# Provably efficient RL with Rich Observations via Latent State Decoding

Simon S. Du [1] Akshay Krishnamurthy [2] Nan Jiang [3] Alekh Agarwal [4] Miroslav Dudík [2] John Langford [2]

## Abstract

We study the exploration problem in episodic MDPs with rich observations generated from a small number of latent states. Under certain identifiability assumptions, we demonstrate how to estimate a mapping from the observations to latent states inductively through a sequence of regression and clustering steps—where previously decoded latent states provide labels for later regression problems—and use it to construct good exploration policies. We provide finite-sample guarantees on the quality of the learned state decoding function and exploration policies, and complement our theory with an empirical evaluation on a class of hard exploration problems. Our method exponentially improves over $Q$-learning with naïve exploration, even when $Q$-learning has cheating access to latent states.

## 1. Introduction

We study reinforcement learning (RL) in episodic environments with rich observations, such as images and texts. While many modern empirical RL algorithms are designed to handle such settings (see, e.g., Mnih et al., 2015), only few works study how to explore well in these environments (Ostrovski et al., 2017; Osband et al., 2016) and the sample efficiency of these techniques is not theoretically understood.

From a theoretical perspective, strategic exploration algorithms for provably sample-efficient RL have long existed in the classical tabular setting (Kearns & Singh, 2002; Brafman & Tennenholtz, 2002). However, these methods are difficult to adapt to rich observation spaces, because they all require a number of interactions polynomial in the number of *observed states*, and, without additional structural assumptions, such a dependency is unavoidable (see, e.g., Jaksch et al., 2010; Lattimore & Hutter, 2012). Consequently, treating the observations directly as unique states makes this class of methods unsuitable for most settings of practical interest.

In order to avoid the dependency on the observation space, one must exploit some inherent structure in the problem. The recent line of work on contextual decision processes (Krishnamurthy et al., 2016; Jiang et al., 2017; Dann et al., 2018) identified certain low-rank structures that enable exploration algorithms with sample complexity polynomial in the rank parameter. Such low-rank structure is crucial to circumventing information-theoretic hardness, and is typically found in problems where complex observations are emitted from a small number of *latent states*. Unlike tabular approaches, which require the number of states to be small and observed, these works are able to handle settings where the observation spaces are uncountably large or continuous and the underlying states never observed during learning. They achieve this by exploiting the low-rank structure implicitly, operating only in the observation space. The resulting algorithms are sample-efficient, but either provably computationally intractable, or practically quite cumbersome even under strong assumptions (Dann et al., 2018).

In this work, we take an alternative route: we recover the latent-state structure explicitly by learning a *decoding function* (from a large set of candidates) that maps a rich observation to the corresponding latent state; note that if such a function is learned perfectly, the rich-observation problem is reduced to a tabular problem where exploration is tractable. We show that our algorithms are:

*Provably sample-efficient:* Under certain identifiability assumptions, we recover a mapping from the observations to underlying latent states as well as a good exploration policy using a number of samples which is polynomial in the number of latent states, horizon and the complexity of the decoding function class with no explicit dependence on the observation space size. Thus we significantly generalize beyond the works of Dann et al. (2018) who require deterministic dynamics and Azizzadenesheli et al. (2016a) whose guarantees scale with the observation space size.

*Computationally practical:* Unlike many prior works in this vein, our algorithm is easy to implement and substantially outperforms naïve exploration in experiments, even when the baselines have cheating access to the latent states.

[1] Carnegie Mellon University [2] Microsoft Research, New York [3] University of Illinois at Urbana-Champaign [4] Microsoft Research, Redmond. Correspondence to: Simon S. Du <ssdu@cs.cmu.edu>.

*Proceedings of the $36^{th}$ International Conference on Machine Learning*, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

In the process, we introduce a formalism called *block Markov decision process* (also implicit in some prior works), and a new solution concept for exploration called $\epsilon$–policy cover.

The main challenge in learning the decoding function is that the hidden states are never directly observed. Our key novelty is the use of a backward conditional probability vector (Equation 1) as a representation for latent state, and learning the decoding function via conditional probability estimation, which can be solved using least squares regression. While learning a low-dimensional representations of rich observations has been explored in recent empirical works (e.g., Silver et al., 2017; Oh et al., 2017; Pathak et al., 2017), our work provides a precise mathematical characterization of the structures needed for such approaches to succeed and comes with rigorous sample-complexity guarantees.

## 2. Setting and Task Definition

We begin by introducing some basic notation. We write $[h]$ to denote the set $\{1, \ldots, h\}$. For any finite set $S$, we write $U(S)$ to denote the uniform distribution over $S$. We write $\triangle_d$ for the simplex in $\mathbb{R}^d$. Finally, we write $\|\cdot\|$ and $\|\cdot\|_1$, respectively, for the Euclidean and the $\ell_1$ norms of a vector.

### 2.1. Block Markov Decision Process

In this paper we introduce and analyze a *block Markov decision process* or *BMDP*. It refers to an environment described by a finite, but unobservable *latent state space* $\mathcal{S}$, a finite *action space* $\mathcal{A}$, with $|\mathcal{A}| = K$, and a possibly infinite, but observable *context space* $\mathcal{X}$. The dynamics of a BMDP is described by the *initial state* $s_1 \in \mathcal{S}$ and two conditional probability functions: the *state-transition function* $p$ and *context-emission function* $q$, defining conditional probabilities $p(s' \mid s, a)$ and $q(x \mid s)$ for all $s, s' \in \mathcal{S}$, $a \in \mathcal{A}$, $x \in \mathcal{X}$.[1]

The model may further include a distribution of reward conditioned on context and action. However, rewards do not play a role in the central task of the paper, which is the exploration of all latent states. Therefore, we omit rewards from our formalism, but we discuss in a few places how our techniques apply in the presence of rewards (for a thorough discussion see Appendix B).

We consider episodic learning tasks with a finite horizon $H$. In each episode, the environment starts in the state $s_1$. In the step $h \in [H]$ of an episode, the environment generates a context $x_h \sim q(\cdot|s_h)$, the agent observes the context $x_h$ (but not the state $s_h$), takes an action $a_h$, and the environment transitions to a new state $s_{h+1} \sim p(\cdot|s_h, a_h)$. The sequence $(s_1, x_1, a_1, \ldots, s_H, x_H, a_H, s_{H+1}, x_{H+1})$ generated in an episode is called a *trajectory*. We emphasize that a learning agent does not observe components $s_h$ from the trajectory.

So far, our description resembles that of a partially observable Markov decision process (POMDP). To finish the definition of BMDP, and distinguish it from a POMDP, we make the following assumption:

**Assumption 2.1** (Block structure). *Each context $x$ uniquely determines its generating state $s$. That is, the context space $\mathcal{X}$ can be partitioned into disjoint blocks $\mathcal{X}_s$, each containing the support of the conditional distribution $q(\cdot \mid s)$.*

The sets $\mathcal{X}_s$ are unique up to the sets of measure zero under $q(\cdot \mid s)$. In the paper, we say "*for all $x \in \mathcal{X}_s$*" to mean "*for all $x \in \mathcal{X}_s$ up to a set of measure zero under $q(\cdot \mid s)$*."

The block structure implies the existence of a *perfect decoding function* $f^* : \mathcal{X} \to \mathcal{S}$, which maps contexts into their generating states. This means that a BMDP is indeed an MDP with the transition operator $P(x' \mid x, a) = q\big(x' \mid f^*(x')\big)p\big(f^*(x') \mid f^*(x), a\big)$. Hence the contexts $x$ observed by the agent form valid Markovian states, but the size of $\mathcal{X}$ is too large, so only learning the MDP parameters in the smaller, latent space $\mathcal{S}$ is tractable.

