Deep Bayesian Nonparametric Learning of Rules and Plans from Demonstrations with a Learned Automaton Prior

Brandon Araki,[1] **Kiran Vodrahalli,**[2] **Thomas Leech,**[1, 3]
Cristian-Ioan Vasile,[1] **Mark Donahue,**[3] **and Daniela Rus**[1]
[1]CSAIL, Massachusetts Institute of Technology
[2] Columbia University
[3] MIT Lincoln Laboratory
araki@mit.edu, knv2109@columbia.edu, tleech@mit.edu, cvasile@mit.edu, mark.donahue@ll.mit.edu, rus@csail.mit.edu

Abstract

We introduce a method to learn imitative policies from expert demonstrations that are *interpretable* and *manipulable*. We achieve interpretability by modeling the interactions between high-level actions as an automaton with connections to formal logic. We achieve manipulability by integrating this automaton into planning, so that changes to the automaton have predictable effects on the learned behavior. These qualities allow a human user to first understand what the model has learned, and then either correct the learned behavior or zero-shot generalize to new, similar tasks. We build upon previous work by no longer requiring additional supervised information which is hard to collect in practice. We achieve this by using a deep Bayesian nonparametric hierarchical model. We test our model on several domains and also show results for a real-world implementation on a mobile robotic arm platform.

1 Introduction

Imitation learning (IL) is a method for producing desired behaviors in agents with expert demonstrations [Abbeel and Ng (2004), Daumé III, Langford, and Marcu (2009), Ross, Gordon, and Bagnell (2011)]. IL is a successful approach in many tasks including camera control, speech imitation, and self-driving for cars [Taylor et al. (2017), Codevilla et al. (2018), Ho and Ermon (2016)]. Despite these successes, current approaches to IL fall short of human learning, particularly in the domain of multi-step tasks. When a human learns a complicated task, such as driving or preparing a meal, they can explain, at a high level, the rules they followed or the steps they took to perform the task. Furthermore, if the rules change or if they want to, say, cook a slightly different dish, they can easily alter their behavior to adapt to the new circumstances. Most current approaches to IL cannot achieve this level of flexibility because their learned policies are black boxes.

Our prior work, Araki et al. (2019b), introduces the notions of *interpretability* and *manipulability*. A policy is called *interpretable* if the relations between its high-level actions are defined by a finite state automaton (FSA), and it is *manipulable* if, by changing the transitions in the FSA, a human user can produce predictable changes in the learned behavior. These properties were achieved by designing a hierarchical model with high-level actions represented by an FSA and incorporating the FSA into a differentiable planning algorithm. However, in addition to expert trajectories, this model requires the trajectories to be labeled with the current automaton state at every time step, which is almost equivalent to requiring the user to know the automaton ahead of time.

In this paper, we remove the necessity of FSA state labels by interpreting the learning problem as solving a partially observable Markov decision process (POMDP) where the hidden states and transitions correspond to an unknown FSA. We model the FSA as part of a hidden Markov model (HMM), which represents the composition of a planning policy with the POMDP. Learning the HMM is challenging because the FSA state labels, transitions, and the number of FSA states are all unknown. We fit the HMM with stochastic variational inference (SVI) on expert demonstration data, simultaneously learning the unknown FSA and planning over the resulting MDP.

We also use spectral techniques applied to a matrix representation of an automaton to create a prior for the FSA. We report considerable improvements over baselines in several domains, including two robotic manipulation tasks with real-world experiments. Because our model satisfies the interpretability and manipulability properties from Araki et al. (2019b), we can zero-shot generalize to new tasks and fix mistakes in incorrect models without additional demonstrations.

1.1 Contributions

1. We solve logic-structured POMDPs by learning a planning policy from task demonstrations alone, extending ideas from the differentiable planning literature.
2. We use spectral techniques applied to a matrix representation of a finite automaton to design a novel prior for nonparametric hierarchical Bayesian models.
3. Our model learns the FSA transition matrix, allowing us to *interpret* the rules that the model has learned.
4. We explain how to modify the learned transition matrix to *manipulate* the behavior of the agent on a real-world robotic platform in the contexts of packing a lunchbox and opening a locked cabinet.

