Can Agents Learn by Analogy? An Inferable Model for PAC Reinforcement Learning

Yanchao Sun
University of Maryland
College Park, MD
ycs@cs.umd.edu

Furong Huang
University of Maryland
College Park, MD
furongh@cs.umd.edu

ABSTRACT

Model-based reinforcement learning algorithms make decisions by building and utilizing a model of the environment. However, none of the existing algorithms attempts to infer the dynamics of any state-action pair from known state-action pairs before meeting it for sufficient times. We propose a new model-based method called Greedy Inference Model (GIM) that infers the unknown dynamics from known dynamics based on the internal spectral properties of the environment. In other words, GIM can "learn by analogy". We further introduce a new exploration strategy which ensures that the agent rapidly and evenly visits unknown state-action pairs. GIM is much more computationally efficient than state-of-the-art model-based algorithms, as the number of dynamic programming operations is independent of the environment size. Lower sample complexity could also be achieved under mild conditions compared against methods without inferring. Experimental results demonstrate the effectiveness and efficiency of GIM in a variety of real-world tasks.

KEYWORDS

Model-based reinforcement learning, Spectral method, Sample complexity, Computational complexity

ACM Reference Format:
Yanchao Sun and Furong Huang. 2020. Can Agents Learn by Analogy? An Inferable Model for PAC Reinforcement Learning. In *Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), Auckland, New Zealand, May 9–13, 2020,* IFAAMAS, 17 pages.

1 INTRODUCTION

In Reinforcement Learning (RL)[37], an agent interacts with the environment by taking actions and receiving rewards or payoffs to figure out a policy that maximizes the total rewards. Recently, RL has been successfully applied in manyfi elds such as robotics [27], games [30], recommendation systems [44], etc. However, the high sample complexity and cost of computational resources prevent RL algorithms from being successfully deployed in many real-world tasks.

We call the RL algorithms which explicitly learn a model from experiences *model-based*, and algorithms that directly learn from interactions without any model *model-free*. Although these two types of algorithms are both effective in learning, they usually differ in terms of *sample complexity*, *computational complexity* and *space complexity*, which respectively measure the amount of interactions, computations, and memory an algorithm needs in RL tasks. Model-based algorithms are more sample efficient, but cost more computations and space. In contrast, model-free algorithms save computations and space, but usually need more samples/experience to learn, and easily get trapped in local optima.

In this paper, we focus on model-based algorithms due to the following two reasons. First, model-based algorithms make more efficient use of samples than model-free ones. Most existing PAC-MDP algorithms[1] are model-based. Second, the learned model is an abstraction of the environment, and can be easily transferred to other similar tasks [8]. For instance, if we change the reward of a state, only the reward value of the state should be changed in the learned model in model-based methods. However, for model-free methods, many state and action values will be affected.

Our goal is tofi nd a method that can reduce both the sample and computational complexity of model-based methods. We focus on the following challenges:

- *High stochasticity.* The transitions among states are usually stochastic. The highly stochastic transitions require a large number of trials and errors to reach the right decisions. Can we avoid visiting the highly stochastic transitions and still achieve a good policy?
- *Dilemma between sample and computational complexity.* Utilizing samples in an efficient manner requires more operations, while pursuing high speed may sacrifice accuracy and lead to more errors. Can we achieve both sample and computational efficiency?
- *Interplay of exploration and exploitation.* The trade-off between exploration and exploitation is a crucial problem in RL. Should we always iteratively alternate between exploration and exploitation as implemented by most existing algorithms?

Recent efforts [20, 33, 38] improve the sample and computational efficiency of model-based RL algorithms from various perspectives. However, the internal structure of the underlying Markov Decision Process (MDP) is ignored. The internal structure of MDPs refers to the spectral properties of the transition and reward distributions. More explicitly, we observed that many MDPs have locally or globally interrelated dynamics, resulting from the existence of similar states, actions or transitions. For example, consider a simple 2×3 grid world in Table 1(left) where the agent can go up, down, left and right. Thefl oor is slippery so when it wants to go up, it may slip to either left or right with probability 0.2. (Slipping also happens for

Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, G. Sukthankar (eds.), May 9–13, 2020, Auckland, New Zealand.

[1] An algorithm is PAC-MDP (Probably Approximately Correct in Markov Decision Processes) if its sample complexity is polynomial in the environment size and approximation parameters with high probability.

other actions). If there is a wall in the objective direction, it stays in the current state. Table 1(right) is the transition table from other states to state 2; the entries are the transition probabilities from state-action pairs to state 2. The rows of state 4 and 6 are omitted because state 2 is not immediately reachable from state 4 or 6. We find that the rows of state 2 and 5 are exactly the same, and the rank of this matrix is 3, smaller than the number of states or actions. This phenomenon becomes more ubiquitous in larger environments.

1	2	3
4	5	6

	up	down	left	right
1	0.2	0.2	0	0.6
2	0.6	0	0.2	0.2
3	0.2	0.2	0.6	0
5	0.6	0	0.2	0.2

Table 1: A grid world example and the transition table to state 2.

Due to the existence of such similar structures, we do not have to learn every state and action thoroughly, and the exploration can be much more efficient. We propose to explore a subset of "essential" transition dynamics, then infer the remaining dynamics using spectral methods, to achieve sample and computational efficiency.

In this paper, we propose a novel model-based RL algorithm called Greedy Inference Model (GIM), which utilizes the structural and spectral characteristics of MDPs and thus expedites the learning process. GIM introduces a novel exploration strategy to discover the unknowns efficiently, and a spectral method to estimate the entire model with the knowledge of a fraction of the model. The core idea of GIM can be applied to any model-based methods. We prove that GIM is PAC-MDP, and it has significantly lower computational complexity and potentially lower sample complexity than state-of-the-art model-based algorithms. Systematic empirical studies demonstrate that GIM outperforms both model-based and model-free state-of-the-art approaches on a wide variety of tasks.

Our contributions are summarized as follows:

- To the best of our knowledge, we are **thefi rst to estimate the model by utilizing the internal structure of the MDPs with guaranteed accuracy**. We avoid directly estimating the highly stochastic transitions, which is sample-consuming.
- We show that GIM can significantly **reduce the computational cost**, as the number of dynamic programming operations is independent of the environment size. We also prove GIM could **improve the sample efficiency** of model-based algorithms.
- We propose a **new mechanism to address the exploration and exploitation dilemma**. By using a new exploration strategy (β-curious walking), GIM takes fewer exploration steps in total than existing methods.

2 RELATED WORK

2.1 RL Algorithms

Model-based algorithms. Model-based algorithms like E^3 [24], RMax [6] and MBIE [35] construct a model from interactions, storing the transition probabilities and rewards of every state and action pair, and then make predictions with the model. Followup works improve the efficiency of aforementioned model-based algorithms. RTDP-RMAX and RTDP-MBIE [33] reduce the number of updates and achieve lower computational complexity, with minor performance loss on the accumulated rewards achieved. MORMAX [38] modifies RMax algorithm and reduces the sample complexity [12, 23] by maintaining an imperfect model, but the model estimation is not accurate which prevents accurate rewards predictions for some state-action pairs. [20] proposes a method with no dependence on the size of the state-action space, but it assumes that an approximate imperfect model is given.