The BMDP model is assumed in several prior works (e.g., Krishnamurthy et al., 2016; Azizzadenesheli et al., 2016a; Dann et al., 2018), without the explicit name. It naturally captures visual grid-world environments studied in empirical RL (e.g., Johnson et al., 2016), and also models noisy observations of the latent state due to imperfect sensors. While the block-structure assumption appears severe, it is necessary for efficient learning if the reward is allowed to depend arbitrarily on the latent state (cf. Propositions 1 and 2 of Krishnamurthy et al. 2016). In our experiments, we study the robustness of our algorithms to this assumption.

To streamline our analysis, we make a standard assumption for episodic settings. We assume that $\mathcal{S}$ can be partitioned into disjoints sets $\mathcal{S}_h$, $h \in [H+1]$, such that $p(\cdot \mid s, a)$ is supported on $\mathcal{S}_{h+1}$ whenever $s \in \mathcal{S}_h$. We refer to $h$ as the *level* and assume that it is observable as part of the context, so the context space is also partitioned into sets $\mathcal{X}_h$. We use notation $\mathcal{S}_{[h]} = \cup_{\ell \in [h]} \mathcal{S}_\ell$ for the set of states up to level $h$, and similarly define $\mathcal{X}_{[h]} = \cup_{\ell \in [h]} \mathcal{X}_\ell$.

We assume that $|\mathcal{S}_h| \leq M$. We seek learning algorithms that scale polynomially in parameters $M$, $K$ and $H$, but do not explicitly depend on $|\mathcal{X}|$, which might be infinite.

### 2.2. Solution Concept: Cover of Exploratory Policies

In this paper, we focus on the problem of exploration. Specifically, for each state $s \in \mathcal{S}$, we seek an agent strategy for reaching that state $s$. We formalize an agent strategy as an *$h$-step policy*, which is a map $\pi : \mathcal{X}_{[h]} \to \mathcal{A}$ specifying which action to take in each context up to step $h$. When executing an $h$-step policy $\pi$ with $h < H$, an agent acts

[1] For continuous context spaces, $q(\cdot \mid s)$ describes a density function relative to a suitable measure (e.g., Lebesgue measure).

according to $\pi$ for $h$ steps and then arbitrarily until the end of the episode (e.g., according to a specific default policy).

For an $h$-step policy $\pi$, we write $\mathbb{P}^\pi$ to denote the probability distribution over $h$-step trajectories induced by $\pi$. We write $\mathbb{P}^\pi(\mathcal{E})$ for the probability of an event $\mathcal{E}$. For example, $\mathbb{P}^\pi(s)$ is the probability of reaching the state $s$ when executing $\pi$.

We also consider randomized strategies, which we formalize as *policy mixtures*. An $h$-step *policy mixture* $\eta$ is a distribution over $h$-step policies. When executing $\eta$, an agent randomly draws a policy $\pi \sim \eta$ at the beginning of the episode, and then follows $\pi$ throughout the episode. The induced distribution over $h$-step trajectories is denoted $\mathbb{P}^\eta$.

Our algorithms create specific policies and policy mixtures via concatenation. Specifically, given an $h$-step policy $\pi$, we write $\pi \odot a$ for the $(h+1)$-step policy that executes $\pi$ for $h$ steps and chooses action $a$ in step $h+1$. Similarly, if $\eta$ is a policy mixture and $\nu$ a distribution over $\mathcal{A}$, we write $\eta \odot \nu$ for the policy mixture equivalent to first sampling and following a policy according to $\eta$ and then independently sampling and following an action according to $\nu$.

We finally introduce two key concepts related to exploration: *maximum reaching probability* and *policy cover*.

**Definition 2.1** (Maximum reaching probability.). *For any $s \in \mathcal{S}$, its* maximum reaching probability *$\mu(s)$ is*

$$\mu(s) := \max_\pi \mathbb{P}^\pi(s),$$

*where the maximum is taken over all maps $\mathcal{X}_{[H]} \to \mathcal{A}$. The policy attaining the maximum for a given $s$ is denoted $\pi_s^*$.*[2]

Without loss of generality, we assume that all the states are reachable, i.e., $\mu(s) > 0$ for all $s$. We write $\mu_{\min} = \min_{s \in \mathcal{S}} \mu(s)$ for the $\mu(s)$ value of the hardest-to-reach state. Since $\mathcal{S}$ is finite and all states are reachable, $\mu_{\min} > 0$.

Given maximum reaching probabilities, we formalize the task of finding policies that reach states $s$ as the task of finding an $\epsilon$*–policy cover* in the following sense:

**Definition 2.2** (Policy cover of the state space). *We say that a set of policies $\Pi_h$ is an $\epsilon$–*policy cover *of $\mathcal{S}_h$ if for all $s \in \mathcal{S}_h$ there exists an $(h-1)$-step policy $\pi \in \Pi_h$ such that $\mathbb{P}^\pi(s) \geq \mu(s) - \epsilon$. A set of policies $\Pi$ is an $\epsilon$–*policy cover *of $\mathcal{S}$ if it is an $\epsilon$–policy cover of $\mathcal{S}_h$ for all $h \in [H+1]$.*

Intuitively, we seek a policy cover of a small size, typically $O(|\mathcal{S}|)$, and with a small $\epsilon$. Given such a cover, we can reach every state with the largest possible probability (up to $\epsilon$) by executing each policy from the cover in turn. This enables us to collect a dataset of observations and rewards at all (sufficiently) reachable states $s$ and further obtain a policy that maximizes any reward (details in Appendix B).

## 3. Embedding Approach

A key challenge in solving the BMDP exploration problem is the lack of access to the latent state $s$. Our algorithms work by explicitly learning a decoding function $f$ which maps contexts to the corresponding latent states. This appears to be a hard unsupervised learning problem, even under the block-structure assumption, unless we make strong assumptions about the structure of $\mathcal{X}_s$ or about the emission distributions $q(\cdot \mid s)$. Here, instead of making assumptions about $q$ or $\mathcal{X}_s$, we make certain "separability" assumptions about the latent transition probabilities $p$. Thus, we retain a broad flexibility to model rich context spaces, and also obtain the ability to efficiently learn a decoding function $f$. In this section, we define key components of our approach and formally state the separability assumption.

### 3.1. Embeddings and Function Approximation

In order to construct the decoding function $f$, we learn low-dimensional representations of contexts as well as latent states in a shared space, namely $\Delta_{MK}$. We learn embedding functions $\mathbf{g} : \mathcal{X} \to \Delta_{MK}$ for contexts and $\boldsymbol{\phi} : \mathcal{S} \to \Delta_{MK}$ for states, with the goal that $\mathbf{g}(x)$ and $\boldsymbol{\phi}(s)$ should be close if and only if $x \in \mathcal{X}_s$. Such embedding functions always exist due to the block-structure: for any set of distinct vectors $\{\boldsymbol{\phi}(s)\}_{s \in \mathcal{S}}$, it suffices to define $\mathbf{g}(x) = \boldsymbol{\phi}(s)$ for $x \in \mathcal{X}_s$.

