Efficient Planning under Partial Observability with Unnormalized Q Functions and Spectral Learning

Tianyu Li[*†‡] **Bogdan Mazoure**[*†‡] **Doina Precup**[†‡¶‖] **Guillaume Rabusseau**[‡§¶]

Abstract

Learning and planning in partially-observable domains is one of the most difficult problems in reinforcement learning. Traditional methods consider these two problems as independent, resulting in a classic two-stage paradigm: first learn the environment dynamics and then compute the optimal policy accordingly. This approach, however, disconnects the reward information from the learning of the environment model and can consequently lead to representations that are sample inefficient and time consuming for planning purpose. In this paper, we propose a novel algorithm that incorporate reward information into the representations of the environment to unify these two stages. Our algorithm is closely related to the spectral learning algorithm for predicitive state representations and offers appealing theoretical guarantees and time complexity. We empirically show on two domains that our approach is more sample and time efficient compared to classical methods.

1 Introduction

A common assumption in reinforcement learning (RL) is that the agent has the knowledge of the entire dynamics of the environment, including the state space, transition probabilities, and a reward model. However, in many real world applications, this assumption may not always be valid. Instead, the environment is often *partially observable*, meaning that the true state of the system is not completely visible to the agent. This partial observability can result in numerous difficulties in terms of learning the dynamics of the environment and planning to maximize returns.

Partially observable Markov decision Processes (POMDPs) [Sondik, 1978, Cassandra et al., 1994] provide a formal framework for single-agent planning under a partially observable environment. In contrast with MDPs, agents in POMDPs do not have direct access to the state space. Instead of observing the states, agents only have access to observations and need to operate on the so-called *belief states*, which describe the distribution over the state space given some past trajectory. Therefore, POMDPs model the dynamics of an RL environment in a latent variable fashion and explicitly reason about uncertainty in both action effects and state observability [Boots et al., 2011]. Planning under a POMDP has long been considered a difficult problem [Kaelbling et al., 1998]. To perform *exact* planning under a POMDP, one common approach is to optimize the value function over all possible belief states. Value iteration for POMDPs [Sondik, 1978] is one particular example of this approach. However, due to the curse of dimensionality and curse of history [Pineau et al., 2006], this method is often computationally intractable for most realistic POMDP planning problems [Boots et al., 2011].

As an alternative to exact planning, the family of predictive state representations (PSRs) has attracted many interests. In fact, PSRs are no weaker than POMDPs in terms of their representation power [Littman and Sutton, 2002], and there are many efficient algorithms to estimate PSRs and their variants relying on likelihood based algorithms [Singh et al., 2003, 2004] or spectral learning techniques [Boots et al., 2011, Hamilton et al., 2013]. However, to plan with PSRs is not straightforward. Typically, a two-stage process is applied to discover the optimal policy with PSRs: first, the model that describes the environment needs to be estimated. This includes learning a PSR model in an unsupervised fashion, and a reward function based on the learned model. After this stage, a planning method is used to discover the optimal policy. Several planning algo-

Proceedings of the 23[rd]International Conference on Artificial Intelligence and Statistics (AISTATS) 2020, Palermo, Italy. PMLR: Volume 108.

[*] Equal Contribution [‡] Quebec AI Institute (Mila) [†] McGill University [§] DIRO, Université de Montréal [¶] Canadian Institute for Advanced Research (CIFAR) AI Chair [‖] Deepmind Montreal

rithms can be used for the second stage of this process. For example in [Boots et al., 2011, Izadi and Precup, 2008], point based value iteration (PBVI) [Pineau et al., 2003] is used to obtain an approximation of the value function and hence the optimal policy; in [Hamilton et al., 2014], the authors use the fitted-Q method [Ernst et al., 2005] to iteratively regress Bellman updates on the learned state representations, thus approximating the action value function.

However, despite numerous successes, this two-stage process still suffers from significant drawbacks. To begin with, the PSRs parameters are learned independently from the reward information, resulting in a less efficient representation for planning. Secondly, planning with PSRs often involves multiple stages of regression, and these extra steps of approximation can be detrimental for obtaining the optimal policy. Last but not least, the planning methods for PSRs are often iterative methods that can be very time consuming.

In this work, we propose an alternative to the traditional paradigm of planning in partially observable environments. Inspired by PSRs, our method leverages the spectral learning algorithm for subspace identification, treating the environment as a latent variable model. However, instead of explicitly learning the dynamics of the environment, we learn a function that is proportional to the action value function, which we call *unnormalized Q function* (UQF). In doing so, we incorporate the reward information into the dynamics in a supervised learning fashion, which unifies the two stages of the classic learning-planning paradigm for POMDPs. To some extent, our approach effectively learns a goal-oriented representation of the environment and therefore is more sample efficient compared to the classic methods. Our algorithm relies on the spectral learning algorithm for *weighted finite automata* (WFAs), which are an extension of PSRs that can model not only probability distributions, but arbitrary real-valued functions. Our method inherits the benefits of spectral learning: it provides a consistent estimation of the UQF and is computationally more efficient than EM based methods. Furthermore, planning with PSRs usually requires multiple steps and is often based on iterative methods, which can be time consuming. In contrast, our algorithm directly learns a policy in one step, offering a more time efficient method. In addition, we also adopt *matrix compressed sensing* techniques to extend this approach to complex domains. This technique has also been used in PSRs based methods to overcome similar problems [Hamilton et al., 2014].

We conduct experiments on partially observable grid world and S-PocMan environment [Hamilton et al., 2014] where we compare our approach with classical PSR based methods. In both domains, our approach is significantly more data-efficient than PSR based methods with considerably smaller running time.

2 Background

In this section, we will introduce some basic RL concepts, including partially observable Markov decision processes (POMDPs), predictive state representations (PSRs) and their variants as well as the notion of WFAs. We will also introduce the spectral learning algorithms for WFAs.

2.1 Partially Observable Markov Decision Processes (POMDPs)

Markov decision processes have been widely applied in the field of reinforcement learning. A Markov decision process (MDP) of size k is characterized by a 6-tuple $\langle \boldsymbol{\mathcal{T}}, \mathbf{r}, \mathcal{A}, \mathcal{S}, \boldsymbol{\mu}, \gamma \rangle$ where $\boldsymbol{\mathcal{T}} \in [0,1]^{\mathcal{S}\times\mathcal{A}\times\mathcal{S}}$ is the transition probability; $\boldsymbol{\mu} \in [0,1]^{\mathcal{S}}$ is the initial state distribution; $\mathbf{r} \in \mathbb{R}^{\mathcal{S}}$ is the reward vector over states; $\gamma \in [0,1)$ is the discount factor; $\mathcal{S}$ is the set of states and $\mathcal{S} = \{s^1, \cdots s^k\}$ and $\mathcal{A}$ is the set of actions. The goal of an RL task is often to learn a policy that governs the actions of the agent to maximize the *accumulated discounted rewards* (return) in the future. A policy in an MDP environment is defined as $\boldsymbol{\Pi} \in [0,1]^{\mathcal{S}\times\mathcal{A}}$. $\boldsymbol{\Pi}$ operates at the state level. At each timestep, the optimal action is selected probabilistically with respect to $\boldsymbol{\Pi}$ given the state of the current step. The agent then move to the next state depending on the corresponding transition matrix indexed by a and collect potential rewards from the state.