2 Related Work

Our model extends Araki et al. (2019b), which builds upon the Value Iteration Network (VIN) model (Tamar et al. 2016) by applying a more structured variant of VIN to the product of a low-level MDP with a logical specification defined by an FSA. Other works incorporating logical structure into the imitation learning setting include Paxton et al. (2017), Li, Ma, and Belta (2017), Hasanbeig, Abate, and Kroening (2018), Icarte et al. (2018), Burke, Penkov, and Ramamoorthy (2019), and Gordon, Fox, and Farhadi (2019). These models assume that at least part of the logic specification is known, and they are not interpretable and manipulable. By contrast, our model learns the FSA end-to-end in an unsupervised manner from demonstrations.

Shah et al. (2018) gives methods for learning a posterior over logic specifications from demonstrations, and Shah, Li, and Shah (2019) defines objectives for planning over a distribution of planning problems defined by logic specifications which reduces to the problem of solving MDPs. Our work directly considers the problem as a POMDP and introduces data-driven prior assumptions on the distribution of an FSA to mitigate the hardness of the problem. In addition, we never have to enumerate an exponentially-sized MDP. Furthermore, since our model is learned end-to-end, we recover an FSA specification that is tuned for the imitation policy that we fit, enhancing manipulability and interpretability simultaneously. Karkus, Hsu, and Lee (2017) also extends VIN to partially observable settings. Our work departs from theirs since we 1) factor the transition dynamics hierarchically into an interpretable and manipulable FSA, and 2) employ Bayesian nonparametric inference and a data-driven prior distribution on the FSA structure over the whole model, making our policy interpretable and manipulable.

Our model draws inspiration from many methods in the Bayesian machine learning literature, and can be formulated as a recurrent, autoregressive HMM. Several other works have studied hierarchical dynamical models in the nonparameteric Bayesian inference setting and have developed various techniques for inference [Fox et al. (2011a), Fox et al. (2011b), Chen, Linderman, and Wilson (2016), Johnson et al. (2016), Linderman et al. (2017)]. Our primary contribution to this literature is the way we use the structure of logical automata in the prior over the hidden states, making our model interpretable and manipulable.

Rhinehart, McAllister, and Levine (2018) also develop a deep Bayesian model for solving POMDPs and do not require new data to change the objective of the policy (although they require the policy to be re-optimized). Since we model the learned rules as an FSA, we can affect precise changes to the learned policy by adjusting the FSA, versus assigning probability distributions to goals or tweaking the learned policy by adjusting arbitrary costs to states.

3 Problem Formulation

Our key objective is to infer rules and policies from task demonstrations. The rules are represented as a probabilistic automaton $\mathcal{W} = (\mathcal{F}, \mathcal{P}, TM, \mathtt{I}, \mathtt{F})$, where $\mathcal{F}$ is the states of the automaton; $\mathcal{P}$ is the set of input symbols or propositions; $TM : \mathcal{F} \times \mathcal{P} \times \mathcal{F} \to [0, 1]$ defines the transition probabilities between states; $\mathtt{I}$ is a vector of initial state probabilities; and $\mathtt{F}$ is the final state vector. We take $F = |\mathcal{F}|$ and $P = |\mathcal{P}|$. We assume that $\mathtt{I}$ and $\mathtt{F}$ are deterministic – the initial state is always state 0, and the final state is always state $(F - 1)$.