As we see later in this section, embedding functions $\boldsymbol{\phi}$ and $\mathbf{g}$ can be constructed via an essentially supervised approach, assuming separability. The state embedding $\boldsymbol{\phi}$ is a lower complexity object (a tuple of at most $|\mathcal{S}|$ points in $\Delta_{MK}$), whereas the context embedding $\mathbf{g}$ has a high complexity for even moderately rich context spaces. Therefore, as is standard in supervised learning, we limit attention to functions $\mathbf{g}$ from some class $\mathcal{G} \subseteq \{\mathcal{X} \to \Delta_{MK}\}$, such as generalized linear models, tree ensembles, or neural nets. This is a form of function approximation where the choice of $\mathcal{G}$ includes any inductive biases about the structure of the contexts. By limiting the richness of $\mathcal{G}$, we can generalize across contexts as well as control the sample complexity of learning. At the same time, $\mathcal{G}$ needs to include embedding functions that reflect the block structure. Allowing a separate $\mathbf{g}_h \in \mathcal{G}$ for each level, we require realizability in the following sense:

**Assumption 3.1** (Realizability). *For any $h \in [H+1]$ and $\boldsymbol{\phi} : \mathcal{S}_h \to \triangle_{MK}$, there exists $\mathbf{g}_h \in \mathcal{G}$ such that $\mathbf{g}_h(x) = \boldsymbol{\phi}(s)$ for all $x \in \mathcal{X}_s$ and $s \in \mathcal{S}_h$.*

In words, the class $\mathcal{G}$ must be able to match any state-embedding function $\boldsymbol{\phi}$ across all blocks $\mathcal{X}_s$. To satisfy this assumption, it is natural to consider classes $\mathcal{G}$ obtained via a composition $\boldsymbol{\phi}' \circ f$ where $f$ is a decoding function from some class $\mathcal{F} \subseteq \{\mathcal{X} \to \mathcal{S}\}$ and $\boldsymbol{\phi}'$ is any mapping $\mathcal{S} \to \triangle_{MK}$. Conceptually, $f$ first decodes the context $x$ to a state $f(x)$ which is then embedded by $\boldsymbol{\phi}'$ into $\triangle_{MK}$. The

[2] It suffices to consider maps $\mathcal{X}_{[h]} \to \mathcal{A}$ for $s \in \mathcal{S}_{h+1}$.

realizability assumption is satisfied as long as $\mathcal{F}$ contains a perfect decoding function $f^*$, for which $f^*(x) = s$ whenever $x \in \mathcal{X}_s$. The core representational power of $\mathcal{G}$ is thus driven by $\mathcal{F}$, the class of candidate decoding functions $f$.

Given such a class $\mathcal{G}$, our goal is find a suitable context-embedding function in $\mathcal{G}$ using a number of trajectories that is proportional to $\log|\mathcal{G}|$ when $\mathcal{G}$ is finite, or a more general notion of complexity such as a log covering number when $\mathcal{G}$ is infinite. Throughout this paper, we assume that $\mathcal{G}$ is finite as it serves to illustrate the key ideas, but our approach generalizes to the infinite case using standard techniques.

As we alluded to earlier, we learn context embeddings $\mathbf{g}_h$ by solving supervised learning problems. In fact, we only require the ability to solve least squares problems. Specifically, we assume access to an algorithm for solving vector-valued least-squares regression over the class $\mathcal{G}$. We refer to such an algorithm as the ERM oracle:

**Definition 3.1** (ERM Oracle). *Let $\mathcal{G}$ be a function class that maps $\mathcal{X}$ to $\triangle_{MK}$. An* empirical risk minimization oracle (ERM oracle) *for $\mathcal{G}$ is any algorithm that takes as input a data set $D = \{(x_i, \mathbf{y}_i)\}_{i=1}^n$ with $x_i \in \mathcal{X}$, $\mathbf{y}_i \in \triangle_{MK}$, and computes* $\operatorname{argmin}_{\mathbf{g}\in\mathcal{G}} \sum_{(x,\mathbf{y})\in D} \|\mathbf{g}(x) - \mathbf{y}\|^2$.

### 3.2. Backward Probability Vectors and Separability

For any distribution $\mathbb{P}$ over trajectories, we define *backward probabilities* as the conditional probabilities of the form $\mathbb{P}(s_{h-1}, a_{h-1} \mid s_h)$—note that conditioning is the opposite of transitions in $p$. For the backward probabilities to be defined, we do not need to fully specify a full distribution over trajectories, only a distribution $\nu$ over $(s_{h-1}, a_{h-1})$. For any such distribution $\nu$, any $s \in \mathcal{S}_{h-1}$, $a \in \mathcal{A}$ and $s' \in \mathcal{S}_h$, the backward probability is defined as

$$b_\nu(s, a \mid s') = \frac{p(s' \mid s, a)\,\nu(s, a)}{\sum_{\tilde{s},\tilde{a}} p(s' \mid \tilde{s}, \tilde{a})\,\nu(\tilde{s}, \tilde{a})}. \tag{1}$$

For a given $s' \in \mathcal{S}_h$, we collect the probabilities $b_\nu(s, a \mid s')$ across all $s \in \mathcal{S}_{h-1}$, $a \in \mathcal{A}$ into the *backward probability vector* $\mathbf{b}_\nu(s') \in \triangle_{MK}$, padding with zeros if $|\mathcal{S}_{h-1}| < M$. Backward probability vectors are at the core of our approach, because they correspond to the state embeddings $\boldsymbol{\phi}(s)$ approximated by our algorithms. Our algorithms require that $\mathbf{b}_\nu(s')$ for different states $s' \in \mathcal{S}_h$ be sufficiently separated from one another for a suitable choice of $\nu$:

**Assumption 3.2** ($\gamma$-Separability). *There exists $\gamma > 0$ such that for any $h \in \{2, \ldots, H+1\}$ and any distinct $s', s'' \in \mathcal{S}_h$, the backward probability vectors with respect to the uniform distribution are separated by a margin of at least $\gamma$, i.e., $\|\mathbf{b}_\nu(s') - \mathbf{b}_\nu(s'')\|_1 \geq \gamma$, where $\nu = U(\mathcal{S}_{h-1} \times \mathcal{A})$.*

We show in Section 4.1 that this assumption is automatically satisfied with $\gamma = 2$ when latent-state transitions are deterministic (as assumed, e.g., by Dann et al., 2018). However, the class of $\gamma$-separable models is substantially larger. In Appendix F we show that the uniform distribution in the assumption can be replaced with any distribution supported on $\mathcal{S}_{h-1} \times \mathcal{A}$, although the margins $\gamma$ would be different.

The key property that makes vectors $\mathbf{b}_\nu(s')$ algorithmically useful is that they arise as solutions to a specific least squares problem with respect to data generated by a policy whose marginal distribution over $(s_{h-1}, a_{h-1})$ matches $\nu$. Let $\mathbf{e}_{(s,a)}$ denote the vector of the standard basis in $\mathbb{R}^{MK}$ corresponding to the coordinate indexed by $(s, a) \in \mathcal{S}_{h-1} \times \mathcal{A}$. Then the following statement holds:

**Theorem 3.1.** *Let $\nu$ be a distribution supported on $\mathcal{S}_{h-1} \times \mathcal{A}$ and let $\tilde{\nu}$ be a distribution over $(s, a, x')$ defined by sampling $(s, a) \sim \nu$, $s' \sim p(\cdot \mid s, a)$, and $x' \sim q(\cdot \mid s')$. Let*

$$\mathbf{g}_h \in \operatorname*{argmin}_{\mathbf{g}\in\mathcal{G}} \mathbb{E}_{\tilde{\nu}}\left[\left\|\mathbf{g}(x') - \mathbf{e}_{(s,a)}\right\|^2\right]. \tag{2}$$

*Then, under Assumption 3.1, every minimizer $\mathbf{g}_h$ satisfies $\mathbf{g}_h(x') = \mathbf{b}_\nu(s')$ for all $x' \in \mathcal{X}_{s'}$ and $s' \in \mathcal{S}_h$.*

The distribution $\tilde{\nu}$ is exactly the marginal distribution induced by a policy whose marginal distribution over $(s_{h-1}, a_{h-1})$ matches $\nu$. Any minimizer $\mathbf{g}_h$ yields context embeddings corresponding to state embeddings $\boldsymbol{\phi}(s') = \mathbf{b}_\nu(s')$. Our algorithms build on Theorem 3.1: they replace the expectation by an empirical sample and obtain an approximate minimizer $\hat{\mathbf{g}}_h$ by invoking an ERM oracle.

