on the loss from approximate optimal-value functions. *Mach. Learn.*, 16(3), 1994.

[Strehl and Littman, 2005] Alexander L. Strehl and Michael L. Littman. A theoretical analysis of model-based interval estimation. In *ICML*, 2005.

[Strehl *et al.*, 2009] Alexander L. Strehl, Lihong Li, and Michael L. Littman. Reinforcement learning in finite MDPs: PAC analysis. *J. Mach. Learn. Res.*, 10, 2009.

[Sutton and Barto, 2018] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning*. MIT Press, 2018.

[Szita and Szepesvári, 2010] Istvan Szita and Csaba Szepesvári. Model-based reinforcement learning with nearly tight exploration complexity bounds. In *ICML*, 2010.

[Valiant, 1984] Leslie G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11), 1984.

[Wang *et al.*, 2020] Ruosong Wang, Simon S. Du, Lin F. Yang, and Sham M. Kakade. Is long horizon RL more difficult than short horizon RL? In *NeurIPS*, 2020.