Model-free algorithms. Model-free algorithms [18, 40, 41] decide what actions to take based on the trajectory/history. Delayed Q-learning [34] is a special model-free algorithm as it is PAC-MDP, whose sample complexity depends linearly on the state and the action number. However Delayed Q-learning has higher dependence on the discount factor γ and the error tolerance ϵ than RMax.

Deep RL algorithms. Recently, researchers have made significant progress by combining deep learning with both model-based or model-free RL [30, 39] and achieving impressive empirical performance. However theoretical understanding of deep learning , and thus deep RL, remains unsettled. Deep RL, usually applied for large-scale decision-making problems, requires large number of training examples, which are not practical for tasks with limited training examples.

PAC RL. A key goal of RL algorithms is to maximize the reward with as few samples as possible. The sample-efficiency of RL algorithms can be measured by the PAC performance metric (sample complexity)fi rst formally defined in [23]. [12] derives a tighter PAC upper bound for episodicfi xed-horizon RL tasks. Recently, more strict metrics like Uniform-PAC and IPOC [13, 14] are proposed to measure the performance of RL algorithms. And by computing certificates for optimistic RL algorithms [14], minimax-optimal PAC bounds up to lower-order terms are achieved under certain conditions.

2.2 Spectral Methods

Matrix completion. The spectral method we will use in this paper is mainly the well-studied matrix completion. It is proved that we can recover a matrix with only a fraction of its (noisy) entries [9, 26].

Spectral methods and RL. Spectral methods have been applied in RL in the learning of POMDP (Partially Observable Markov Decision Process) [4] and ROMDP (Rich-Observation Markov Decision Process) [3], where a multi-view model [2] is used. Researchers discover that knowledge can be transferred between tasks, domains or agents [7, 16, 28] using spectral methods. Moreover, some recent works propose new learning algorithms by constructing certain low-rank models [5, 21, 31], where spectral methods are involved.

Low-rank transition model. There is a line of works learning the low-rank structure of the transition models [15, 22, 29, 42, 43], although we focus on different low-rank objects and use different models as well as assumptions.

3 NOTATIONS AND PROBLEM SETUP

3.1 Notations for RL

In this paper, we focus on episodic, discrete-time, andfi xed horizon MDPs withfi nite state and action spaces. A Markov decision process (MDP) is defined as a tuple $< \mathcal{S}, \mathcal{A}, p(\cdot|\cdot,\cdot), r(\cdot,\cdot), \mu >$, where $\mathcal{S}$ is

the state space (with cardinality S); $\mathcal{A}$ is the action space (with cardinality A); $p(\cdot|\cdot,\cdot)$ is the transition probability function with $p(s_k|s_i,a_j)$ representing the probability of transiting to state s_k from state s_i by taking action a_j; $r(\cdot,\cdot)$ is the reward function with $r(s_i,a_j)$ recording the reward one can get by taking action a_j in state s_i; μ is the initial state distribution. $p(\cdot|\cdot,\cdot)$ and $r(\cdot,\cdot)$ together are called the dynamics of the MDP. We use H to denote the horizon (number of steps one can take in an episode) of an MDP.

Definition 3.1 (Dynamic Matrices). Given an MDP M denoted by tuple $< \mathcal{S}, \mathcal{A}, p(\cdot|\cdot,\cdot), r(\cdot,\cdot), \mu >$, we define $S+1$ dynamic matrices $\{\boldsymbol{M}^s\}_{s\in\mathcal{S}}$ and $\boldsymbol{M}^r$. $\{\boldsymbol{M}^s\}_{s\in\mathcal{S}}$ are called transition dynamic matrices, where $\boldsymbol{M}^s_{ij} = p(s|s_i,a_j)$ for all $s \in \mathcal{S}$. $\boldsymbol{M}^r$ is called reward dynamic matrix in which $\boldsymbol{M}^r_{ij} = r(s_i,a_j)$.

The empirical estimations of the dynamic matrices are:

$$\hat{\boldsymbol{M}}^s_{ij} = \frac{n(s|s_i,a_j)}{n(s_i,a_j)} \ \forall s \text{ and } \hat{\boldsymbol{M}}^r_{ij} = \frac{R(s_i,a_j)}{n(s_i,a_j)}, \tag{1}$$

where $n(s_i,a_j)$ is the total number of visits to state-action pair (s_i,a_j), $n(s|s_i,a_j)$ the total number of transitions from s_i to s by taking action a_j and $R(s_i,a_j)$ the total rewards [2] for (s_i,a_j). The empirical dynamic matrix $\hat{\boldsymbol{M}}$ is an approximation of the corresponding dynamic matrix $\boldsymbol{M}$, so we have $\hat{\boldsymbol{M}} = \boldsymbol{M} + \boldsymbol{Z}$ where $\boldsymbol{Z}$ is a noise matrix. The more observations we have, the more accurate the approximation is.

Our main goal is to recover every $\boldsymbol{M}$ based on $\hat{\boldsymbol{M}}$. More explicitly, given the empirical dynamic matrix $\hat{\boldsymbol{M}}$, the algorithm should return a matrix $\widetilde{\boldsymbol{M}}$ that is ϵ-close to the original $\boldsymbol{M}$, i.e., $\|\widetilde{\boldsymbol{M}} - \boldsymbol{M}\| \le \epsilon$.

The value function of a policy π for a given MDP M with horizon H is the expected average reward $V^\pi_M = \mathbb{E}_{s_0\sim\mu}[\frac{1}{H}\sum_{h=0}^{H-1} r(s_h,\pi(s_h))]$. The optimal policy π^* is the policy that achieves the largest possible value V^*_M. In an RL task, an agent searches for the optimal policy by interacting with the MDP. The general goal of RL algorithms is to learn the optimal policy for any given MDP with as few interactions as possible. A widely-used framework to evaluate the performance of RL algorithms is *sample complexity of exploration* [23], or *sample complexity* for short.

Definition 3.2 (Sample complexity of exploration). For any $\epsilon > 0$ and $0 < \delta < 1$, and at any episode t, if the policy π_t generated by an RL algorithm L satisfies $V^* - V^{\pi_t} \le \epsilon$, we say L is near-optimal at episode t. If with probability at least $1-\delta$, the total number of episodes that L is not near-optimal is upper bounded by a function $\zeta(\epsilon,\delta)$, then ζ is called the sample complexity of L.

Intuitively, sample complexity illustrates the number of steps in which the agent does not act near-optimally.

3.2 Notations for Spectral Methods

Incoherence [26] of a matrix is an important property that demonstrates the "sparsity" of the singular vectors of the matrix: all coordinates of each singular vector are of comparable magnitude (a.k.a., a dense singular vector) vs just a few coordinates having significantly larger magnitudes (a.k.a., a sparse singular vector).