However, in practice, it is rarely the case that we can observe the exact state of the agent. For example, in the game of poker, the player only knows the cards at hands and this information alone does not determine the exact state of the game. Partially observable Markov decision processes (POMDPs) were introduced to model this type of problems. Under the POMDP setting, the true state space of the model is hidden from the agent through partial observability. That is, an observation o_t is obtained probabilistically based on the agent's current state and the observation emission probability. A *partially observable Markov decision process* (POMDP) is characterized by an 8-tuple $\langle \boldsymbol{\mathcal{T}}, \boldsymbol{\mathcal{O}}, \mathbf{r}, \mathcal{A}, \mathcal{O}, S, \boldsymbol{\mu}, \gamma \rangle$, where, $\mathcal{O}$ is a set of observations and $\boldsymbol{\mathcal{O}} \in \mathbb{R}^{S\times A\times O}$ is the observation emission probability and the rest parameters follow the definitions in MDPs.

As the agent cannot directly observe which state it is at, one classic problem in POMDP is to compute the *belief state* $\boldsymbol{b}(h) \in \mathbb{R}^{\mathcal{S}}$ knowing the past trajectory h. Formally, given $h = a_1 o_1 \cdots a_n o_n \in (\mathcal{A}\times\mathcal{O})^*$, we want to compute $\boldsymbol{b}(h)^\top =$

$[\mathbb{P}(s^1|h), \cdots, \mathbb{P}(s^k|h)]^\top$. This can be solved with a forward method similar to HMM [Juang and Rabiner, 1991]. Let $\tilde{\mathbf{O}}_{ao} = \text{diag}(\boldsymbol{\mathcal{O}}_{s^1,a,o}, \boldsymbol{\mathcal{O}}_{s^2,a,o}, \cdots, \boldsymbol{\mathcal{O}}_{s^k,a,o})$, $\tilde{\mathbf{M}}_a = \text{diag}(\boldsymbol{\Pi}_{s^1,a}, \boldsymbol{\Pi}_{s^2,a}, \cdots, \boldsymbol{\Pi}_{s^k,a})$ and denote $\mathbf{E}_{ao} = \tilde{\mathbf{M}}_a \boldsymbol{\mathcal{T}}_{:,a,:} \tilde{\mathbf{O}}_{ao}$. It can be shown that $\boldsymbol{b}(h)^\top = \boldsymbol{\mu}^\top \mathbf{E}_{a_1 o_1} \cdots \mathbf{E}_{a_n o_n}$ and $\boldsymbol{b}(\lambda) = \boldsymbol{\mu}^\top$, where λ denotes the empty string.

Similarly to the MDP setting, the state-level policy for POMDPs is defined by $\boldsymbol{\Pi} \in [0,1]^{\mathcal{S}\times\mathcal{A}}$, where $\boldsymbol{\Pi}_{s,a} = \mathbb{P}(a|s)$. However, due to the partial observability, the agent's true state cannot be directly observed. Nonetheless, any state-level policy implicitly induces a probabilistic policy over past trajectories, defined by $\Pi(a|h) = \sum_{s\in\mathcal{S}} \mathbb{P}(s|h)\boldsymbol{\Pi}_{s,a}$ for each $h \in (\mathcal{A}\times\mathcal{O})^*$. Similarly, every state-level policy induces a probabilistic distribution over trajectories. With a slight abuse of notation, denote the probability of a trajectory h under the policy Π by $\mathbb{P}^\Pi(h)$. Here, we assume Π is induced by a state-level policy $\boldsymbol{\Pi}$ and define $\mathbb{P}^\Pi(h) = \boldsymbol{b}(h)^\top \mathbf{1}$, where $\mathbf{1}$ is an all-one vector. To make clear of the notations, we will use $\pi : \Sigma^* \to \mathcal{A}$ for deterministic policies in the later sections.

2.2 Predictive state representations

One common approach for modelling the dynamics of a POMDP is the so-called predictive state representations (PSRs). A PSR is a model of a dynamical system in which the current state is represented as a set of predictions about the future behavior of the system [Littman and Sutton, 2002, Singh et al., 2004]. This is done by maintaining a set of action-observation pairs, called *tests*, and the representation of the current state is given by the conditional probabilities of these tests given the past trajectory, which is referred to as *history*. Although there are multiple methods to select a set of tests [Singh et al., 2003, James and Singh, 2004], it has been shown that with a large action-observation set, finding these tests can be exponentially difficult [Boots et al., 2011].

Instead of explicitly finding the tests, *transformed predictive state representations* (TPSRs) [Rosencrantz et al., 2004, Boots et al., 2011] offer an alternative. TPSRs implicitly estimate a linear transformation of the PSR via subspace methods. This approach drastically reduces the complexity of estimating a PSR model and has shown many benefits in different RL domains [Boots et al., 2011, Singh et al., 2004].

Although this approach is able to obtain a small transformed space of the original PSRs, it still faces scalability problems. Typically, one can obtain an estimate of TPSR by performing truncated SVD on the estimated *system-dynamics matrix* [Singh et al., 2004], which is indexed by histories and tests. The scalability issue arises in complex domains, which require a large number of histories and tests to form the system-dynamics matrix. As the time complexity of SVD is cubic in the number of histories and tests, the computation time explodes in these types of environments.