We formulate the overall problem as a POMDP. The state space is $\mathcal{C} = \mathcal{S} \times \mathcal{P} \times \mathcal{F}$, where $\mathcal{S} = \mathcal{X} \times \mathcal{Y}$ is a 2D grid (examples are shown in Fig. 4); $\mathcal{F}$ is the set of states of the automaton; and $\mathcal{P}$ is the set of propositions (an illustration of this joint state space is in Araki et al. (2019a)). The action space $\mathcal{A} = \{N, E, S, W, NE, NW, SE, SW\}$ – the agent can travel to any neighboring cell. In addition to the automaton transition function TM, there is also a low-level transition function $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ and a "proposition map" $M : \mathcal{S} \times \mathcal{P} \times t \to \{0, 1\}$. We assume that every low-level state is associated with a single proposition that is true at that state; the proposition map defines this association. We allow M to vary with time, implying that propositions can change location over time; however, we consider only one domain that changes with time. Note that we overload all the transition functions so that given their inputs, they return the next state instead of a probability. The reward function is $\mathcal{R} : \mathcal{C} \times \mathcal{A} \to \mathbb{R}$; we assume that the reward function does not vary with $\mathcal{P}$, $\mathcal{S}$, or $\mathcal{A}$, so $\mathcal{R} : \mathcal{F} \to \mathbb{R}$. The observation space is $\Omega = \mathcal{S} \times \mathcal{P}$, and the observation probability function is $\mathcal{O} : \mathcal{C} \times \mathcal{A} \times \Omega \to [0, 1]$. Therefore we have a POMDP described by the tuple $(\mathcal{C}, \mathcal{A}, T \times M \times TM, \mathcal{R}, \Omega, \mathcal{O}, \gamma_d)$.

With this notation, we formulate the POMDP learning problem as follows. Given N data points, data point i has T_i time steps. Dataset $\mathcal{D} = \langle d_0, \ldots, d_N \rangle$, where $d_i = \langle (s^i_0, a^i_0), \ldots, (s^i_{T_i}, a^i_{T_i}) \rangle$. Expanding the POMDP tuple gives $(\mathcal{S} \times \mathcal{P} \times \underline{\mathcal{F}}, \mathcal{A}, T \times M \times \underline{TM}, \underline{\mathcal{R}}, \mathcal{S} \times \mathcal{P}, \mathcal{O}, \gamma_d)$. Unknown elements have been underlined to emphasize the learning objective. We assume that the actions $\mathcal{A}$, the low-level transitions T, the proposition map M, the observation probabilities $\mathcal{O}$, and the discount factor γ_d are known. We also assume that $\mathcal{O}$ is deterministic – in other words, that the agent has sensors that can perfectly sense its state in the environment. The goal of solving a POMDP is to find a policy that maximizes reward – in this case, a policy that mimics the expert demonstrations. This is achieved by learning the automaton transition function TM, the reward function $\mathcal{R}$, and the number of states of the automaton F.

4 Method

To learn a policy for the POMDP, we formulate the POMDP composed with the policy as a hierarchical Hidden Markov Model (HMM) with recurrent transitions and autoregressive emissions, where the hidden states and transitions of the POMDP are latent variable parameters (shown in Fig. 2). The unobserved logic state at every time step is interpreted as a hidden state of the FSA, and TM, $\mathcal{R}$, and F are interpreted as high-level latent variables.

A sketch of the learning algorithm is shown in Alg. 1, and Fig. 1 shows example inputs and outputs. The data consist of a set of expert trajectories over a domain. The domain shown is a 3×3 gridworld where the agent must first go to a, then to b, while avoiding obstacles o. e is the "empty"

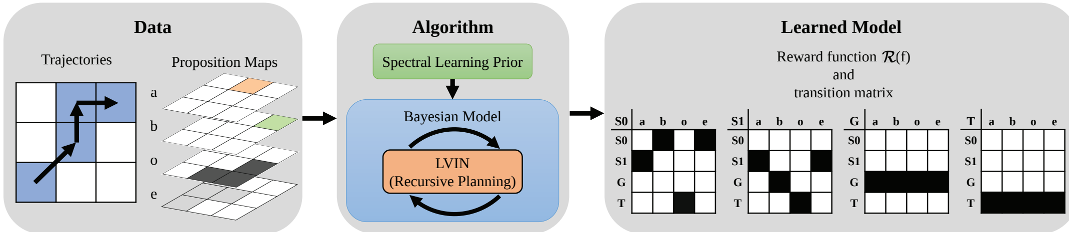


Figure 1: The data consist of a set of expert trajectories over a domain. Each trajectory has an associated proposition map that relates every position in the domain to a proposition. Spectral learning generates an automaton prior for the Bayesian model, which learns variational parameters that are equivalent to the reward function and transition matrix of the environment.