[2] $R(s_i,a_j)$ is the empirical total rewards gained by visiting (s_i,a_j) in the history, and is different from $r(s_i,a_j)$.

Definition 3.3 ((μ_0,μ_1)-incoherence). A matrix $\boldsymbol{M} \in \mathbb{R}^{m\times n}$ with rank r has SVD $\boldsymbol{M} = \boldsymbol{U}\Sigma\boldsymbol{V}^\mathbf{T}$, where $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthonormal. We say $\boldsymbol{M}$ is (μ_0,μ_1)-incoherent if (1) for all $i \in [m]$, $j \in [n]$ we have $\sum_{k=1}^r U^2_{ik} \le \mu_0 r$, and $\sum_{k=1}^r V^2_{jk} \le \mu_0 r$; (2) There exist μ_1 such that $|\sum_{k=1}^r U_{ik}(\Sigma_k/\Sigma_1)V_{jk}| \le \mu_1\sqrt{r}$, where Σ_k is the k-th singular value of $\boldsymbol{M}$.

The smaller μ_0 and μ_1 are, the more spread-out the singular vectors are. As a result, matrix completion methods require a smaller number of known entries to confidently recover the entire matrix. See Appendix[3] A for details about matrix completion and incoherence.

4 MOTIVATIONS

Before the formal introduction of our proposed learning algorithm, we consider two questions:

Is it necessary to learn every state-action pair from scratch? The key to RL is to evaluate the value of every state and action, with or without a model. The agent makes observations of each state-action pair, accumulates experience, and estimates the model or the values. It knows nothing about a state-action pair before meeting it. However, is it necessary to learn every new state-action pair from scratch? As humans, we can learn by analogy. For example, if one has jumped out of a window on the secondfl oor and got injured, he will learn never to jump from another window on the thirdfl oor, because he accumulates knowledge from his previous experience andfi nds the internal connections between these two situations. But existing RL agents, which are not able to analyze new states, tend to make the same mistakes in similar situations.

Therefore, in this work, we extract and use the internal connections of the environment via spectral methods to reduce the unnecessary trials and errors for the agent.

Should we always interleave exploration and exploitation? The exploration-exploitation dilemma has been intensively studied for decades and remains unsolved. In RL, exploration is to try the unknowns, while exploitation maximizes rewards based on the current knowledge. Most RL algorithms interleave (or alternate between) exploration and exploitation. An example is the widely-used ε-greedy exploration method, which chooses actions greedily with respect to the action values with probability $1-\varepsilon$ (exploit), and randomly chooses actions with probability ε (explore). Moreover, many model-based algorithms, such as E^3 and RMax, choose actions with the maximum value in known states, and execute the action that has been tried the fewest times in unknown states. However, is this interleaving the only manner to get the optimal results? If the agent "greedily" chooses the most rewarding action when it knows little about the whole environment, it usually misses the largest possible reward in the long run. Can the agent ignore the short-term benefits in the beginning, and keep exploring before it gains enough knowledge?

In this work, we implement a two-phase algorithm in which exploitation follows after exploration, instead of interleaving the two. And we prove that our new method requires fewer samples and computations.

[3] The Appendix of this paper is in https://arxiv.org/abs/1912.10329

5 GREEDY INFERENCE MODEL

In this section, we present a novel model-based RL algorithm called Greedy Inference Model (GIM) that considers the structural properties of MDPs and separates exploration from exploitation. More explicitly, two main ideas of GIM are (1) using matrix completion to recover/complete the dynamic matrices, and (2) greedily exploring the unknowns.

5.1 Complete Unknowns with Knowns

As in many model-based algorithms [6, 24], we distinguish all state-action pairs as "m-known" or "m-unknown" (we will say known and unknown for short) pairs: a state-action pair is known if we have visited it for over m times, so that the estimations for its transition probabilities and average reward are guaranteed to be nearly accurate with high probability. We use $\mathcal{K}$ to denote the set of all known state-action pairs, and $\bar{\mathcal{K}}$ for unknowns.

Definition 5.1 (Known-ness Mask). For an MDP M with S states and A actions, the known-ness mask $P^{\mathcal{K}} \in \mathbb{R}^{S\times A}$ is a binary matrix defined as

$$P^{\mathcal{K}}_{i,j} = \begin{cases} 1 & \text{if } (s_i, a_j) \text{ is } m\text{-known} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Remark. The summation of all entries of the known-ness mask is the total number of known state-action pairs in the MDP. Row sums and column sums are the numbers of known state-action pairs related to every state or action respectively.

As discussed in Motivations, unlike previous model-based methods such as RMax, we avoid the necessity of observing and gaining knowledge on every single state-action pair as the MDPs usually have some internal structure/pattern. We use matrix completion, a widely used spectral method, to estimate the missing values in partially observed matrices. We now introduce how to complete the "unknowns" with "knowns" in an MDP.

5.1.1 Estimate Unknowns via Matrix Completion. Matrix completion is the problem of recovering unknown entries in the matrix from a small fraction of its known (noisy) entries, which are randomly sampled from the matrix. When the matrix satisfies some assumptions that we will discuss later, the recovery is guaranteed to be accurate, even under noisy known entries. Based on matrix completion theory, GIM needs only a fraction of state-action pairs to be known to recover the unknown state-action pairs. Now we formally define the matrix completion problem as the optimization problem: for every dynamic matrix $\boldsymbol{M}$

$$\begin{aligned} &\text{minimize}_{\widetilde{\boldsymbol{M}}} && \|\boldsymbol{P}^{\mathcal{K}} \odot (\widetilde{\boldsymbol{M}} - \hat{\boldsymbol{M}})\| \\ &\text{subject to} && rank(\widetilde{\boldsymbol{M}}) \le r \end{aligned} \tag{3}$$

where $\boldsymbol{P}^{\mathcal{K}}$ is the known-ness mask defined in Definition 5.1, $\odot$ denotes element-wise product, and r is the rank of $\boldsymbol{M}$ or the upper bound of the rank.

5.1.2 Requirements for Accurate Completion. Matrix completion makes it possible to know all the dynamics from some known state-action pairs, but the accuracy of completed dynamics is determined by the structure of the matrix, as well as the number and the locations of known entries. In general, matrix completion with noisy observations requires (1) the number of known entries be greater than some threshold, and (2) the known entries be spread out randomly. We propose the following exploration strategy that conforms to the two requirements above.

5.2 Greedily Explore the Environment

To satisfy the two requirements on the known state-action pairs and guarantee the success of matrix completion, we propose a new exploration strategy called *β-curious walking*.

Let ρ denote the fraction of known state-action pairs over all state-action pairs. Therefore $\rho SA = |\mathcal{K}|$. We introduce the concept of ρ-known state below.

Definition 5.2 (ρ-known state). A state s is ρ-known if there exist ρA distinct actions such that the corresponding state-action pair (s, a) is known.