Compressed predictive state representations (CPSRs) [Hamilton et al., 2013] were introduced to circumvent this issue. The main idea of this approach is to project the high dimensional system-dynamics matrix onto a much smaller subspace spanned by randomly generated bases that satisfy the Johnson-Lindenstrauss (JL) lemma [Johnson and Lindenstrauss, 1984]. The projection matrices corresponding to these bases are referred to as JL matrices. Intuitively, JL matrices define a low-dimensional embeddings which approximately preserves Euclidean distance between the projected points. More formally, given a matrix $\mathbf{H} \in \mathbb{R}^{m\times n}$ and JL random projection matrices $\boldsymbol{\Phi}_1 \in \mathbb{R}^{m\times d_1}$ and $\boldsymbol{\Phi}_2 \in \mathbb{R}^{n\times d_2}$, the compressed matrix $\mathbf{H}_c$ is computed by:

$$\mathbf{H}_c = \boldsymbol{\Phi}_1^\top \mathbf{H} \boldsymbol{\Phi}_2$$

where $\mathbf{H}_c$ is the compressed matrix. The choice of random projection matrix is rather empirical and often depends on the task. Gaussian matrices [Baraniuk and Wakin, 2009] and Rademacher matrices [Achlioptas, 2003] are common choices for the random projection matrices that satisfy JL lemma. Although does not satisfy JL lemma, hashed random projection have also been shown to preserve certain kernel-functions and perform extremely well in practice [Weinberger et al., 2009, Shi et al., 2009].

2.3 Weighted finite automata (WFAs)

In fact, TPSRs (CPSRs) are a subclass of a wider family called weighted finite automata (WFAs). More precisely, TPSRs belong to *stochastic weighted finite automata* (SWFAs) [Droste et al., 2009] in the formal language community or *observable operator models* (OOMs) [Jaeger, 2000] in control theory. Further connections between SWFAs, OOMs and TPSRs are shown in [Thon and Jaeger, 2015]. WFAs are an extension to TPSRs in the sense that, instead of only computing the probabilities of trajectories, WFAs can compute functions with arbitrary scalar outputs over the given trajectories. Formally, a *weighted finite automaton* (WFA) with k states is a tuple $A = \langle \boldsymbol{\alpha}, \{\mathbf{A}_\sigma\}_{\sigma\in\Sigma}, \boldsymbol{\omega}, \Sigma\rangle$, where $\boldsymbol{\alpha}, \boldsymbol{\omega} \in \mathbb{R}^k$ are the initial and terminal weights; $\mathbf{A}_\sigma \in \mathbb{R}^{k\times k}$ is the transition matrix associated with symbol σ from alphabet Σ. Given a trajectory $x = x_1 x_2 \cdots x_n \in \Sigma^*$, a WFA A computes a function $f_A : \Sigma^* \to \mathbb{R}$ defined by:

$$f_A(x) = \alpha^\top \mathbf{A}_{x_1} \mathbf{A}_{x_2} \cdots \mathbf{A}_{x_n} \omega$$

We will denote $\mathbf{A}_{x_1}\mathbf{A}_{x_2}\cdots\mathbf{A}_{x_n}$ by $\mathbf{A}_x$ in the following sections for simplicity. For a function $f:\Sigma^*\to\mathbb{R}$, the *rank* of f is defined as the minimal number of states of a WFA computing f. If f cannot be computed by a WFA, we let $\text{rank}(f)=\infty$. In the context of TPSRs, we often let $\Sigma = A\times O$ and f_A computes the probability of the trajectory.

2.3.1 Hankel matrix

The learning algorithm of WFAs, relies on the spectral decomposition of the so-called Hankel matrix. The *Hankel matrix* $\mathbf{H}_f\in\mathbb{R}^{\Sigma^*\times\Sigma^*}$ associated with a function $f:\Sigma^*\to\mathbb{R}$ is a bi-infinite matrix with entries $(\mathbf{H}_f)_{u,v}=f(uv)$ for all words $u,v\in\Sigma^*$. The spectral learning algorithm for WFA relies on the following fundamental relation between the rank of f and the rank of the Hankel matrix $\mathbf{H}_f$ [Carlyle and Paz, 1971, Fliess, 1974]:

Theorem 2.1. *For any* $f:\Sigma^*\to\mathbb{R}$, $\text{rank}(f)=\text{rank}(\mathbf{H}_f)$.

In practice, one deals with finite sub-blocks of the Hankel matrix. Given a basis $\mathcal{B}=(\mathcal{U},\mathcal{V})\subset\Sigma^*\times\Sigma^*$, where $\mathcal{U}$ is a set of *prefixes* and $\mathcal{V}$ is a set of *suffixes*, denote the corresponding sub-block of the Hankel matrix by $\mathbf{H}_\mathcal{B}\in\mathbb{R}^{\mathcal{U}\times\mathcal{V}}$. For an arbitrary basis $\mathcal{B}=(\mathcal{U},\mathcal{V})$, define its *p-closure* by $\mathcal{B}'=(\mathcal{U}',\mathcal{V}),]$ where $\mathcal{U}'=\mathcal{U}\cup\mathcal{U}\Sigma$. It turns out that a Hankel matrix over a p-closed basis can be partitioned into $|\Sigma|+1$ blocks of the same size [Balle et al., 2014a]:

$$\mathbf{H}_{\mathcal{B}'}^\top=[\mathbf{H}_\lambda^\top|\mathbf{H}_{\sigma_1}^\top|\cdots|\mathbf{H}_{\sigma_{|\Sigma|}}^\top]$$

, where λ denotes the empty string and for each $\sigma\in\Sigma\cup\{\lambda\}$ the matrix $\mathbf{H}_\sigma\in\mathbb{R}^{\mathcal{U}\times\mathcal{V}}$ is defined by $(\mathbf{H}_\sigma)_{u,v}=f(u\sigma v)$. We say that a basis $\mathcal{B}=(\mathcal{U},\mathcal{V})$ is *complete* for the function f if the sub-block $\mathbf{H}_\mathcal{B}$ has full rank: $\text{rank}(\mathbf{H}_\mathcal{B})=\text{rank}(\mathbf{H}_f)$ and we call $\mathbf{H}_\mathcal{B}$ a *complete sub-block* of $\mathbf{H}_f$ and $\mathbf{H}_{\mathcal{B}'}$ a prefix-closure of $\mathbf{H}_\mathcal{B}$. It turns out that one can recover the WFA that realizes f via the prefix-closure of a complete sub-block of $\mathbf{H}_f$ [Balle et al., 2014a] using the *spectral learning* algorithm of WFAs.