Algorithm 1 Maximum A Posteriori Estimation on Bayesian LVIN model

procedure MAP-ESTIMATION
  Training Inputs: $\{\{(s_t, a_t)^{(n)}\}_{t=0}^{T_i}\}_{i=0}^{N-1}$
  To learn:
    Number of automaton states prior $\widehat{\alpha} \in \mathbb{R}_{>0}^{2P-3}$
    Transition matrix prior $\widehat{\beta}^F \in \mathbb{R}_{>0}^{F \times P \times F}$ for $F = 4, \ldots, 2P$
    Reward function prior $\widehat{\gamma}^F \in \mathbb{R}^F \times \mathbb{R}_{>0}^F$ for $F = 4, \ldots, 2P$
  Generate spectral learning prior (Sec. 4.3)
  Stochastic variational inference on the learning objective $\mathcal{L}$ (Sec. 4.1)
  return $\widehat{\alpha}^*, \widehat{\overline{\beta}}^*, \widehat{\overline{\gamma}}^* = \arg\min_{(\widehat{\alpha}, \widehat{\overline{\beta}}, \widehat{\overline{\gamma}})} \mathcal{L}$
end procedure

proposition. The figure also illustrates the proposition map M, which maps every location to a proposition. Locations where a proposition is true are highlighted with a color. Note that only one proposition is true at any given location, and that each instance of an environment has a unique M.

The algorithm consists of approximating the posterior of a Bayesian model and returning the modes of the latent variable approximations (Sec. 4.1); LVIN, a differentiable variation of value iteration that incorporates both the low- and high-level transitions, plays an important role (Sec. 4.2). A common issue with learning complex Bayesian models is that they are very sensitive to their initialization. The discrete high-level transition function TM is particularly vulnerable to converging to local minima, so we use spectral learning to obtain a good prior (Sec. 4.3).

Posterior inference finds likely values for the number of automaton states F, the reward function $\mathcal{R}$, and the transition matrix TM. In the figure, the TM is represented as a collection of matrices - each matrix is associated with the current logic state; columns correspond to propositions and rows correspond to the next logic state. The entry in each grid is the probability $TM(s'|s, p)$. Black indicates 1 and white indicates 0. Therefore in the initial state $S0$, a causes a transition to $S1$, whereas o causes a transition to the trap state T. The outputs of the algorithm are valuable for two reasons: 1) TM is relatively easy to interpret, giving insight into the rules that the expert is following; and 2) $\mathcal{R}$ and TM can be used for planning. Furthermore, modifications to TM result in predictable changes in the agent's behavior.

4.1 Bayesian Model

We now define the Bayesian model for the policy class (all variables in this section are listed in Araki et al. (2019a)). The nonparametric model is stated in Araki et al. (2019a); in practice, we approximate the nonparametric model by assuming the possible number of FSA states is finite. We will discuss this approximate model for the rest of the paper.

One of the main challenges in this learning problem is to infer the number of states F of the transition matrix – this problem is nonparametric because F is theoretically unbounded. We approximate the nonparametric nature of the problem by assuming that F is upper-bounded by $2P$. This upper bound implies that each proposition is not responsible for more than 2 transitions to distinct FSA states, which we believe is a reasonable assumption for normal domains. TM and $\mathcal{R}$ both rely on F to determine their dimensionality. Hidden-state transitions are a function of TM and the previous observation, among other things (see Fig. 2). Transitions that depend on variables besides the previous hidden state are called recurrent. Transitions between low-level states s_{t-1} and s_t are determined by $T(s_t|a_t, s_{t-1})$. Actions are chosen by a policy found using value iteration. Our value iteration module incorporates TM and is called LVIN (Sec. 4.2). Both T and LVIN depend on variables besides the current hidden state and are therefore called autoregressive.