Intuitively, the idea of our proposed β-curious walking is: if the current state s is not ρ-known, choose an action a that the agent has taken the most but (s, a) is still unknown; if the current state is ρ-known, select the action which most likely leads to a non-ρ-known state. The agent also chooses actions randomly with a small probability β to avoid being trapped in local optima.

Algorithm 1 shows the procedure of *β-curious walking*, where Random() generates a random number from a uniform distribution in $[0, 1]$; $n(s, a)$ is the total number of visits to state-action pair (s, a); $n(s'|s, a)$ is the total number of transitions from s to s' by taking action a; the indicator function $\mathbb{I}(s'$ is non-ρ-known) is 1 if s' is not ρ-known, and 0 otherwise.

Algorithm 1: β-CuriousWalking

Input: The current state s, a hyper-parameter β
Output: The chosen action a^*

```
if Random() < β then
    return a* ← a random action
if s is non-ρ-known then
    Ã ← A
    foreach a ∈ Ã do
        Ã ← Ã/{a} if (s,a) is known
    a* ← arg max_a n(s, a), a ∈ Ã
else
    foreach a ∈ A do
        t(a) = Σ_{s'} (n(s'|s, a) / n(s, a)) 𝕀(s' is non-ρ-known)
    a* ← arg max_a t(a), a ∈ A
return a*
```

Compared with the balanced walking method used in both E^3 and RMax, β-curious walking:

- Encourages the agent to choose the actions it has the most experience with, until the action is known for the current state. So the agent rapidly knows ρSA state-action pairs.
- Spreads the knowledge evenly, i.e., the agent attempts to know every state with ρA actions instead of attempting to know all A actions for some states but nothing for other states.

5.3 GIM Algorithm

The proposed GIM algorithm is described in Algorithm 2. When there are less than ρSA m-known state-action pairs, the agent keeps exploring with β-curious walking (see lines 7-14). As long as ρSA pairs are m-known, the algorithm performs matrix completion for all dynamic matrices (see lines 15-20). MatComp could be any off-the-shelf matrix completion algorithm that solves the problem defined in Equation (3). Note that matrix completion algorithms implicitly estimate the rank r of the input matrix, so there is no need to specify the rank as an input to Algorithm 2.

Known threshold. m is the least number of visits to one state-action pair to make the estimation and the completion accurate. The choice of m is specified in Theorem 6.3.

Fraction of known state-action pairs. ρ controls the fraction of known state-action pairs, based on which the matrix completion can get convincing results for unknowns. The value of ρ is determined by the structure of the underlying MDP. The more the states and actions in the MDP are interrelated, the smaller ρ can be. If the underlying MDP has completely unrelated dynamics, then ρ is set to be 1, and matrix completion does nothing but returning the empirical transition model itself.

We will further discuss parameters m and ρ in theory and practice in the next two sections. More importantly, we will show that the advantage of our proposed algorithm over previous model-based algorithms is larger for smaller ρ. Even under the worst scenario of $\rho = 1$, our β-curious walking improves learning efficiency under certain conditions.

Algorithm 2: Greedy Inference Algorithm

Input: $T, H, \epsilon, m, \rho, \beta$
Output: Near-optimal policy $\tilde{\pi}$ such that $V^{\tilde{\pi}} \geq V^* - \epsilon$

1 Initialize dynamic matrices $\{\hat{\boldsymbol{M}}^s\}_{s\in\mathcal{S}}, \hat{\boldsymbol{M}}^r$
2 Initialize $n(s,a), n(s'|s,a), R(s,a)$ for all $s, s' \in \mathcal{S}, a \in \mathcal{A}$
3 Initialize $\boldsymbol{P}^{\mathcal{K}}$ as all zeros
4 **for** *episode* $t \leftarrow 1$ **to** T **do**
5 $s_1 \leftarrow$ initial state
6 **for** *step* $h \leftarrow 1$ **to** H **do**
7 **if** $sum(\boldsymbol{P}^{\mathcal{K}}) < \rho SA$ **then**
8 $a_h \leftarrow \beta$-CuriousWalking(s_h, β)
9 Execute a_h, get s_{h+1} and r_{h+1}
10 $n(s_h, a_h) \leftarrow n(s_h, a_h) + 1$
11 $n(s_{h+1}|s_h, a_h) \leftarrow n(s_{h+1}|s_h, a_h) + 1$
12 $R(s_h, a_h) \leftarrow R(s_h, a_h) + r_{h+1}$
13 Update $\hat{\boldsymbol{M}}^{s_{h+1}}$ and $\hat{\boldsymbol{M}}^r$ by Equation (1)
14 **if** $n(s_h, a_h) \geq m$ **then** Update $\boldsymbol{P}^{\mathcal{K}}$
15 **if** $sum(\boldsymbol{P}^{\mathcal{K}}) \geq \rho SA$ **then**
16 **for** $s \in \mathcal{S}$ **do**
17 $\widetilde{\boldsymbol{M}}^s \leftarrow$ `MatComp`$(\hat{\boldsymbol{M}}^s, \boldsymbol{P}^{\mathcal{K}})$
18 $\widetilde{\boldsymbol{M}}^r \leftarrow$ `MatComp`$(\hat{\boldsymbol{M}}^r, \boldsymbol{P}^{\mathcal{K}})$
19 Set all state-action pairs as known
20 Compute the optimal policy $\tilde{\pi}$
21 **else**
22 Choose a_h with optimal policy $\tilde{\pi}$

5.4 GIM As a Framework

Although Algorithm 2 estimates the dynamics by directly averaging collected samples, which is similar to the classic RMax algorithm, GIM can also be regarded as a framework and can be combined with other model-based methods. The simple RMax-style structure in Algorithm 2 is an illustration of how GIM could be combined with a model-based method; the analysis we will provide in Section 6 exhibits how GIM could improve a model-based method.

The key ideas of GIM are to infer the unknown dynamics as well as to know the environment greedily and evenly, which improve the model-based method combined. For example, in algorithms driven by confidence interval estimation, such as MBIE [35], UCRL2 [19] and etc, we can also use matrix completion to recover the "uncertain dynamics" using the "dynamics with high confidence". This extra operation will not affect what has been learned, but rather make a guaranteed estimation of the unlearned parts. Therefore, the learning process is boosted by utilizing the internal structures of the environment, and as a result, samples are saved.

Overall, it is not our goal to propose a specific algorithm with the best complexity. Instead, we attempt to improve any model-based algorithm by inferring the dynamics.

6 THEORETICAL ANALYSIS ON COMPLEXITIES

In this section, we analyze the computational complexity, sample complexity and space complexity of GIM. By comparing with existing model-based methods, we show that GIM achieves a much better computational complexity and improves the sample complexity under mild conditions.

6.1 Computational Complexity

Theorem 6.1 states the computational complexity of GIM.

Theorem 6.1 (ComputationalC omplexity ofGIM). *Given an MDP M with S states and A actions, if GIM is executed for $\mathcal{N}$ steps, then the total computational complexity of GIM is*

$$\widetilde{O}(\varphi + S\max\{S, A\} + \mathcal{N}), \tag{4}$$

where φ be the number of computations for one dynamic programming operation, i.e., updating the policy by solving Bellman equations.