## 4. Algorithm for Separable BMDPs

With the main components defined, we can now derive our algorithm for learning a policy cover in a separable BMDP.

The algorithm proceeds inductively, level by level. On each level $h$, we learn the following objects:

- The set of discovered latent states $\widehat{\mathcal{S}}_h \subseteq [M]$ and a decoding function $\hat{f}_h : \mathcal{X} \to \widehat{\mathcal{S}}_h$, which allows us to identify latent states at level $h$ from observed contexts.
- The estimated transition probabilities $\hat{p}(\hat{s}_h \mid \hat{s}_{h-1}, a)$ across all $\hat{s}_{h-1} \in \widehat{\mathcal{S}}_{h-1}$, $a \in \mathcal{A}$, $\hat{s}_h \in \widehat{\mathcal{S}}_h$.
- A set of $(h-1)$-step policies $\Pi_h = \{\pi_{\hat{s}}\}_{\hat{s}\in\widehat{\mathcal{S}}_h}$.

We establish a correspondence between the discovered states and true states via a bijection $\alpha_h$, under which the functions $\hat{f}_h$ accurately decode contexts into states, the probability estimates $\hat{p}$ are close to true probabilities, and $\Pi_h$ is an $\epsilon$–policy cover of $\mathcal{S}_h$. Specifically, we prove the following statement for suitable accuracy parameters $\epsilon_{\text{f}}$, $\epsilon_{\text{p}}$ and $\epsilon$:

**Claim 4.1.** *There exists a bijection $\alpha_h : \widehat{\mathcal{S}}_h \to \mathcal{S}_h$ such that the following conditions are satisfied for all $\hat{s} \in \widehat{\mathcal{S}}_{h-1}$, $a \in \mathcal{A}$, $\hat{s}' \in \widehat{\mathcal{S}}_h$, and $s = \alpha_{h-1}(\hat{s}_{h-1})$, $s' = \alpha_h(\hat{s}')$, where $\alpha_{h-1}$ is the bijection for the previous level:*

$$\textit{Accuracy of } \hat{f}_h\textit{:} \quad \mathbb{P}_{x'\sim q(\cdot|s')}\left[\hat{f}_h(x') = \hat{s}'\right] \geq 1 - \epsilon_{\text{f}}, \tag{3}$$

**Algorithm 1** PCID (**P**olicy **C**over via **I**nductive **D**ecoding)

```
1:  Input:
        $N_{\mathrm{g}}$: sample size for learning context embeddings
        $N_\phi$: sample size for learning state embeddings
        $N_{\mathrm{p}}$: sample size for estimating transition probabilities
        $\tau > 0$: a clustering threshold for learning latent states
2:  Output: policy cover $\Pi = \Pi_1 \cup \cdots \cup \Pi_{H+1}$
3:  Let $\widehat{\mathcal{S}}_1 = \{s_1\}$. Let $\hat{f}_1(x) = s_1$ for all $x \in \mathcal{X}$.
4:  Let $\Pi_1 = \{\pi_0\}$ where $\pi_0$ is the trivial 0-step policy.
5:  Initialize $\hat{p}$ to an empty mapping.
6:  for $h = 2, \dots, H+1$ do
7:      Let $\eta_h = U(\Pi_{h-1}) \odot U(\mathcal{A})$
8:      Execute $\eta_h$ for $N_{\mathrm{g}}$ times. $D_{\mathrm{g}} = \{\hat{s}^i_{h-1}, a^i_{h-1}, x^i_h\}_{i=1}^{N_{\mathrm{g}}}$
          for $\hat{s}_{h-1} = \hat{f}_{h-1}(x_{h-1})$.
9:      Learn $\hat{\mathbf{g}}_h$ by calling ERM oracle on input $D_{\mathrm{g}}$:
          $\hat{\mathbf{g}}_h = \mathrm{argmin}_{\mathbf{g} \in \mathcal{G}} \sum_{(\hat{s},a,x') \in D_{\mathrm{g}}} \left\| \mathbf{g}(x') - \mathbf{e}_{(\hat{s},a)} \right\|^2$.
10:     Execute $\eta_h$ for $N_\phi$ times. $\mathcal{Z} = \{\hat{\mathbf{z}}_i = \hat{\mathbf{g}}_h(x^i_h)\}_{i=1}^{N_\phi}$.
11:     Learn $\widehat{\mathcal{S}}_h$ and the state embedding map $\widehat{\boldsymbol{\phi}}_h : \widehat{\mathcal{S}}_h \to \mathcal{Z}$
          by clustering $\mathcal{Z}$ with threshold $\tau$ (see Algorithm 2).
12:     Define $\hat{f}_h(x') = \mathrm{argmin}_{\hat{s} \in \widehat{\mathcal{S}}_h} \left\| \widehat{\boldsymbol{\phi}}(\hat{s}) - \hat{\mathbf{g}}_h(x') \right\|_1$.
13:     Execute $\eta_h$ for $N_{\mathrm{p}}$ times. $D_{\mathrm{p}} = \{\hat{s}^i_{h-1}, a^i_{h-1}, \hat{s}^i_h\}_{i=1}^{N_{\mathrm{p}}}$
          for $\hat{s}_{h-1} = \hat{f}_{h-1}(x_{h-1})$, $\hat{s}_h = \hat{f}_h(x_h)$.
14:     Define $\hat{p}(\hat{s}_h \mid \hat{s}_{h-1}, a_{h-1})$
          equal to empirical conditional probabilities in $D_{\mathrm{p}}$.
15:     for $\hat{s}' \in \widehat{\mathcal{S}}_h$ do
16:         Run Algorithm 3 with inputs $\hat{p}$ and $\hat{s}'$
              to obtain $(h-1)$-step policy $\psi_{\hat{s}'} : \widehat{\mathcal{S}}_{[h-1]} \to \mathcal{A}$.
17:         Set $\pi_{\hat{s}'}(x_\ell) = \psi_{\hat{s}'}(\hat{f}_\ell(x_\ell))$, $\ell \in [h-1]$, $x_\ell \in \mathcal{X}_\ell$.
18:     end for
19:     Let $\Pi_h = (\pi_{\hat{s}})_{\hat{s} \in \widehat{\mathcal{S}}_h}$.
20: end for
```

*Accuracy of* $\hat{p}$:

$$\sum_{\hat{s}'' \in \widehat{\mathcal{S}}_h,\ s'' = \alpha_h(\hat{s}'')} \left| \hat{p}(\hat{s}'' \mid \hat{s}, a) - p(s'' \mid s, a) \right| \le \epsilon_{\mathrm{p}}, \quad (4)$$

*Coverage by* $\Pi_h$*:* $\mathbb{P}^{\pi_{\hat{s}'}}(s') \ge \mu(s') - \epsilon$. (5)

Algorithm 1 constructs $\widehat{\mathcal{S}}_h$, $\hat{f}_h$, $\hat{p}$ and $\Pi_h$ level by level. Given these objects up to level $h-1$, the construction for the next level $h$ proceeds in the following three steps, annotated with the lines in Algorithm 1 where they appear:

**(1) Regression step: learn** $\hat{\mathbf{g}}_h$ (lines 7–9). We collect a dataset of trajectories by repeatedly executing a specific policy mixture $\eta_h$. We use $\hat{f}_{h-1}$ to identify $\hat{s}_{h-1} = \hat{f}_{h-1}(x_{h-1})$ on each trajectory, obtaining samples $(\hat{s}_{h-1}, a_{h-1}, x_h)$ from $\tilde{\nu}$ induced by $\eta_h$. The context embedding $\hat{\mathbf{g}}_h$ is then obtained by solving the empirical version of (2).