2.3.2 Spectral learning of WFAs

It can be shown that the rank of the Hankel matrix $\mathbf{H}$ is upper bounded by the rank of f [Balle et al., 2014b]. Moreover, given a rank factorization of the Hankel matrix $\mathbf{H}=\mathbf{PS}$, it is also true that $\mathbf{H}_\sigma=\mathbf{PA}_\sigma\mathbf{S}$ for each $\sigma\in\Sigma$. The spectral learning algorithm relies on the non-trivial observation that this construction can be reversed: given any rank k factorization $\mathbf{H}_\lambda=\mathbf{PS}$, the WFA $A=\langle\boldsymbol{\alpha}_0^\top,\boldsymbol{\alpha}_\infty,\{\mathbf{A}_\sigma\}_{\sigma\in\Sigma}\rangle$ defined by

$$\boldsymbol{\alpha}_0^\top=\mathbf{P}_{\lambda,:},\quad \boldsymbol{\alpha}_\infty=\mathbf{S}_{:,\lambda},\quad \text{and } \mathbf{A}_\sigma=\mathbf{P}^+\mathbf{H}_\sigma\mathbf{S}^+,$$

is a minimal WFA computing f [Balle et al., 2014a, Lemma 4.1], where $\mathbf{H}_\sigma$ for $\sigma\in\Sigma\cup\lambda$ denote the finite matrices defined above for a prefix closed complete basis $\mathcal{B}$. In practice, we compute empirical estimates of the Hankel matrices such that $\hat{\mathbf{H}}_{u,v}=\frac{1}{|D|}\sum_{i\in|D|}\mathbb{I}_{uv}(D_i)\boldsymbol{y}_i$, where D is a dataset of trajectories, $\boldsymbol{y}$ is a vector of the outputs of D, $\mathbb{I}_{uv}(D_i)=1$ if $uv=D_i$ and zero otherwise.

In fact, the above algorithm can be readily used for learning TPSRs. In PSRs terminology, the prefixes are the histories while the suffixes are the tests and the alphabet Σ is the set of all possible action observation pairs, i.e. $\Sigma=\mathcal{A}\times\mathcal{O}$. By simply replacing Hankel matrix $\mathbf{H}$ by the system-dynamics matrix proposed in [Singh et al., 2004], one will exactly fall back to TPSRs learning algorithm [Singh et al., 2004, Boots et al., 2011].

3 Planning with Unnormalized Q Function

In this section, we will introduce our POMDP planning method. The main idea of our algorithm is to directly compute the optimal policy based on the estimation of *unnormalized Q function* that is proportional to the action value function. Moreover, the value of this function, given a past trajectory, can be computed via a WFA and it is then straight-forward to use the classical spectral learning algorithm to recover this WFA. Unlike traditional PSR methods, our approach takes advantage of the reward information by integrating the reward into the learned representations. In contrast, classical PSRs based methods construct the representations solely with the environment dynamics, completely ignoring the reward information. Consequently, our method offers a more sample efficient representation of the environment for planning under POMDPs. In addition, our algorithm only needs to construct a WFA and there is no other iterative method involved. Therefore, compared to traditional methods to plan with PSRs, our algorithm is more time efficient. Finally, with the help of compressed sensing techniques, we are able to scale our algorithm to complex domains.

3.1 Unnormalized Q function

The estimation of the action value function is of great importance for planning under POMDP. Typically, given a probabilistic sampling policy $\Pi:\mathcal{A}\times\Sigma^*\to[0,1]$, where $\Sigma=\mathcal{A}\times\mathcal{O}$, the action value function (Q function) of a given trajectory $h\in\Sigma^*$ is defined by:

$$Q^\Pi(h,a)=\mathbb{E}^\Pi(r_t+\gamma r_{t+1}+\cdots+\gamma^i r_{t+i}+\cdots|ha)$$

where $|h|=t$ and r_t is the immediate rewards collected at time step t.

Given a POMDP $\psi = \langle \boldsymbol{\mathcal{T}}, \boldsymbol{\mathcal{O}}, \mathbf{r}, \mathcal{A}, \mathcal{O}, \mathcal{S}, \boldsymbol{\mu}, \gamma \rangle$, denote the expected immediate reward collected after h by $\tilde{R}(h)$, which is defined as:

$$\tilde{R}(h) = \mathbb{E}_{s \in \mathcal{S}}(\boldsymbol{r}_s | h) = \sum_{s \in \mathcal{S}} \boldsymbol{r}_s \mathbb{P}(s|h)$$

The action value function can then be expanded to:

$$\begin{aligned} Q^{\Pi}(h,a) &= \mathbb{E}^{\Pi}(r_t + \gamma r_{t+1} + \cdots + \gamma^i r_{t+i} + \cdots | ha) \\ &= \sum_{z \in \Sigma^*} \sum_{o \in \mathcal{O}} \gamma^{|z|} \tilde{R}(haoz) \mathbb{P}^{\Pi}(haoz|ha) \\ &= \frac{\sum_{o \in \mathcal{O}} \sum_{z \in \Sigma^*} \gamma^{|z|} \tilde{R}(haoz) \mathbb{P}^{\Pi}(haoz)}{\mathbb{P}^{\Pi}(ha)} \\ &= \frac{\sum_{o \in \mathcal{O}} \sum_{z \in \Sigma^*} \gamma^{|z|} \tilde{R}(haoz) \mathbb{P}^{\Pi}(haoz)/\Pi(a|h)}{\mathbb{P}^{\Pi}(h)} \\ &:= \frac{\tilde{Q}^{\Pi}(h,a)}{\mathbb{P}^{\Pi}(h)} \end{aligned}$$

where we will refer to the function $\tilde{Q}^{\Pi}(h,a)$ as the *unnormalized Q function*(UQF). It is trivial to show that given the same trajectory h:

$$\tilde{Q}^{\Pi}(h,\cdot) \propto Q^{\Pi}(h,\cdot)$$

Therefore, we have $\text{argmax}_{a \in \mathcal{A}} Q^{\Pi}(h,a) = \text{argmax}_{a \in \mathcal{A}} \tilde{Q}^{\Pi}(h,a)$ and we can then plan according to the UQF instead of Q^{Π}.

3.2 A spectral learning algorithm for UQF

In this section, we will present our spectral learning algorithm for UQF. First, we will show that the value of a UQF given a past trajectory can be computed via a WFA. Let us denote $\sum_{z \in \Sigma^*} \gamma^{|z|} \tilde{R}(hz) \mathbb{P}^{\Pi}(hz)$ by $\tilde{V}^{\Pi}(h)$, we have:

$$\tilde{Q}^{\Pi}(h,a) = \frac{\sum_{o \in \mathcal{O}} \tilde{V}^{\Pi}(hao)}{\Pi(a|h)}$$

Assume the probabilistic sampling policy Π is given, then we only need to compute the value of the function $\tilde{V}^{\Pi}(hao)$. As a special case, if Π is a random policy that uniformly select the actions, we can replace the term $\Pi(a|h)$ by 1 without affecting the learned policy.