Remark. φ depends on the environment size. More specifically, if the maximum number of iterations for dynamic programming is set as $\mathcal{U}$, then $\varphi = O(SA\mathcal{U})$.

Proof. During the execution of GIM, both dynamic programming and matrix completion are implemented only once, which lead to $O(\varphi)$ and $\widetilde{O}(S\max\{S, A\})$ [17] computations. For every time step, GIM updates $n(s,a)$, $n(s'|s,a)$ and $R(s,a)$, which can be done in constant time. Although β-curious walking in Algorithm 1 performs a loop over all the actions, in practice we are able tofi nd the best action within constant time or logarithmic time through maintaining the known-ness table. These constant-time per step computations together lead to the $\widetilde{O}(\mathcal{N})$ term.

Comparison with RMax and RTDP-RMAX[33]. Wefi rst compare the cost of dynamic programming, the major computational burden for most RL algorithms. Since within each dynamic programming,

the amount of computation required is the same for GIM and other model-based methods, we use the number of dynamic programming operations as the metric for computation complexity comparison.

Lemma6.2. *The number of dynamic programming operations required by GIM is $O(1)$, whereas the number of dynamic programming operations required by RMax and RTDP-RMAX are $O(S)$ and $O(\frac{SA\varepsilon}{V_{\max}})$.*

RMax's computational complexity is $O(S\varphi)$, as RMax computes the action values every time a new state is known. RTDP-RMAX is proposed to reduce the computational complexity of RMax. It initializes all action values to be $V_{\max}$, the maximum possible value for an episode, and only updates the value of one state-action pair when the value decreases more than some threshold ε. So it requires at most $O(\frac{SA\varepsilon}{V_{\max}}\varphi)$ computations for the dynamic programming, where ε is the error tolerance of action-value estimation. Thus in terms of dynamic programming computation complexity, GIM is much faster than RMax and RTDP-RMAX.

Besides the dynamic programming, as shown in Theorem 6.1, GIM requires some constant-time operations per step, which is inevitable for all algorithms, and an extra matrix completion computation, which is a one-time cost and is negligible in a long learning process. Experiments in Section 7.1.4 verifies this fact.

6.2 Sample Complexity

As stated in Section 5.4, what we propose is a new exploration and estimation approach, that could be combined with model-based PAC algorithms [12, 36, 38] to get lower sample complexity. In this section, we analyze the sample complexity of Algorithm 2 by adapting the analysis of RMax [23]. RMax is chosen due to its simplicity and versatility.

We now introduce a few notations that are essential in our analysis. (1) Denote the upper bounds of the condition number and the rank of every dynamic matrix by κ and r. (2) All dynamic matrices are at least (μ_0, μ_1)-incoherent. (3) Let $M_{in} = \min\{S, A\}$, and $M_{ax} = \max\{S, A\}$.

We also make the following two mild assumptions.

Assumption1. *There is a known diameter D, such that any state s' is reachable from any state s in at most D steps on average. Assume that the diameter D is smaller than the horizon H.*

The assumption about diameter is commonly used in RL [19], and it ensures the reachability of all states from any state on average. It is mild to assume $D < H$ as the horizon is often set large.

Assumption2. *The distribution of the estimation noise, $p(\cdot|\cdot,\cdot) - \hat{p}(\cdot|\cdot,\cdot)$ and $r(\cdot,\cdot) - \hat{r}(\cdot,\cdot)$, is sub-Gaussian with 0 mean. And the estimation noises for different state-action pairs are independent.*

This modeling of difference between the ground-truth probability and empirical estimation using sub-Gaussian variables is widely used.

The sample complexity of GIM is in Theorem 6.3.

Theorem 6.3 (SampleC omplexity ofGIM). *Given an MDP M withfi xed horizon H and diameter D, suppose the upper bounds of the condition number, rank and incoherence parameters of dynamic matrices are κ, r, μ_0 and μ_1, for any $0 < \epsilon < 1$, $0 \le \delta < 1$, with completion fraction*

$$\rho \ge \Omega(\frac{1}{\sqrt{SA}}\kappa^2 \max\{\mu_0 r\sqrt{\frac{M_{ax}}{M_{in}}}\log M_{in}, \mu_0^2 r^2 \frac{M_{ax}}{M_{in}}\kappa^4, \mu_1^2 r^2 \frac{M_{ax}}{M_{in}}\kappa^4\}), \tag{5}$$

and the known threshold

$$m \ge O(\frac{\kappa^4 r S H^2 M_{ax}}{\rho A \epsilon^2}), \tag{6}$$

algorithm 2 produces a policy $\hat{\pi}$, which satisfies $V_M^{\hat{\pi}} \ge V_M^* - \epsilon$ for all but $O(\frac{\kappa^4 r S^2 M_{ax} HD}{(1-\beta)\epsilon^2}\log\frac{1}{\delta})$ episodes, with probability at least $1 - \delta - 1/M_{in}^3$.

Remark. For low-rank dynamic matrices, κ tends to be small. In our experiments, κ is typically less than 2. If we regard $\frac{\kappa^4 D}{(1-\beta)}$ as a constant, the sample complexity of GIM becomes $O(\frac{rS^2 M_{ax} H}{\epsilon^2}\log\frac{1}{\delta})$.

Proof Sketch. Wefi rst set m to be the least number of visits to a state-action pair to make it known (see Condition 1 in Appendix B.1). Then, we prove with at most $O(\frac{\rho m SAD}{(1-\beta)H}\log\frac{1}{\delta})$ episodes, we know ρSA pairs as Lemma B.1. Finally, we prove the value of m should be $O(\frac{\kappa^4 r S H^2 M_{ax}}{\rho A \epsilon^2})$ by Lemma B.2, Lemma B.4 and Lemma B.3. The full proof and the lemmas are in Appendix B.

Comparison with RMax. The sample complexity of RMax, in our settings, is $O(\frac{S^2AH^2}{\epsilon^3}\log\frac{1}{\delta}\log\frac{SA}{\delta})$[23]. We compare the sample complexity of GIM and RMax in the following scenarios.
(1) When $A \ge S$. GIM has lower sample complexity than RMax if $r < O(\frac{H}{\epsilon}\log\frac{SA}{\delta})$.
(2) When $S > A$. GIM has lower sample complexity than RMax if $r < O(\frac{AH}{S\epsilon}\log\frac{SA}{\delta})$.
(3) Worst Scenario ($\rho = 1$). We deactivate the matrix completion steps by simply setting $\rho = 1$ and $m = O(\frac{SH^2}{\epsilon^2}\log\frac{SA}{\delta})$, when the underlying MDP does not have any inner-related structure. This makes GIM follow β-curious walking until all the state-actions are known. In this case, the sample complexity becomes $O(\frac{S^2AHD}{(1-\beta)\epsilon^2}\log\frac{1}{\delta}\log\frac{SA}{\delta})$. Because $H \gg D$ and $1-\beta$ is close to 1, GIM generates less non-ϵ-optimal episodes than RMax does. So β-curious walking strategy itself saves samples.