**Algorithm 2** Clustering to Find Latent-state Embeddings.

```
Input: Data points $\mathcal{Z} = \{\mathbf{z}_i\}_{i=1}^n$ and threshold $\tau > 0$.
Output: Cluster indices $\widehat{\mathcal{S}}$ and centers $\widehat{\boldsymbol{\phi}} : \widehat{\mathcal{S}} \to \mathcal{Z}$.
Let $\widehat{\mathcal{S}} = \emptyset$, $k = 0$ (number of clusters).
while $\mathcal{Z} \ne \emptyset$ do
    Pick any $\mathbf{z} \in \mathcal{Z}$ (a new cluster center).
    Let $\mathcal{Z}' = \{\mathbf{z}' \in \mathcal{Z} :\ \|\mathbf{z} - \mathbf{z}'\|_1 \le \tau\}$.
    Add cluster: $k \leftarrow k+1$, $\widehat{\mathcal{S}} \leftarrow \widehat{\mathcal{S}} \cup \{k\}$, $\widehat{\boldsymbol{\phi}}(k) = \mathbf{z}$.
    Remove the newly covered points: $\mathcal{Z} \leftarrow \mathcal{Z} \setminus \mathcal{Z}'$.
end while
```

Our specific choice of $\eta_h$ ensures that each state $s_{h-1}$ is reached with probability at least $(\mu_{\min} - \epsilon)/M$, which is bounded away from zero if $\epsilon$ is sufficiently small. The uniform choice of actions then guarantees that each state on the next level is also reached with sufficiently large probability.

**(2) Clustering step: learn** $\widehat{\boldsymbol{\phi}}$ **and** $\hat{f}_h$ (lines 10–12). Thanks to Theorem 3.1, we expect that $\hat{\mathbf{g}}_h(x') \approx \mathbf{g}_h(x') = \mathbf{b}_\nu(s')$ for the distribution $\nu(\hat{s}_{h-1}, a_{h-1})$ induced by $\eta_h$.[3] Thus, all contexts $x'$ generated by the same latent state $s'$ have embedding vectors $\hat{\mathbf{g}}_h(x')$ close to each other and to $\mathbf{b}_\nu(s')$. Thanks to separability, we can therefore use clustering to identify all contexts generated by the same latent state, and this procedure is sample-efficient since the embeddings are low-dimensional vectors. Each cluster corresponds to some latent state $s'$ and any vector $\hat{\mathbf{g}}_h(x')$ from that cluster can be used to define the state embedding $\widehat{\boldsymbol{\phi}}(s')$. The decoding function $\hat{f}_h$ is defined to map any context $x'$ to the state $s'$ whose embedding $\widehat{\boldsymbol{\phi}}(s')$ is the closest to $\hat{\mathbf{g}}_h(x')$.

**(3) Dynamic programming: construct** $\Pi_h$ (lines 13–19). Finally, with the ability to identify states at level $h$ via $\hat{f}_h$, we can use collected trajectories to learn an approximate transition model $\hat{p}(\hat{s}' \mid \hat{s}, a)$ up to level $h$. This allows us to use dynamic programming to find policies that (approximately) optimize the probability of reaching any specific state $s' \in \mathcal{S}_h$. The dynamic programming finds policies $\psi_{\hat{s}'}$ that act by directly observing decoded latent states. The policies $\pi_{\hat{s}'}$ are obtained by composing $\psi_{\hat{s}'}$ with the decoding functions $\{\hat{f}_\ell\}_{\ell \in [h-1]}$.

The next theorem guarantees that with a polynomial number of samples, Algorithm 1 finds a small $\epsilon$–policy cover.[4]

**Theorem 4.1** (Sample Complexity of Algorithm 1)**.** *Fix any* $\epsilon = O\left(\frac{\mu^3_{\min}\gamma}{M^4 K^3 H}\right)$ *and a failure probability* $\delta > 0$*. Set* $N_{\mathrm{g}} = \tilde{\Omega}\left(\frac{M^4 K^4 H \log|\mathcal{G}|}{\epsilon \mu^3_{\min} \gamma^2}\right)$, $N_\phi = \tilde{\Theta}\left(\frac{MK}{\mu_{\min}}\right)$, $N_{\mathrm{p}} = \tilde{\Omega}\left(\frac{M^2 K H^2}{\mu_{\min}\epsilon^2}\right)$, $\tau = \frac{\gamma}{30MK}$*. Then with probability at least* $1-\delta$*, Algorithm 1 returns an* $\epsilon$*–policy cover of* $\mathcal{S}$*, with size at most* $MH$*.*

[3] Theorem 3.1 uses distributions $\nu$ and $\tilde{\nu}$ over true states $s_{h-1}$, but its analog also holds for distributions over $\hat{s}_{h-1}$, as long as decoding is approximately correct at the previous level.

[4] The $\tilde{O}(\cdot)$, $\tilde{\Omega}(\cdot)$, and $\tilde{\Theta}(\cdot)$ notation suppresses factors that are polynomial in $\log M$, $\log K$, $\log H$ and $\log(1/\delta)$.

**Algorithm 3** Dynamic Programming for Reaching a State

```
Input: target state $\hat{s}^* \in \widehat{\mathcal{S}}_h$,
       transition probabilities $\hat{p}(\hat{s}' \mid \hat{s}, a)$
         for all $\hat{s} \in \widehat{\mathcal{S}}_\ell$, $a \in \mathcal{A}$, $\hat{s}' \in \widehat{\mathcal{S}}_{\ell+1}$, $\ell \in [h-1]$.
Output: policy $\psi : \widehat{\mathcal{S}}_{[h-1]} \to \mathcal{A}$ maximizing $\hat{\mathbb{P}}^\psi(\hat{s}^*)$.
Let $v(\hat{s}^*) = 1$ and let $v(\hat{s}) = 0$ for all other $\hat{s} \in \widehat{\mathcal{S}}_h$.
for $\ell = h-1, h-2, \ldots, 1$ do
    for $\hat{s} \in \widehat{\mathcal{S}}_\ell$ do
        $\psi(\hat{s}) = \max_{a \in \mathcal{A}} \left[\sum_{\hat{s}' \in \widehat{\mathcal{S}}_{\ell+1}} v(\hat{s}')\, \hat{p}(\hat{s}' \mid \hat{s}, a)\right]$.
        $v(\hat{s}) = \sum_{\hat{s}' \in \mathcal{S}_{\ell+1}} v(\hat{s}')\, \hat{p}(\hat{s}' \mid \hat{s}, a = \psi(\hat{s}))$.
    end for
end for
```