The full generative model is described below:

$$\begin{aligned}
\alpha \in \mathbb{R}_{>0}^{2P-3},\ \overline{\beta} \in (\beta^4, \ldots, \beta^{2P}),\ \overline{\gamma} \in (\gamma^4, \ldots, \gamma^{2P}) \\
\theta &\sim \text{Dirichlet}(\alpha) \\
F &\sim \text{Categorical}(\theta) \\
\beta^F \in \mathbb{R}_{>0}^{F \times P \times F},\ \gamma^F \in \mathbb{R}^F \times \mathbb{R}_{>0}^F \\
TM^F|\beta^F &\sim \text{Dirichlet}(\beta^F) \\
\mathcal{R}^F|\gamma^F &\sim \text{Normal}(\gamma^F) \\
\pi &:= \text{LVIN}(TM^F, \mathcal{R}^F) \\
a_t|s_{t-1}, f_{t-1} &\sim \pi(s_{t-1}, f_{t-1}) \\
s_t := T(a_t, s_{t-1}),\ p_t &:= M(s_t) \\
f_t|p_t, f_{t-1}, TM^F &\sim \text{Categorical}(TM^F(f_{t-1}, p_t))
\end{aligned}$$

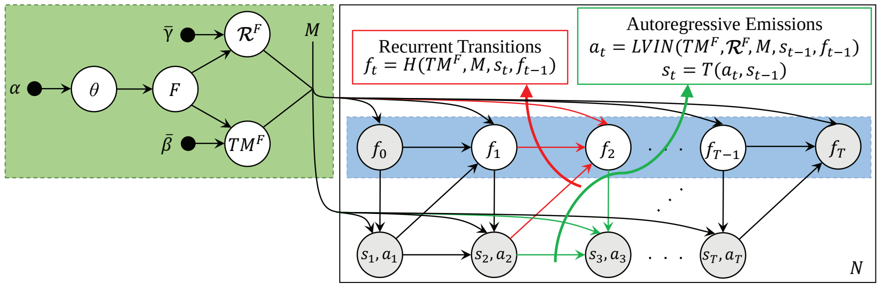


Figure 2: The graphical model. The model is a nonparametric, hierarchical (green box), recurrent autoregressive HMM. White circles are latent variables; grey circles are observed variables; and black dots are priors. M is included as an input to the model to emphasize that every instance of the environment has a unique proposition map. The plate around the HMM indicates that those variables are repeated for N trajectories.

α is the prior over distributions θ of potential numbers of FSA states F. $\overline{\beta}$ is a collection of $2P-3$ priors β^F, where β^F is the prior of the transition matrix TM^F of dimensionality $F \times P \times F$. Similarly, $\overline{\gamma}$ is a collection of $2P-3$ priors γ^F, where γ^F represents the mean and variance priors for the reward function $\mathcal{R}^F$. π is the policy found using LVIN; action a_t is drawn from the policy. State s_t and proposition p_t are given by the deterministic functions T and M. Lastly, the current automaton state f_t is drawn from TM^F.

We incorporate known features of the environment into the model as priors. Many of these priors rely on the assumption that every automaton we consider has one initial state, one goal state, and one trap state. Our assumptions about the rules of the environment are built into each β^F, which are the Dirichlet priors for TM^F. Each β^F is populated with the value 0.5 before adding other values, since for Dirichlet priors, values below 1 encourage peaked/discrete distributions. Therefore this prior biases the TM towards deterministic automata. We add a prior to the trap state that favors self-transitions because we know that the trap state is a "dead-end" state. We add an obstacle prior to bias the model in favor of automata where obstacles lead to the trap state. We add a goal state prior so that the model favors self-transitions for the goal state. We add an empty state prior so that the model favors self-transitions for all states given the empty proposition. We use spectral learning to give a prior for the other transitions as well as for α (Sec. 4.3). We also give priors to the reward function so that the goal state has a positive reward and the trap state has a negative reward.