Note that $r \le \min\{S, A\}$, so the conditions in (1) and (2) are satisfied for most tasks.

Achieve Lower Sample Complexity. One may note that the sample complexity bound in Theorem 6.3 is not optimal in terms of the dependency on S and A. When $A > S$, GIM needs $\tilde{O}(S^2A)$ samples to learn a near-optimal policy. However, the best known bounds of model-based algorithms are of order $\tilde{O}(SA)$ [13, 14, 38]. The saved factor of S results from the direct analysis of the value function (and an imperfect model approximation). Since we claim that GIM can work as a framework, can GIM also achieve linear dependency? In our analysis and the original analysis of RMax, the known threshold is at least $m = O(S\ln S)$, but as indicated by [25], with specific updating strategies, $m = O(\ln S)$ samples might be enough to maximize the rewards. So it is possible for GIM to avoid an S factor by

incorporating re-estimating methods, although this is out of the scope of this paper.

6.3 Space Complexity

The memory GIM needs is mainly for the storage of dynamic matrices. Similar with other model-based RL algorithms, the space complexity of GIM is $\Theta(S^2A)$, as we have $S+2$ matrices with size $S \times A$. In contrast, the space complexity of model-free algorithm such as Delayed Q-learning could be as low as $O(SA)$. Although a large space complexity seems to be unavoidable for model-based methods, one can consider storing the sparse dynamics in a sparse format where only non-zero elements are stored if the dynamic matrices are sparse. Then the space complexity will be reduced to $\Theta(nnz)$, where nnz is the number of non-zero entries.

7 EXPERIMENTS

7.1 Performance on Multiple Tasks

7.1.1 Tasks. To exhibit the universal applicability of GIM, we conduct experiments on multiple tasks of varying levels of hardness: (1) *Synthetic.* We create various MDPs by randomly generating the dynamic matrices with varying numbers of states, actions and ranks. (2) *GridWorld.* A classic grid world task, with world size 4×4, slip probability 0.4 and step cost 0.2. (3) *CasinoLand.* A challenging task constructed by [32], which consists of six rooms and three levers. Pulling some levers may lead to a large reward with a small probability. (4) *RiverSwim.* Another challenging task constructed by [32], where a chain of 6 states represents a river, and an agent needs to "swim" from thefi rst state to the last one to get a large reward. See Appendix C.1 for detailed description of CasinoLand and RiverSwim.

7.1.2 Baselines. To verify the effectiveness and efficiency of GIM, it is compared against popular model-based and model-free methods: RMax, Q-learning, Delayed Q-learning, and Double Q-learning [18] methods. We select RMax among all model-based methods, because Algorithm 2 is designed on the basis of RMax. So the effectiveness of our proposed strategies can be justified by comparing with RMax. Note that it is also possible to apply similar strategies to other existing model-based algorithms, as claimed in Section 5.4.

Moreover, we implement an "optimal" agent which knows all the dynamics and deploys the optimal policy from the beginning, as well as a "random" agent which chooses action randomly. The "optimal" agent is the best any agent could achieve, while the "random" agent is the worst any agent could perform. We use the simple_rl framework provided by Abel [1] to conduct the experiments.

7.1.3 Reward Comparison. We demonstrate the average cumulative reward over 20 runs of GIM and baselines on various tasks in Figure 1. The plots are smoothed out by plotting every few hundred epochs for better illustration. Note that a line of reward increases linearly after itfi nds the best policy. Agents whose sample complexities are lower converge to the best policy earlier. For hyperparameters, we set the known threshold m for both RMax and GIM to be 40, and set the completion fraction threshold ρ for GIM to be 0.8. We will discuss the influence of different settings on RMax and GIM later. Among all these methods, GIM achieves the highest total

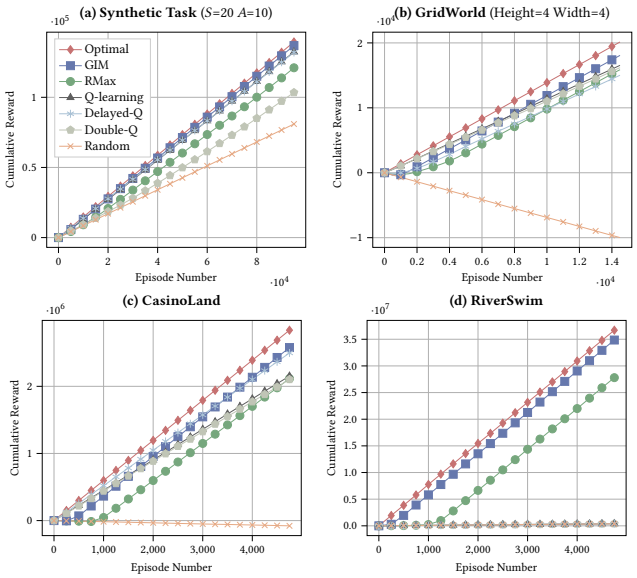

Figure 1: Comparison of the mean cumulative reward over 20 runs of GIM and baselines on various tasks.

reward. Although Q-learning, Delayed-Q and Double-Q converge to a good policy quickly, they are sometimes trapped in local-optima and cannot win in the long run. On the contrary, RMaxfi gures out a policy that is near-optimal, but it often takes more episodes to converge to that policy. GIM avoids these two drawbacks; it converges quickly and the returned policy is near-optimal.

	Model-based		*Model-free*		
	GIM	RMax	Q	Delayed-Q	Double-Q
Synthetic	539.19	694.15	364.83	530.47	619.41
Gridworld	11.75	13.94	12.68	11.2	13.7
Casinoland	7.21	8.18	3.61	3.00	3.93
RiverSwim	7.62	6.06	5.87	4.88	6.33

Table 2: Comparison of running times in seconds.

7.1.4 Running Time Comparison. Table 2 provides the average running times of each agent on various tasks. In general, our model-based GIM is faster than model-based RMax, and is even comparable to the model-free methods which are generally faster as there is no need to maintain a model. For RiverSwim, GIM is slightly slower than RMax as the one-time cost of computation of matrix completion slows down the GIM agent. This one-time cost is less significant for time-consuming tasks where running time is the bottleneck.

7.1.5 Scale Up to Larger Environments. Although many model-based algorithms perform well in small environments, it is usually not easy for them to work on large-scale problems. However, GIM avoids exhaustedly visiting and estimating the whole environment by inferring some dynamics. When the environment is highly internal-dependent, only the knowledge of a small fraction of the space is needed. Thus fewer samples and computations are consumed. To evaluate the scalability of GIM compared with baselines, we gradually enlarge the size of the synthetic task and show the cumulative rewards in Figure 2. GIM performs well compared with baselines, while RMax fails to converge to the optima within the

given number of episodes. We set the known thresholds for both GIM and RMax to be 100. As a result, the performance of learned policy by GIM is slightly worse than Q-learning, since $m = 100$ is not adequate for estimating the large environments. But one can anticipate higher rewards by GIM, once m is set higher.