**Algorithm 4** PCID for Deterministic BMDPs

```
1:  Input:
      $N_\mathrm{g}$: sample size for learning context embeddings
      $N_\mathrm{b}$: sample size for boosting embedding accuracy
      $\tau > 0$: a clustering threshold for learning latent states
2:  Output: policy cover $\Pi = \Pi_1 \cup \cdots \cup \Pi_{H+1}$
3:  Let $\widehat{\mathcal{S}}_1 = \{s_1\}$. Let $\Pi_1 = \{\pi_0\}$ for the 0-step policy $\pi_0$.
4:  for $h = 2, \ldots, H+1$ do
5:      Let $\eta_h = U(\Pi_{h-1}) \odot U(\mathcal{A})$
6:      Execute $\eta_h$ for $N_\mathrm{g}$ times. $D_\mathrm{g} = \{\hat{s}^i_{h-1}, a^i_{h-1}, x^i_h\}_{i=1}^{N_\mathrm{g}}$
          where $\hat{s}_{h-1}$ is the index of $\pi_{\hat{s}_{h-1}}$ sampled by $\eta_h$.
7:      Learn $\hat{\mathbf{g}}_h$ by calling the ERM oracle on input $D_\mathrm{g}$:
          $\hat{\mathbf{g}}_h = \mathrm{argmin}_{\mathbf{g} \in \mathcal{G}} \sum_{(\hat{s}, a, x') \in D_\mathrm{g}} \left\| \mathbf{g}(x') - \mathbf{e}_{(\hat{s}, a)} \right\|^2$.
8:      Initialize $\mathcal{Z} = \emptyset$ (dataset for learning latent states).
9:      for $(\pi, a) \in \Pi_{h-1} \times \mathcal{A}$ do
10:         Execute $\pi \odot a$ for $N_\mathrm{b}$ times. $D_\mathrm{b} = \{x^i_h\}_{i=1}^{N_\mathrm{b}}$.
11:         Set $\mathbf{z}_{\pi \odot a} = \sum_{x \in D_\mathrm{b}} \hat{\mathbf{g}}_h(x) / |D_\mathrm{b}|$, add $\mathbf{z}_{\pi \odot a}$ to $\mathcal{Z}$.
12:     end for
13:     Learn $\widehat{\mathcal{S}}_h$ and the state embedding map $\widehat{\boldsymbol{\phi}}_h : \widehat{\mathcal{S}}_h \to \mathcal{Z}$
          by clustering $\mathcal{Z}$ with threshold $\tau$ (see Algorithm 2).
14:     Set $\Pi_h = (\pi_{\hat{s}})_{\hat{s} \in \widehat{\mathcal{S}}_t}$ where $\pi_{\hat{s}} = \pi \odot a$ if $\widehat{\boldsymbol{\phi}}_h(\hat{s}) = \mathbf{z}_{\pi \odot a}$.
15: end for
```

In addition to dependence on the usual parameters like $M, K, H$ and $1/\epsilon$, our sample complexity also scales inversely with the separability margin $\gamma$ and the worst-case reaching probability $\mu_{\min}$. While the exact dependence on these parameters is potentially improvable, Appendix F suggest that some inverse dependence is unavoidable for our approach. Compared with Azizzadenesheli et al. (2016a), there is no explicit dependence on $|\mathcal{X}|$, although they make spectral assumptions instead of the explicit block structure.

### 4.1. Deterministic BMDPs

As a special case of general BMDPs, many prior works study the case of deterministic transitions, that is, $p(s' \mid s, a) = 1$ for a unique state $s'$ for each $s, a$. Also, many simulation-based empirical RL benchmarks exhibit this property. We refer to these BMDPs as deterministic, but note that only the transitions $p$ are deterministic, not the emissions $q$. In this special case, the algorithm and guarantees of the previous section can be improved, and we present this specialization here, both for a direct comparison with prior work and potential usability in deterministic environments.

To start, note that $\mu_{\min} = 1$ and $\gamma = 2$ in any deterministic BDMP. The former holds as any reachable state is reached with probability one. For the latter, if $(s, a)$ transitions to $s'$, then $(s, a)$ cannot appear in the backward distribution of any other state $s''$. Consequently, the backward probabilities for distinct states $s' \in \mathcal{S}_h$ must have disjoint support over $(s, a) \in \mathcal{S}_{h-1} \times \mathcal{A}$, and thus their $\ell_1$ distance is exactly two.

Deterministic transitions allow us to obtain the policy cover with $\epsilon = 0$; that is, we learn policies that are guaranteed to reach any given state $s$ with probability one. Moreover, it suffices to consider policies with simple structure: those that execute a fixed sequence of actions. Also, since we have access to policies reaching states in the prior level with probability one, there is no need for a decoding function $\hat{f}_{h-1}$ when learning states and context embeddings on level $h$. The final, more technical implication of determinism (which we explain below) is that it allows us to boost the accuracy of the context embedding in the clustering step, leading to improved sample complexity.

The details are presented in Algorithm 4. At each level $h \in [H+1]$, we construct the following objects:

- A set of discovered states $\widehat{\mathcal{S}}_h$.
- A set of $(h-1)$-step policies $\Pi_h = \{\pi_{\hat{s}}\}_{\hat{s} \in \widehat{\mathcal{S}}_h}$.

We proceed inductively and for each level $h$ prove that the following claim holds with a high probability:

**Claim 4.2.** *There exists a bijection $\alpha_h : \widehat{\mathcal{S}}_h \to \mathcal{S}_h$ such that $\pi_{\hat{s}}$ reaches $\alpha_h(\hat{s})$ with probability one.*

This implies that $\widehat{\mathcal{S}}_h$ can be viewed as a latent state space, and $\Pi_h$ is an $\epsilon$–policy cover of $\mathcal{S}_h$ with $\epsilon = 0$.

To construct these objects for next level $h$, Algorithm 4 proceeds in three steps similar to Algorithm 1 for the stochastic case. The regression step, that is, learning of $\hat{\mathbf{g}}_h$ (lines 5–7), is identical. The clustering step (lines 8–13) is slightly more complicated. We boost the accuracy of the learned context embedding $\hat{\mathbf{g}}_h$ by repeatedly sampling contexts that are guaranteed to be emitted from the same latent state (because they result from the same sequence of actions), and taking an average. This step allows us to get away with a lower accuracy of $\hat{\mathbf{g}}_h$ compared with Algorithm 1. Finally, the third step, learning of $\Pi_h$ (line 14), is substantially simpler. Since any action sequence reaching a given cluster can be picked as a policy to reach the corresponding latent state, dynamic programming is not needed.

The following theorem characterizes the sample complexity of Algorithm 4. It shows we only need $\tilde{O}\big(M^2K^2H\log|\mathcal{G}|\big)$ samples to find a policy cover with $\epsilon = 0$.