It turns out that the function $\tilde{V}^{\Pi}$ can be computed by a WFA. To show that this is true, we first introduce the following lemma stating that the function $\tilde{R}(h)\mathbb{P}^{\Pi}(h)$ can be computed by a WFA B:

Lemma 3.1. *Given a POMDP $\psi = \langle \boldsymbol{\mathcal{T}}, \boldsymbol{\mathcal{O}}, \mathbf{r}, \mathcal{A}, \mathcal{O}, S, \boldsymbol{\mu}, \gamma \rangle$ of size k and a sampling policy Π induced by $\boldsymbol{\Pi} \in [0,1]^{\mathcal{S} \times \mathcal{A}}$, there exists a WFA $B = \langle \boldsymbol{\beta}, \{\mathbf{B}_\sigma\}_{\sigma \in \Sigma}, \boldsymbol{\tau} \rangle$ with k states that realizes the function $g(h) = \tilde{R}(h)\mathbb{P}^{\Pi}(h)$, where $\Sigma = \mathcal{A} \times \mathcal{O}$ and $h \in \Sigma^*$.*

In fact, we can show that the function $\tilde{V}^{\Pi}$ can be computed by another WFA A, and one can easily convert B to A.

Theorem 3.2. *Given a POMDP ψ of size k, a sampling policy Π and a WFA $B = \langle \boldsymbol{\beta}^\top, \{\mathbf{B}_\sigma\}_{\sigma \in \Sigma}, \boldsymbol{\tau} \rangle$ realizing the function $g : h \mapsto \tilde{R}(h)\mathbb{P}^{\Pi}(h)$ such that the spectral radius $\rho(\gamma \sum_{\sigma \in \Sigma} \mathbf{B}_\sigma) < 1$, the WFA $A = \langle \boldsymbol{\beta}^\top, \{\mathbf{B}_\sigma\}_{\sigma \in \Sigma}, (\mathbf{I} - \gamma \sum_{\sigma \in \Sigma} \mathbf{B}_\sigma)^{-1} \boldsymbol{\tau} \rangle$ of size k realizes the function $\tilde{V}^{\Pi}(h) = \sum_{z \in \Sigma^*} \gamma^{|z|} \tilde{R}(hz) \mathbb{P}^{\Pi}(hz)$.*

Note the condition on the spectral radius implies that the function $\tilde{V}$ needs be bounded. This condition is in fact very easy to satisfy as in RL we typically assume that the value function $V(h) = \sum_{z \in \Sigma^*} \gamma^{|z|} \tilde{R}(hz)\, \mathbb{P}(z|h)$ is bounded.

Therefore, in order to compute the function $\tilde{Q}^{\Pi}$, we only need to learn a WFA that computes the function g. Following the classical spectral learning algorithm, we present our learning algorithm of POMDP planning in Algorithm 1. In fact, it has been shown that the spectral learning algorithm of WFAs is statistically consistent [Balle et al., 2014a]. Therefore our approximation of the function $\tilde{Q}^{\Pi}$ is consistent with respect to sample sizes.

Algorithm 1: Spectral algorithm for UQF

input : A set of actions $\mathcal{A}$, a set of observations $\mathcal{O}$, discount factor γ, a probabilistic sampling policy Π, training trajectories D and their immediate reward $\boldsymbol{y}$, rank of the truncated SVD k

Result: A new deterministic policy function $\pi^{new} : \Sigma^* \to \mathcal{A}$

1. For a prefix u and a suffix v, we estimate its value in the Hankel matrix as $\hat{\mathbf{H}}_{u,v} = \sum_{i=0}^{|D|} \mathbb{I}_{uv}(D_i) \boldsymbol{y}_i / |D|$, where $|D|$ is the cardinality of the training set D, $\Sigma = \mathcal{A} \times \mathcal{O}$.
2. Perform truncated SVD of rank k on the estimated Hankel matrix: $\hat{\mathbf{H}} \simeq \mathbf{U}\mathbf{D}\mathbf{V}^\top$
3. Recover the WFA $B = \langle \boldsymbol{\beta}^\top, \{\mathbf{B}_\sigma\}_{\sigma \in \Sigma}, \boldsymbol{\tau} \rangle$ realizing the function $g(h) = \tilde{R}(h)\mathbb{P}(h)$: $\boldsymbol{\beta}^\top = (\mathbf{U}\mathbf{D})_{\lambda,:}$, $\boldsymbol{\tau} = \mathbf{V}^\top_{:,\lambda}$, $\mathbf{B}_\sigma = (\mathbf{U}\mathbf{D})^+ \hat{\mathbf{H}}_\sigma \mathbf{V}^\top$
4. Convert the WFA B to $A = \langle \boldsymbol{\alpha}^\top, \{\mathbf{A}_\sigma\}_{\sigma \in \Sigma}, \boldsymbol{\omega} \rangle$, which realizes the function $\tilde{V}^{\Pi}$, following Theorem 6.2, we have $\boldsymbol{\alpha}^\top = \boldsymbol{\beta}^\top$, $\mathbf{A}_\sigma = \mathbf{B}_\sigma$, and $\boldsymbol{\omega} = (\mathbf{I} - \gamma \sum_{\sigma \in \Sigma} \mathbf{B}_\sigma)^{-1} \boldsymbol{\tau}$.
5. **Return** A new deterministic policy function π^{new}, such that given $h \in \Sigma^*$, $\pi^{new}(h) = \text{argmax}_{a \in \mathcal{A}} \frac{\sum_{o \in \mathcal{O}} \boldsymbol{\alpha}^\top \mathbf{A}_h \mathbf{A}_{ao} \boldsymbol{\omega}}{\Pi(a|h)}$

Algorithm 2: Scalable spectral algorithm for UQF

input : A set of actions $\mathcal{A}$, a set of observations $\mathcal{O}$, discount factor γ, a probabilistic sampling policy Π, training trajectories D and their immediate reward $\boldsymbol{y} \in \mathbb{R}^{|D|}$, the rank of the truncated SVD k, a set of prefixes $\mathcal{U}$, a set of suffixes $\mathcal{V}$, mapping functions for both prefixes and suffixes, ϕ_U, ϕ_V, and the corresponding projection matrix $\boldsymbol{\Phi}_{\mathcal{U}}$

Result: A new deterministic policy function $\pi^{new} : \Sigma^* \to \mathcal{A}$

1. Compute the compressed estimation of Hankel matrices:

$$\hat{\mathbf{c}}_{\mathcal{U}} = \sum_{i=0}^{|D|} \sum_{u \in \mathcal{U}} \mathbb{I}_v(D_i) \boldsymbol{y}_i \phi_U(u)$$

$$\hat{\mathbf{C}}_{\mathcal{U}\mathcal{V}} = \sum_{i=0}^{|D|} \sum_{u,v \in \mathcal{U}\times\mathcal{V}} \mathbb{I}_{uv}(D_i) \boldsymbol{y}_i (\phi_U(u) \otimes \phi_V(v))$$