The running times corresponding to Figure 2, are shown in Table 3, where one canfi nd GIM is much faster than RMax and Double Q-learning. As the environment size gets larger, GIM takes slightly longer to run, while RMax spends much more time due to the increased computation requirements.

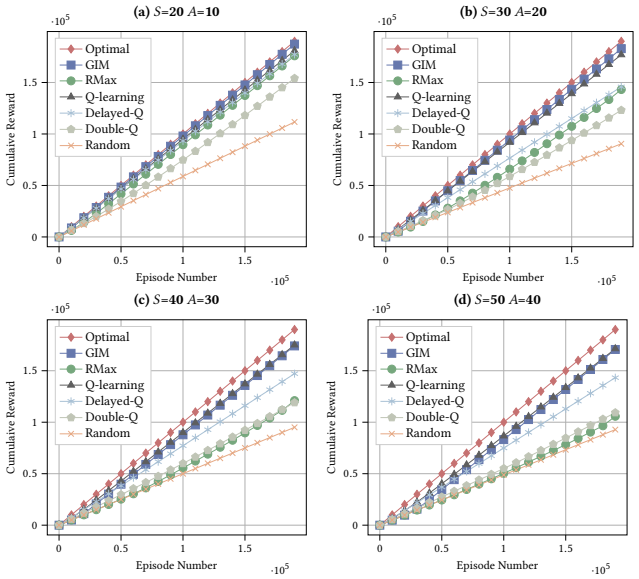

Figure 2: Comparison of the mean cumulative reward of GIM and baselines on synthetic tasks with various environment size.

	Model-based		*Model-free*		
	GIM	RMax	Q	Delayed-Q	Double-Q
S=20, A=10	2165	2410	1386	2244	2465
S=30, A=20	2296	4581	1635	2541	3107
S=40, A=30	2304	9778	2007	2477	3413
S=50, A=40	2423	19870	2287	2399	3424

Table 3: Comparison of running times (in seconds) on synthetic tasks with various environment size. The number of episodes are all set to be 2×10^5.

7.2 Experiments on Parameters

In practice, it is important to select an appropriate known threshold m for GIM and RMax. GIM also requires a completion fraction ρ. Although we can choose ρ based on experience, it is related to the properties of the dynamic matrices $(\kappa, r, \mu_0, \mu_1)$ as proved in Theorem 6.3. So, we design and conduct a series of systematic tests to study how $m, \kappa, r, \mu_0, \mu_1$ (forfi xed ρ) influence the learning effectiveness and efficiency of GIM, in comparison with RMax.

The following three measurements are evaluated. (1) **AvgReward**: average reward per episode; (2) **TotalEps**: number of episodes needed to know all states and actions; and (3) **PostAvgReward**: average reward after knowing all states and actions (after exploration), which reflects accurateness of the learned dynamics.

7.2.1 Values of the Known Threshold m. We run experiments on synthetic and GridWorld tasks with different known threshold m. Figure 3 shows the AvgReward, PostAvgReward and TotalEps of RMax and GIM with different values of m. For the same m, GIM gains more rewards (Figure 3a), and completes exploration faster than RMax (Figure 3b), with a slightly worse returned policy (Figure 3c). When m varies, RMax requires much more episodes to explore, while GIM is more robust to the changing m as shown in Figure 3b. We present the results on higher-rank synthetic tasks and the GridWorld task in Appendix C.2.

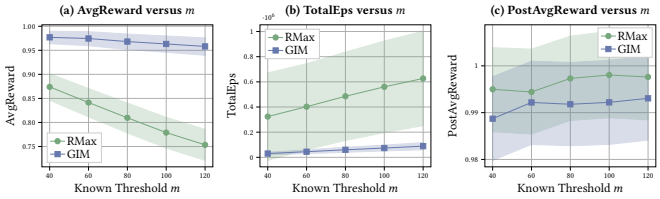

Figure 3: Comparison of AvgReward, TotalEps, and PostAvgReward of RMax and GIM on synthetic task with different known thresholds. S=20, A=10 and rank=2.

7.2.2 Properties of Dynamic Matrices. We visualize the AvgReward of GIM and RMax under varying dynamic matrix ranks r's, incoherence parameter μ_0's and condition number κ's in Figure 4.
(1) Influence of r. As Figure 4a shows, the average rewards of both GIM and RMax drop **slightly** when rank becomes higher. Even if the dynamic matrix is full-rank, the policy returned by GIM still obtains a high reward.
(2) Influence of μ_0 and κ. We see from Figure 4b and Figure 4c that GIM outperforms RMax, and does not change much with varying μ_0 or κ.

Experiments of matrix properties are conducted on synthetic tasks because it is easy to control the properties of their underlying MDPs. But the observed connections between studied properties and learning results can be extended to any other tasks. Additional experimental results including PostAvgReward, TotalEps are shown in Appendix C.3, as well as the results for μ_1.

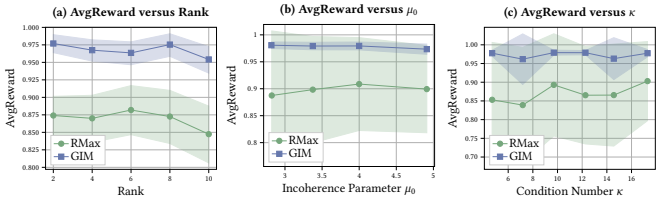

Figure 4: Comparison of AvgReward of GIM and RMax with varying ranks, μ_0 and κ. $S = 20$, $A = 10$ and $m = 40$. The generated ranks for (b) and (c) are 2 and 4 respectively.

8 CONCLUSION

This paper proposes a model-based RL algorithm called GIM. GIM utilizes the internal structures of MDPs to infer the unknown dynamics, and uses a novel exploration strategy to efficiently explore the environment. Theoretical analysis and empirical results show that GIM can reduce both sample complexity and computational complexity when combined with classic model-based algorithms. We envision incorporating our proposed techniques for multi-task RL where internal structures of MDPs can connect different tasks.

REFERENCES

[1] David Abel. 2019. simple_rl: Reproducible Reinforcement Learning in Python. (2019).

[2] Animashree Anandkumar, Rong Ge, Daniel Hsu, Sham M Kakade, and Matus Telgarsky. 2014. Tensor decompositions for learning latent variable models. *The Journal of Machine Learning Research* 15, 1 (2014), 2773–2832.

[3] Kamyar Azizzadenesheli, Alessandro Lazaric, and Animashree Anandkumar. 2016. Reinforcement learning in rich-observation MDPs using spectral methods. *arXiv preprint arXiv:1611.03907* (2016).

[4] Kamyar Azizzadenesheli, Alessandro Lazaric, and Animashree Anandkumar. 2016. Reinforcement learning of POMDPs using spectral methods. *arXiv preprint arXiv:1602.07764* (2016).