**Theorem 4.2** (Sample Complexity of Algorithm 4)**.** *Set $\tau = 0.01$, $N_{\rm g} = \tilde{\Omega}(M^2K^2\log|\mathcal{G}|)$ and $N_{\rm b} = \tilde{\Omega}(MK)$. Then with probability at least $1-\delta$, Algorithm 4 returns an $\epsilon$–policy cover of $\mathcal{S}$, with $\epsilon = 0$ and size at most $MH$.*

In Appendix B, we discuss how to use policy cover to optimize a reward. For instance, if the reward depends on the latent state, the policy cover enables us to reach each state-action pair and collect $O(1/\epsilon^2)$ samples to estimate this pair's expected reward up to $\epsilon$ accuracy. Thus, using $O(MKH/\epsilon^2)$ samples in addition to those needed by Algorithm 4, we can find the trajectory with the largest expected reward within an $H\epsilon$ error. To summarize:

**Corollary 4.1.** *With probability at least $1-\delta$, Algorithm 4 can be used to find an $\epsilon$-suboptimal policy using at most $\tilde{O}\big(M^2K^2H\log|\mathcal{G}| + MKH^3/\epsilon^2\big)$ trajectories from a deterministic BMDP.*

This corollary (proved in Appendix D as Corollary D.1) significantly improves over the prior bound $O(M^3H^8K/\epsilon^5)$ obtained by Dann et al. (2018), although their function-class complexity term is not directly comparable to ours, as their work approximates optimal value functions and policies, while we approximate ideal decoding functions.

## 5. Experiments

We perform an empirical evaluation of our decoding-based algorithms in six challenging RL environments, with two choices of the function class $\mathcal{G}$. We compare our algorithm, which operates directly on rich observations, against two tabular algorithms, which operate on the latent state: a sanity-check baseline and a near-optimal skyline. Some of the environments meet the BMDP assumptions and some do not; the former validate our theoretical results, while the latter demonstrate our algorithm's robustness. Our code is available at *https://github.com/Microsoft/StateDecoding*.

**The environments.** All environments share the same latent structure, and are a form of a "combination lock," with $H$ levels, 3 states per level, and 4 actions. Non-zero reward is only achievable from states $s_{1,h}$ and $s_{2,h}$. From $s_{1,h}$ and $s_{2,h}$ one action leads with probability $1-\alpha$ to $s_{1,h+1}$ and with probability $\alpha$ to $s_{2,h+1}$, another has the flipped behavior, and the remaining two lead to $s_{3,h+1}$. All actions from $s_{3,h}$ lead to $s_{3,h+1}$. The "good" actions are randomly assigned for every state. From $s_{1,H}$ and $s_{2,H}$, two actions receive Ber$(1/2)$ reward; all others provide zero reward. The start state is $s_{1,1}$. We consider deterministic variant ($\alpha = 0$) and stochastic variant ($\alpha = 0.1$). (See Appendix C.)

The environments are designed to be difficult for exploration. For example, the deterministic variant has $2^H$ paths with non-zero reward, but $4^H$ paths in total, so random exploration requires exponentially many trajectories.

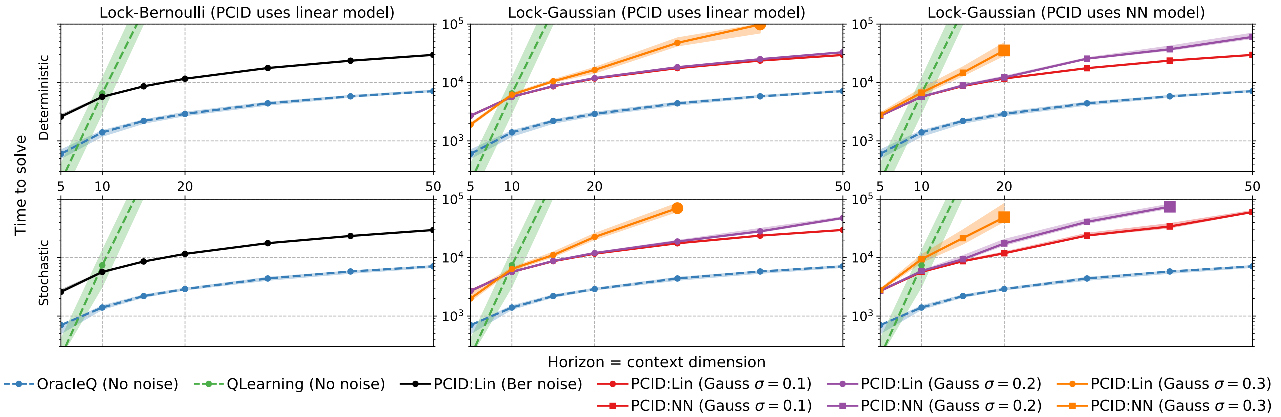
We also consider two observation processes, which we use only for our algorithm, while the baseline and the skyline operate directly on the latent state space. In *Lock-Bernoulli*, the observation space is $\{0,1\}^{H+3}$ where the first 3 coordinates are reserved for one-hot encoding of the state and the last $H$ coordinates are drawn i.i.d. from Ber$(1/2)$. The observation space is not partitioned across time, which our algorithms track internally. Thus, *Lock-Bernoulli* meets the BMDP assumptions and can be perfectly decoded via linear functions. In *Lock-Gaussian*, the observation space is $\mathbb{R}^{H+3}$. As before the first 3 coordinates are reserved for one-hot encoding of the state, but this encoding is corrupted with Gaussian noise. Formally, if the agent is at state $s_{i,h}$ the observation is $\mathbf{e}_i + \mathbf{v} \in \mathbb{R}^{3+H}$, where $\mathbf{e}_i$ is one of the first three standard basis vectors and $\mathbf{v}$ has $\mathcal{N}(0,\sigma^2)$ entries. We consider $\sigma \in \{0.1, 0.2, 0.3\}$. Note that Lock-Gaussian *does not* satisfy Assumption 2.1 since the emission distributions cannot be perfectly separated. We use this environment to evaluate the robustness of our algorithm to violated assumptions.

**Baseline, skyline, hyperparameters.** We compare our algorithm against two *tabular* approaches that cheat by directly accessing the latent state. The first, ORACLEQ, is the Optimistic $Q$-Learning algorithm of Jin et al. (2018), with a near-optimal regret in tabular environments.[5] Because of its near-optimality and direct access to the latent state, we do not expect any algorithm to beat ORACLEQ, and view it as a skyline. The second, QLEARNING, is tabular $Q$-learning with $\epsilon$-greedy exploration. It serves as a sanity-check baseline: any algorithm with strategic exploration should vastly outperform QLEARNING, even though it is cheating.

Each algorithm has two hyperparameters that we tune. In our algorithm (PCID), we use $k$-means clustering instead of Algorithm 2, so one of the hyperparameters is the number of clusters $k$. The second one is the number of trajectories $n$ to collect in each outer iteration. For ORACLEQ, these are the learning rate $\alpha$ and a confidence parameter $c$. For QLEARNING, these are the learning rate $\alpha$ and $\epsilon_{\text{frac}} \in [0,1]$, a fraction of the 100K episodes over which to anneal the exploration probability linearly from 1 down to 0.01.

For both *Lock-Bernoulli* and *Lock-Gaussian*, we experiment with linear decoding functions, which we fit via ordinary least squares. For *Lock-Gaussian* only, we also use two-layer neural networks. Specifically, these functions are of the form $f(\mathbf{x}) = \mathbf{W}_2^\top \,\text{sigmoid}(\mathbf{W}_1^\top\mathbf{x} + \mathbf{c})$ with the standard sigmoid activation, where the inner dimension is set to the clustering hyper-parameter $k$. These networks are

[5] We use the Hoeffding version, which is conceptually much simpler, but statistically slightly worse.