2. Perform truncated SVD on the estimated Hankel matrix with rank k: $\hat{\mathbf{C}}_{\mathcal{U}\mathcal{V}} \simeq \mathbf{U}\mathbf{D}\mathbf{V}^\top$

3. Recover the WFA $B = \langle \boldsymbol{\beta}^\top, \{\mathbf{B}_\sigma\}_{\sigma\in\Sigma}, \boldsymbol{\tau}\rangle$ realizing the function $g(h) = \tilde{R}(h)\mathbb{P}(h)$:

$$\boldsymbol{\beta}^\top = \boldsymbol{e}^\top \mathbf{U}\mathbf{D}, \qquad \boldsymbol{\tau} = \mathbf{D}^{-1}\mathbf{U}^\top \hat{\mathbf{c}}_{\mathcal{U}}$$

$$\mathbf{B}_\sigma = \sum_{i=0}^{|D|} \sum_{u,v \in \mathcal{U}\times\mathcal{V}} \mathbb{I}_{u\sigma v}(D_i) \boldsymbol{y}_i [\mathbf{D}^{-1}\mathbf{U}^\top \phi(u) \otimes \mathbf{V}^\top \phi(v)]$$

where $\boldsymbol{e}$ is a vector s.t. $\boldsymbol{e}^\top \boldsymbol{\Phi}_{\mathcal{U}}^\top = (1, 0, \cdots, 0)^\top$

4. Following Theorem 6.2, convert the WFA B to $A = \langle \boldsymbol{\alpha}^\top, \{\mathbf{A}_\sigma\}_{\sigma\in\Sigma}, \boldsymbol{\omega}\rangle$.

5. **Return** A new deterministic policy function π^{new} defined by $\pi^{new}(h) = \text{argmax}_{a\in\mathcal{A}} \frac{\sum_{o\in\mathcal{O}} \boldsymbol{\alpha}^\top \mathbf{A}_h \mathbf{A}_{ao} \boldsymbol{\omega}}{\Pi(a|h)}$

3.3 Scalable learning of UQF

Now we have established the spectral learning algorithm for UQF. However, similar to the spectral learning algorithm for TPSRs, one can immediately observe that both time and storage complexity are the bottleneck of this algorithm. For complex domains, in order to obtain a complete sub-block of the Hankel matrix, one will need large amount of prefixes and suffixes to form a basis and the classical spectral learning will become intractable.

By projecting matrices down to low-dimensional spaces via randomly generated bases, *matrix compressed sensing* has been widely applied in matrix compression field. In fact, previous work have successfully applied matrix sensing techniques to TPSRs [Hamilton et al., 2013] and developed an efficient online algorithm for learning TPSRs [Hamilton et al., 2014]. Here, we adopt a similar approach.

Assume that we are given a set of prefixes $\mathcal{U}$ and suffixes $\mathcal{V}$ and two independent random full-rank Johnson-Lindenstrauss (JL) projection matrices $\boldsymbol{\Phi}_{\mathcal{U}} \in \mathbb{R}^{\mathcal{U}\times d_{\mathcal{U}}}$, and $\boldsymbol{\Phi}_{\mathcal{V}} \in \mathbb{R}^{\mathcal{V}\times d_{\mathcal{V}}}$, where $d_{\mathcal{U}}$ and $d_{\mathcal{V}}$ are the projection dimension for the prefixes and suffixes. In this work, we use Gaussian projection matrices for $\boldsymbol{\Phi}_{\mathcal{U}}$ and $\boldsymbol{\Phi}_{\mathcal{V}}$, which contain i.i.d. entries from the distribution $\mathcal{N}(0, 1/d_{\mathcal{U}})$ and $\mathcal{N}(0, 1/d_{\mathcal{V}})$, respectively.

Let us now define two injective functions over prefixes and suffixes: $\phi_{\mathcal{U}} : \mathcal{U} \to \mathbb{R}^{d_{\mathcal{U}}}$ and $\phi_{\mathcal{V}} : \mathcal{V} \to \mathbb{R}^{d_{\mathcal{V}}}$, where for all $u \in \mathcal{U}$ and $v \in \mathcal{V}$, we have $\phi(u) = \boldsymbol{\Phi}_{u,:}$ and $\phi(v) = \boldsymbol{\Phi}_{v,:}$. The core step of our algorithm is to obtain the compressed estimation of the Hankel matrix, denoted by $\hat{\mathbf{C}}_{\mathcal{U},\mathcal{V}}$ associated with the function $\tilde{R}(h)\mathbb{P}(h)$ for all $h \in \Sigma^*$. Formally, we can obtain $\hat{\mathbf{C}}_{\mathcal{U},\mathcal{V}}$ by:

$$\begin{aligned}\hat{\mathbf{C}}_{\mathcal{U},\mathcal{V}} &= \boldsymbol{\Phi}_{\mathcal{U}}^\top \mathbf{H} \boldsymbol{\Phi}_{\mathcal{V}} \\ &= \sum_{i=0}^{|D|} \sum_{u,v \in \mathcal{U}\times\mathcal{V}} \mathbb{I}_{uv}(D_i) \boldsymbol{y}_i (\phi_U(u) \otimes \phi_V(v))\end{aligned}$$

where D is the training dataset, containing all sampled trajectories, $\boldsymbol{y}$ is the vector of immediate rewards. Then, after performing the truncated SVD of $\hat{\mathbf{C}}_{\mathcal{U},\mathcal{V}} \simeq \mathbf{U}\mathbf{D}\mathbf{V}^\top$ of rank k, we can compute the transition matrix for the WFA by:

$$\begin{aligned}\mathbf{B}_\sigma &= (\mathbf{U}\mathbf{D})^+ \hat{\mathbf{C}}_{\mathcal{U}\sigma\mathcal{V}} \mathbf{V} \\ &= \sum_{i=0}^{|D|} \sum_{u,v \in \mathcal{U}\times\mathcal{V}} \mathbb{I}_{u\sigma v}(D_i) \boldsymbol{y}_i [(\mathbf{U}\mathbf{D})^+ \phi(u) \otimes \mathbf{V}^\top \phi(v)]\end{aligned}$$

We present the complete method in Algorithm 2. Instead of iterative sweeping through dataset like most planning methods do, one can build an UQF in just two passes of data: one for building the compressed Hankel, one for recovering the parameters. More precisely, let L denote the maximum length of a trajectory in the dataset D, then the time complexity of our algorithm is $O(L|D|)$ [Hamilton et al., 2014], and there is no extra planning time needed. In contrast, fitted-Q algorithm alone requires $O(TL|D|log(L|D|))$ only for the planning stage, where T is the expected number of the fitted-Q iterations. Therefore, in terms of time complexity, our algorithm is linear to the number of trajectories, leading to a very efficient algorithm.