[5] Byron Boots, Sajid M Siddiqi, and Geoffrey J Gordon. 2011. Closing the learning-planning loop with predictive state representations. *The International Journal of Robotics Research* 30, 7 (2011), 954–966.

[6] Ronen I. Brafman and Moshe Tennenholtz. 2003. R-max - a General Polynomial Time Algorithm for Near-optimal Reinforcement Learning. *J. Mach. Learn. Res.* 3 (March 2003), 213–231. https://doi.org/10.1162/153244303765208377

[7] Stefano Bromuri. 2012. A Tensor Factorization Approach to Generalization in Multi-Agent Reinforcement Learning. In *2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*, Vol. 2. IEEE, 274–281.

[8] Emma Brunskill and Lihong Li. 2013. Sample complexity of multi-task reinforcement learning. *arXiv preprint arXiv:1309.6821* (2013).

[9] Emmanuel J Candes and Yaniv Plan. 2010. Matrix completion with noise. *Proc. IEEE* 98, 6 (2010), 925–936.

[10] Emmanuel J Candes and Terence Tao. 2009. The power of convex relaxation: Near-optimal matrix completion. *arXiv preprint arXiv:0903.1476* (2009).

[11] Yudong Chen. 2013. Incoherence-Optimal Matrix Completion. *Information Theory IEEE Transactions on* 61, 5 (2013), 2909–2923.

[12] Christoph Dann and Emma Brunskill. 2015. Sample Complexity of Episodic Fixed-Horizon Reinforcement Learning. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). Curran Associates, Inc., 2818–2826. http://papers.nips.cc/paper/5827-sample-complexity-of-episodic-fixed-horizon-reinforcement-learning.pdf

[13] Christoph Dann, Tor Lattimore, and Emma Brunskill. 2017. Unifying PAC and regret: Uniform PAC bounds for episodic reinforcement learning. In *Advances in Neural Information Processing Systems*. 5713–5723.

[14] Christoph Dann, Lihong Li, Wei Wei, and Emma Brunskill. 2018. Policy certificates: Towards accountable reinforcement learning. *arXiv preprint arXiv:1811.03056* (2018).

[15] Yaqi Duan, Mengdi Wang, Zaiwen Wen, and Yaxiang Yuan. 2018. Adaptive Low-Nonnegative-Rank Approximation for State Aggregation of Markov Chains. *arXiv preprint arXiv:1810.06032* (2018).

[16] Kimberly Ferguson and Sridhar Mahadevan. 2006. Proto-transfer learning in markov decision processes using spectral methods. *Computer Science Department Faculty Publication Series* (2006), 151.

[17] David Gamarnik, Quan Li, and Hongyi Zhang. 2017. Matrix Completion from $O(n)$ Samples in Linear Time. *arXiv preprint arXiv:1702.02267* (2017).

[18] Hado V. Hasselt. 2010. Double Q-learning. In *Advances in Neural Information Processing Systems 23*, J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta (Eds.). Curran Associates, Inc., 2613–2621. http://papers.nips.cc/paper/3964-double-q-learning.pdf

[19] Thomas Jaksch, Ronald Ortner, and Peter Auer. 2010. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research* 11, Apr (2010), 1563–1600.

[20] Nan Jiang. 2018. PAC reinforcement learning with an imperfect model. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[21] Nan Jiang, Akshay Krishnamurthy, Alekh Agarwal, John Langford, and Robert E Schapire. 2017. Contextual decision processes with low Bellman rank are PAC-learnable. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 1704–1713.

[22] Chi Jin, Zhuoran Yang, Zhaoran Wang, and Michael I Jordan. 2019. Provably efficient reinforcement learning with linear function approximation. *arXiv preprint arXiv:1907.05388* (2019).

[23] Sham Machandranath Kakade et al. 2003. *On the sample complexity of reinforcement learning*. Ph.D. Dissertation. University of London London, England.

[24] Michael Kearns and Satinder Singh. 2002. Near-optimal reinforcement learning in polynomial time. *Machine learning* 49, 2-3 (2002), 209–232.

[25] Michael J Kearns and Satinder P Singh. 1999. Finite-sample convergence rates for Q-learning and indirect algorithms. In *Advances in neural information processing systems*. 996–1002.

[26] Raghunandan H Keshavan, Andrea Montanari, and Sewoong Oh. 2010. Matrix completion from noisy entries. *Journal of Machine Learning Research* 11, Jul (2010), 2057–2078.

[27] Jens Kober, J Andrew Bagnell, and Jan Peters. 2013. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research* 32, 11 (2013), 1238–1274.

[28] Alessandro Lazaric, Emma Brunskill, et al. 2013. Sequential transfer in multi-armed bandit withfi nite set of models. In *Advances in Neural Information Processing Systems*. 2220–2228.

[29] Xudong Li, Mengdi Wang, and Anru Zhang. 2018. Estimation of Markov chain via rank-constrained likelihood. *arXiv preprint arXiv:1804.00795* (2018).

[30] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[31] Hao Yi Ong. 2015. Value function approximation via low-rank models. *arXiv preprint arXiv:1509.00061* (2015).

[32] A Strehl and M Littman. 2004. Exploration via model based interval estimation. In *International Conference on Machine Learning*. Citeseer.

[33] Alexander L Strehl, Lihong Li, and Michael L Littman. 2012. Incremental model-based learners with formal learning-time guarantees. *arXiv preprint arXiv:1206.6870* (2012).

[34] Alexander L. Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L. Littman. 2006. PAC Model-free Reinforcement Learning. In *Proceedings of the 23rd International Conference on Machine Learning (ICML '06)*. ACM, New York, NY, USA, 881–888. https://doi.org/10.1145/1143844.1143955

[35] Alexander L. Strehl and Michael L. Littman. 2005. A Theoretical Analysis of Model-Based Interval Estimation. In *Proceedings of the 22Nd International Conference on Machine Learning (ICML '05)*. ACM, New York, NY, USA, 856–863. https://doi.org/10.1145/1102351.1102459

[36] Alexander L Strehl and Michael L Littman. 2008. An analysis of model-based interval estimation for Markov decision processes. *J. Comput. System Sci.* 74, 8 (2008), 1309–1331.

[37] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[38] István Szita and Csaba Szepesvári. 2010. Model-based reinforcement learning with nearly tight exploration complexity bounds. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. 1031–1038.

[39] Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*.

[40] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3-4 (1992), 279–292.

[41] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3-4 (1992), 229–256.

[42] Lin F Yang and Mengdi Wang. 2019. Reinforcement Leaning in Feature Space: Matrix Bandit, Kernels, and Regret Bound. *arXiv preprint arXiv:1905.10389* (2019).

[43] Lin F Yang and Mengdi Wang. 2019. Sample-optimal parametric q-learning with linear transition models. *arXiv preprint arXiv:1902.04779* (2019).

[44] Jing Zhang, Bowen Hao, Bo Chen, Cuiping Li, Hong Chen, and Jimeng Sund. 2019. Hierarchical Reinforcement Learning for Course Recommendation in MOOCs. *Psychology* 5, 4.64 (2019), 5–65.