*Note: Larger markers mean that the next point is off the plot.*

*Figure 1.* Time-to-solve against problem difficulty for the combination lock environment with two observation processes and two function approximation classes. Left: *Lock-Bernoulli* with linear functions. Center: *Lock-Gaussian* with linear functions. Right: *Lock-Gaussian* with neural networks. Top row: deterministic latent transitions. Bottom row: stochastic transitions with switching probability 0.1. ORACLEQ and QLEARNING are cheating and operate directly on latent states.

trained using AdaGrad with a fixed learning rate of $0.1$, for a maximum of 5K iterations. See Appendix C for more details on hyperparameters and training.

**Experimental setup.** We run the algorithms on all environments with varying $H$, which also influences the dimension of the observation space. Each algorithm runs for 100K episodes and we say that it has *solved the lock* by episode $t$ if at round $t$ its running-average reward is $\geq 0.25 = 0.5V^\star$. The *time-to-solve* is the smallest $t$ for which the algorithm has solved the lock. For each hyperparameter, we run 25 replicates with different randomizations of the environment and seeds, and we plot the median time-to-solve of the best hyperparameter setting (along with error bands corresponding to $90^{\text{th}}$ and $10^{\text{th}}$ percentiles) against the horizon $H$. Our algorithm is reasonably fast, e.g., a single replicate of the above protocol for the two-layer neural net model and $H = 50$ takes less than 10 minutes on a standard laptop.

**Results.** The results are in Figure 1 in a log-linear plot. First, QLEARNING works well for small horizon problems but cannot solve problems with $H \geq 15$ within 100K episodes, which is not surprising.[6] The performance curve for QLEARNING is linear, revealing an exponential sample complexity, and demonstrating that these environments cannot be solved with naïve exploration. As a second observation, ORACLEQ performs extremely well, and as we verify in Appendix C demonstrates a linear scaling with $H$.[7]

In *Lock-Bernoulli*, PCID is roughly a factor of 5 worse than the skyline ORACLEQ for all values of $H$, but the curves have similar behavior. In Appendix C, we verify a *near-linear* scaling with $H$, even better than predicted by our theory. Of course PCID is an exponential improvement over QLEARNING with $\epsilon$-greedy exploration here.

In *Lock-Gaussian* with linear functions, the results are similar for the low-noise setting. The performance of PCID degrades as the noise level increases. For example, with noise level $\sigma = 0.3$, it fails to solve the stochastic problem with $H = 40$ in 100K episodes. This is expected, as Assumption 2.1 is severely violated at this noise level. However, the scaling of the sampling complexity still represents a dramatic improvement over QLEARNING.

Finally, PCID with neural networks is less robust to noise and stochasticity in *Lock-Gaussian*. Here, with $\sigma = 0.3$ the algorithm is unable to solve the $H = 30$ problem, both with and without stochasticity, but still does quite well with $\sigma \in \{0.1, 0.2\}$. The scaling with $H$ is still quite favorable.

**Sensitivity analysis.** We also perform a simple sensitivity analysis to assess how the hyperparameters $k$ and $n$ influence the behavior of PCID. We find that if we underestimate either $k$ or $n$ the algorithm fails, either because it cannot identify all latent states, or it does not collect enough data to solve the regression problems. On the other hand, the algorithm is quite robust to over-estimating both parameters. (See Appendix C.3 for further details.)

**Summary.** We have shown on several rich-observation environments with both linear and non-linear functions that PCID scales to large-horizon rich-observation problems. It dramatically outperforms tabular QLEARNING with $\epsilon$-greedy exploration, and is roughly a factor of 5 worse than a near-optimal ORACLEQ with an access to the latent state. PCID's performance is robust to hyperparameter choices and degrades gracefully as the assumptions are violated.

[6] We actually ran QLEARNING for 1M episodes and found it solves $H = 15$ with 170K episodes.

[7] This is incomparable with the result in Jin et al. (2018) since we are not measuring regret here.

## References

Antos, A., Szepesvári, C., and Munos, R. Learning near-optimal policies with bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning*, 2008.

Azizzadenesheli, K., Lazaric, A., and Anandkumar, A. Reinforcement learning of POMDPs using spectral methods. In *Conference on Learning Theory*, 2016a.

Azizzadenesheli, K., Lazaric, A., and Anandkumar, A. Reinforcement learning in rich-observation MDPs using spectral methods. *arxiv:1611.03907*, 2016b.

Bagnell, J. A., Kakade, S. M., Schneider, J. G., and Ng, A. Y. Policy search by dynamic programming. In *Advances in Neural Information Processing Systems*, 2004.

Brafman, R. I. and Tennenholtz, M. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 2002.

Dann, C., Jiang, N., Krishnamurthy, A., Agarwal, A., Langford, J., and Schapire, R. E. On oracle-efficient PAC reinforcement learning with rich observations. In *Advances in Neural Information Processing Systems*, 2018.

Ernst, D., Geurts, P., and Wehenkel, L. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 2005.

Givan, R., Dean, T., and Greig, M. Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*, 2003.

Hallak, A., Di-Castro, D., and Mannor, S. Model selection in Markovian processes. In *International Conference on Knowledge Discovery and Data Mining*, 2013.

Jaksch, T., Ortner, R., and Auer, P. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 2010.

Jiang, N., Kulesza, A., and Singh, S. Abstraction selection in model-based reinforcement learning. In *International Conference on Machine Learning*, 2015.

Jiang, N., Krishnamurthy, A., Agarwal, A., Langford, J., and Schapire, R. E. Contextual decision processes with low Bellman rank are PAC-learnable. In *International Conference on Machine Learning*, 2017.

Jin, C., Allen-Zhu, Z., Bubeck, S., and Jordan, M. I. Is Q-learning provably efficient? In *Advances in Neural Information Processing Systems*, 2018.

Johnson, M., Hofmann, K., Hutton, T., and Bignell, D. The Malmo Platform for artificial intelligence experimentation. In *International Joint Conference on Artificial Intelligence*, 2016.

Kearns, M. and Singh, S. Near-optimal reinforcement learning in polynomial time. *Machine learning*, 2002.

Krishnamurthy, A., Agarwal, A., and Langford, J. PAC reinforcement learning with rich observations. In *Advances in Neural Information Processing Systems*, 2016.

Lattimore, T. and Hutter, M. PAC bounds for discounted MDPs. In *International Conference on Algorithmic Learning Theory*, 2012.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 2015.

Oh, J., Singh, S., and Lee, H. Value prediction network. In *Advances in Neural Information Processing Systems*, 2017.

Ortner, R., Maillard, O.-A., and Ryabko, D. Selecting near-optimal approximate state representations in reinforcement learning. In *International Conference on Algorithmic Learning Theory*, 2014.

Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. Deep exploration via bootstrapped DQN. In *Advances in Neural Information Processing Systems*, 2016.

Ostrovski, G., Bellemare, M. G., Oord, A. v. d., and Munos, R. Count-based exploration with neural density models. In *International Conference on Machine Learning*, 2017.

Papadimitriou, C. H. and Tsitsiklis, J. N. The complexity of markov decision processes. *Mathematics of operations research*, 12(3):441–450, 1987.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning*, 2017.

Silver, D., van Hasselt, H., Hessel, M., Schaul, T., Guez, A., Harley, T., Dulac-Arnold, G., Reichert, D., Rabinowitz, N., Barreto, A., and Degris, T. The predictron: End-to-end learning and planning. In *International Conference on Machine Learning*, 2017.

Weissman, T., Ordentlich, E., Seroussi, G., Verdu, S., and Weinberger, M. J. Inequalities for the L1 deviation of the empirical distribution. *Hewlett-Packard Labs, Tech. Rep*, 2003.

Whitt, W. Approximations of dynamic programs, I. *Mathematics of Operations Research*, 1978.