3.4 Policy iteration

Policy iteration has been widely applied in both MDP and POMDP settings [Bellman, 1957, Sutton et al., 1998], and have shown benefits from both empirical and theoretical perspectives [Bellman, 1957]. It is very

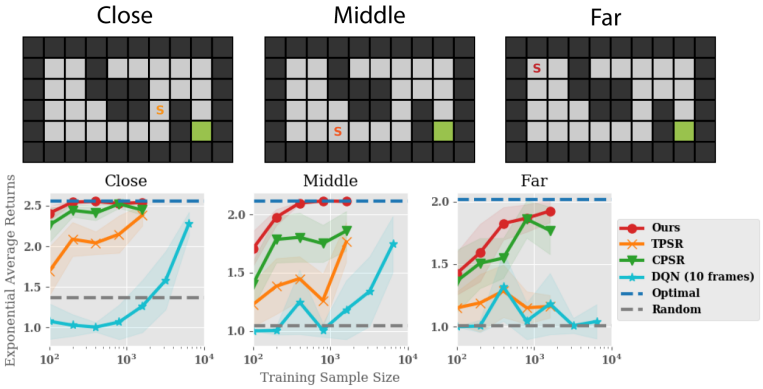

Figure 1: Experiments on three grid world tasks. The plots show the accumulated discounted rewards (returns) over 1,000 test episodes of length 100. The discount factor for computing returns is set to 0.99

natural to apply policy iteration to our algorithm, since we directly learn a policy from data. The policy iteration algorithm is listed in Algorithm 3. Note that for re-sampling, we convert our learned deterministic policy to a probabilistic one in an ϵ-greedy fashion.

Algorithm 3: Policy iteration for UQF

input : An initial deterministic policy π, ϵ-greedy factor ϵ, a decay rate for ϵ-greedy $\eta > 1$, number of policy iterations n, number of trajectories N

Result: The final policy function $\pi^{fin} : \Sigma^* \to \mathcal{A}$

1. Convert the deterministic policy π to a probabilistic policy Π in an ϵ-greedy fashion: at each step, with probability $1 - \epsilon$ select the optimal action according to π, with probability ϵ select a random action.*
2. Sample N trajectories based on policy Π
3. Execute Algorithm 1 or Algorithm 2 and obtain the corresponding new policy π^{new}.
4. $\pi^{new} \to \pi$, $\epsilon/\eta \to \epsilon$
5. Repeat all the above for n times.
6. **Return** The final policy function $\pi^{fin} = \pi^{new}$

4 Experiments

To assess the performance of our method, we conducted experiments on two domains: a toy grid world environment and the S-Pocman game [Hamilton et al., 2014]. We use TPSR/CPSR + fitted-Q as our baseline method. Our experiments have shown that indeed, in terms of sample complexity and time complexity, we outperm the classical two-stage algorithms.

4.1 Grid world experiment

The first benchmark for our method is the simple grid world environment shown in Fig. 1. The agent starts in the tile labeled S and must reach the green goal state. At each time step, the agent can only perceive the number of surrounding walls and proceeds to execute one of the four actions: go up, down, left or right. To make the environment stochastic, with probability 0.2, the execution of the action will fail, resulting instead in a random action at the current time step. The reward function in this navigation task is sparse: the agent receives no reward until it reaches the termination state. We ran three variants of the aforementioned grid world, each corresponding to a different starting state. As one can imagine, the further away the goal state is from the starting state, the harder the task becomes.

We used a random policy to generate training data, which consisted of trajectories of length up to 100. To evaluate the policy learned by the different algorithms, we let the agent execute the learned policy for 1,000 episodes and computed the average accumulated discounted rewards, with discount factor being 0.99. The maximum length for test episodes was also set to 100. Hyperparameters were selected using cross-validation (i.e. the number of rows and columns in the Hankel matrices, the rank for SVD and γ). As a baseline, we use the classical TPSRs and CPSRs as the learning method for the environment, and fitted-Q algorithm as the planning algorithm, as well as a deep Q-network (DQN). We use a three layers MLP of size 50*128*128 to approximate the Q function and Adam optimizer with 10^{-4} learning rate. The DQN is fitted on 10 consecutive observation vectors (number of walls around agent, same as all other methods) and tested with 1,000 test episodes of maximum length 100, as to reproduce the same experimental setup as the other methods.

*Note for the first iteration, one can set $\epsilon = 1$, resulting in a pure random sampling policy.

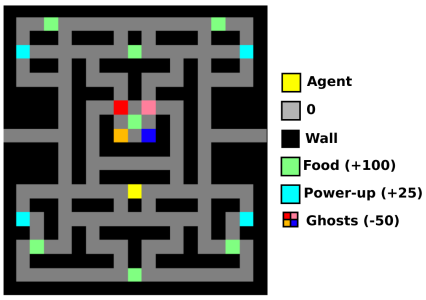

Figure 2: S-PocMan domain

We performed a hyperparameter search for all baseline methods using cross validation. In addition, we report the rewards collected by a random policy as well as the optimal policy for comparison.

Results on this toy domain (see Figure 1) highlight the sample and time efficiency achieved by our method. Indeed, our algorithm outperforms the classical CPSR+fitted-Q method in all three domains, notably achieving better performance in small data regime, showing significant sample efficiency. Furthermore, it is clear that our algorithm reaches consistently to the optimal policy as sample size increases. In addition, our methods are much faster than other compared methods. For example, for the experiment with 800 samples, to achieve similar results, our method is approximately 100 times faster compared to CPSR+fitted-Q. As expected, DQN requires a lot more data to achieve competitive results on the partially observable grid world domain than the spectral approaches.

4.2 S-PocMan domain

For the second experiment, we show the results on the S-PocMan environment [Hamilton et al., 2014]. The partially observable version of the classical game Pacman was first introduced by Silver and Veness [Silver and Veness, 2010] and is referred to as PocMan. In this domain, the agent needs to navigate through a grid world to collect food and avoid being captured by the ghosts. It is an extremely large partially observable domain with 10^{56} states [Veness et al., 2011]. However, Hamilton et al. showed that if one were to treat the partially observable environment as if it was fully observable, a simple memoryless controller can perform extremely well under this set-up, due to extensive reward information [Hamilton et al., 2014]. Hence, they proposed a harder version of PocMan, called S-PocMan. In this new domain, they drop the parts of the observation vector that allow the agent to sense the direction of the food and greatly sparsify the amount of food in the grid world, therefore making the environment more partially observable. In this experiment, we only used the combination CPSR+fitted Q for our baseline algorithm, as TPSR can not scale to the large size of this environment. Similarly to the grid world experiment, we select the best hyperparameters through cross validation. The discount factor for computing returns was set to be 0.99999 in all runs. Table 1 shows the runtime and average return for both our algorithm and the baseline method. One can see that UQF achieves better performance compared to CPSR+fitted-Q. Moreover, UQF exhibits significant reduction in running time: about 200 times faster than CPSR+fitted-Q. Note that building CPSR takes similar amount of time to our method, however, the extra iterative fitted-Q planning algorithm takes considerably more time to converge, as our analysis showed in section 3.3.

Table 1: Training time for one policy iteration and averaged accumulated discounted rewards on S-PocMan trained on 500 trajectories.

Method	Fitted-Q Iterations	Time (s)	Returns
UQF	-	**2**	**-92**
CPSR	400	489	-101
	100	116	-109
	50	60	-150
	10	15	-200

5 Conclusion

In this paper, we propose a novel learning and planning algorithm for partially observable environments. The main idea of our algorithm relies on the estimation of the unnormalized Q function with the spectral learning algorithm. Theoretically, we show that in POMDP, UQF can be computed via a WFA and consequently can be provably learned from data using the spectral learning algorithm for WFAs. Moreover, UQF combines the learning and planning phases of reinforcement learning together, and learns the corresponding policy in one step. Therefore, our method is more sample efficient and time efficient compared to traditional POMDP planning algorithms. This is further shown in the experiments on the grid world and S-PocMan environments.

Future work include exploring some theoretic properties of this planning approach. For example, a first step would be to obtain convergence guarantees for policy iteration based on the UQF spectral learning algorithm. In addition, our approach could be extended to the multitask setting by leveraging the multi-task learning framework for WFAs proposed in [Rabusseau et al., 2017]. Readily, since we combine the environment dynamics and reward information together, our approach should be able to deal with partially shared environment and reward structure, leading to a potentially flexible multi-task RL framework.

Acknowledgements

This research is supported by the Canadian Institute for Advanced Research (CIFAR AI chair program) and Fonds de Recherche du Québec – Nature et technologies (no. 271273). We also thank Compute Canada and Calcul Québec for the computing resources.

References

Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of computer and System Sciences*, 66(4):671–687, 2003.

Borja Balle, Xavier Carreras, Franco M Luque, and Ariadna Quattoni. Spectral learning of weighted automata. *Machine learning*, 96(1-2):33–63, 2014a.

Borja Balle, William Hamilton, and Joelle Pineau. Methods of moments for learning stochastic languages: Unified presentation and empirical comparison. In *International Conference on Machine Learning*, pages 1386–1394, 2014b.

Richard G Baraniuk and Michael B Wakin. Random projections of smooth manifolds. *Foundations of computational mathematics*, 9(1):51–77, 2009.

Richard Bellman. Dynamic programming. *Princeton University Press*, 1957.

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

Jack W. Carlyle and Azaria Paz. Realizations by stochastic finite automata. *Journal of Computer and System Sciences*, 5(1):26–40, 1971.

Anthony R Cassandra, Leslie Pack Kaelbling, and Michael L Littman. Acting optimally in partially observable stochastic domains. In *Association for the Advancement of Artificial Intelligence*, 1994.

Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of weighted automata*. Springer Science & Business Media, 2009.

Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(Apr):503–556, 2005.

Michel Fliess. Matrices de hankel. *Journal de Mathématiques Pures et Appliquées*, 53(9):197–222, 1974.

William Hamilton, Mahdi Milani Fard, and Joelle Pineau. Efficient learning and planning with compressed predictive states. *The Journal of Machine Learning Research*, 15(1):3395–3439, 2014.

William L Hamilton, Mahdi Milani Fard, and Joelle Pineau. Modelling sparse dynamical systems with compressed predictive state representations. In *International Conference on Machine Learning*, pages 178–186, 2013.

Masoumeh T Izadi and Doina Precup. Point-based planning for predictive state representations. In *Conference of the Canadian Society for Computational Studies of Intelligence*, pages 126–137. Springer, 2008.

Herbert Jaeger. Observable operator models for discrete stochastic time series. *Neural Computation*, 12(6): 1371–1398, 2000.

Michael R James and Satinder Singh. Learning and discovery of predictive state representations in dynamical systems with reset. In *Proceedings of the twenty-first international conference on Machine learning*, page 53. ACM, 2004.

William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

Biing Hwang Juang and Laurence R Rabiner. Hidden markov models for speech recognition. *Technometrics*, 33(3):251–272, 1991.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

Michael L Littman and Richard S Sutton. Predictive representations of state. In *Advances in neural information processing systems*, pages 1555–1561, 2002.

Joelle Pineau, Geoff Gordon, Sebastian Thrun, et al. Point-based value iteration: An anytime algorithm for pomdps. In *International Joint Conference on Artificial Intelligence*, volume 3, pages 1025–1032, 2003.

Joelle Pineau, Geoffrey Gordon, and Sebastian Thrun. Anytime point-based approximations for large pomdps. *Journal of Artificial Intelligence Research*, 27:335–380, 2006.

Guillaume Rabusseau, Borja Balle, and Joelle Pineau. Multitask spectral learning of weighted automata. In *Advances in Neural Information Processing Systems*, pages 2588–2597, 2017.

Matthew Rosencrantz, Geoff Gordon, and Sebastian Thrun. Learning low dimensional predictive representations. In *Proceedings of the twenty-first international conference on Machine learning*, page 88. ACM, 2004.

Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alex Smola, and SVN Vishwanathan. Hash kernels for structured data. *Journal of Machine Learning Research*, 10(Nov):2615–2637, 2009.

David Silver and Joel Veness. Monte-carlo planning in large pomdps. In *Advances in neural information processing systems*, pages 2164–2172, 2010.

Satinder Singh, Michael R James, and Matthew R Rudary. Predictive state representations: A new theory for modeling dynamical systems. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 512–519. AUAI Press, 2004.

Satinder P Singh, Michael L Littman, Nicholas K Jong, David Pardoe, and Peter Stone. Learning predictive state representations. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 712–719, 2003.

Edward J Sondik. The optimal control of partially observable markov processes over the infinite horizon: Discounted costs. *Operations research*, 26(2):282–304, 1978.

Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

Michael Thon and Herbert Jaeger. Links between multiplicity automata, observable operator models and predictive state representations: a unified learning framework. *The Journal of Machine Learning Research*, 16(1):103–147, 2015.

Joel Veness, Kee Siong Ng, Marcus Hutter, William Uther, and David Silver. A monte-carlo aixi approximation. *Journal of Artificial Intelligence Research*, 40:95–142, 2011.

Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 1113–1120, 2009.