The joint distribution over the latent variables is shown below, with possible numbers of FSA states F and possible next states f marginalized out. A bar over a variable indicates that it is a list over possible values of F. i represents the data index; t the time index; and F the number-of-automaton-states index.

$$p(\mathcal{D}, \overline{\mathcal{R}}, \overline{TM}, \theta | \alpha, \overline{\beta}, \overline{\gamma}) = \prod_{i=0}^{N-1} \prod_{t=2}^{T^i} \sum_{F=4}^{2P} \sum_{f_{t-1}^{i,F}=0}^{F-1} p(s_t^i | a_t^i, s_{t-1}^i)$$
$$p(a_t^i | s_{t-1}^i, f_{t-1}^{i,F}, TM^F, \mathcal{R}^F) p(f_{t-1}^{i,F} | f_{t-2}^{i,F}, TM^F, s_{t-1}^i)$$
$$p(TM^F | F, \beta^F) p(\mathcal{R}^F | F, \gamma^F) p(F|\theta) p(\theta|\alpha)$$

The posterior can be derived from the joint distribution. Our variational approximation to the posterior is

$$q(\overline{\mathcal{R}}, \overline{TM}, \theta | \mathcal{D}, \widehat{\alpha}, \widehat{\overline{\beta}}, \widehat{\overline{\gamma}}) = \prod_{i=0}^{N-1} \prod_{t=2}^{T^i} \sum_{F=4}^{2P} \sum_{f_{t-1}^{i,F}=0}^{F-1}$$
$$q(f_{t-1}^{i,F} | f_{t-2}^{i,F}, TM^F, s_{t-1}^i) q(\mathcal{R}^F | F, \widehat{\gamma}^F)$$
$$q(TM^F | F, \widehat{\beta}^F) q(F|\theta) q(\theta | \widehat{\alpha})$$

The variational approximation uses amortization (Ritchie, Horsfall, and Goodman 2016) to avoid having a huge number of variational parameters – without amortization, every next FSA state $f_{t-1}^{i,F}$ would have to be drawn from an independent Dirichlet distribution with its own set of variational parameters. We avoid this situation by drawing $f_{t-1}^{i,F}$ from $TM^F(f_{t-2}^{i,F}, M(s_{t-1}^i))$, so that parameters are shared for a given FSA state-proposition pair. Also note that the variational approximation is the same as the joint likelihood, except that the first two data-dependent distributions are removed. Other works such as Damianou and Lawrence (2013) have also used this pattern for constructing variational approximations. The proposal distributions are the same as the corresponding distributions in the joint likelihood function. An illustration of the graphical model of the variational approximation is in Araki et al. (2019a).

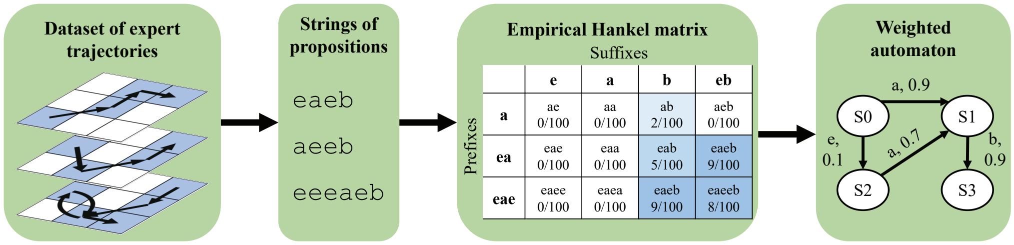
The objective of the variational inference problem is to minimize the KL divergence between the true posterior and the variational distribution:

$$\mathcal{L} = \mathrm{KL}(q(\overline{\mathcal{R}}, \overline{TM}, \theta | \mathcal{D}, \widehat{\alpha}, \widehat{\overline{\beta}}, \widehat{\overline{\gamma}}) || p(\overline{\mathcal{R}}, \overline{TM}, \theta | \mathcal{D}, \alpha, \overline{\beta}, \overline{\gamma}))$$

$$\widehat{\alpha}^*, \widehat{\overline{\beta}}^*, \widehat{\overline{\gamma}}^* = \underset{(\widehat{\alpha}, \widehat{\overline{\beta}}, \widehat{\overline{\gamma}})}{\arg\min} \mathcal{L}$$

$\widehat{\alpha}^*$, $\widehat{\overline{\beta}}^*$, and $\widehat{\overline{\gamma}}^*$ serve as approximations of α, $\overline{\beta}$, and $\overline{\gamma}$, and therefore define distributions over F, $\overline{TM}$, and $\overline{\mathcal{R}}$. Letting $F^* = \arg\max_F \widehat{\alpha}^*$, we get priors for TM^{F^*} ($\widehat{\beta}^{F^*}$) and $\mathcal{R}^{F^*}$ ($\widehat{\gamma}^{F^*}$) which can be used for planning.

The variational problem was implemented using Pyro and Pytorch. Pyro uses stochastic variational inference to approximate the variational parameters.



	e	a	b	eb
a	ae 0/100	aa 0/100	ab 2/100	aeb 0/100
ea	eae 0/100	eaa 0/100	eab 5/100	eaeb 9/100
eae	eaee 0/100	eaea 0/100	eaeb 9/100	eaeeb 8/100

Figure 3: The process of applying spectral learning to our data. A dataset of expert trajectories is converted to a list of input strings. The frequency of each input string is used to create the empirical Hankel matrix (only a portion of the matrix is depicted). From the rank decomposition of the matrix we can derive a weighted automaton as a prior for the Bayesian model.

4.2 Logical Value Iteration Networks (LVIN)

At every time step, the model must choose an optimal action by calculating a policy. In this work, the policy is found using value iteration on the learned MDP. We use the "Logical Value Iteration Network" (Araki et al. 2019b), which integrates the high-level transitions TM into value iteration. The modified value iteration equations are shown below. In the first step, the Q-function Q is updated using reward function R, low-level transitions T and value function V. Next, the value function is updated. LVIN adds a third step, where the values are propagated between logic states using TM. Note that propositions $\mathcal{P}$ are not an input to the Q and value functions because each low-level state s is deterministically associated with a single proposition p, so p is a redundant input for a given s.

$$Q^{t+1}(s,f,a) \leftarrow R(s,f,a) + \gamma_d \sum_{s' \in \mathbb{S}} T(s'|s,a)V^t(s',f)$$

$$\widehat{V}^{t+1}(s,f) \leftarrow \max_a Q^{t+1}(s,f,a)$$

$$V^{t+1}(s,f) \leftarrow \sum_{f' \in \mathbb{F}} TM(f'|f,M(s))\widehat{V}^t(s,f')$$

4.3 Spectral Learning for Weighted Automata

One of the main issues of using variational inference on complex Bayesian models is its tendency to converge to undesirable local minima. To avoid this, we use the output of spectral learning for weighted automata as a prior for TM.

Spectral learning uses tensor decomposition to efficiently learn latent variables. We can use this technique by representing an automaton as a Hankel matrix (Arrivault et al. 2017). The Hankel matrix is a bi-infinite matrix with rows that correspond to prefixes and columns to suffixes of all possible input strings. The value of a cell is the probability of the corresponding string. The input string corresponds to the propositions that are true at each time step. A string from the environment in Fig. 1 could be `eaeeeb`, indicating that the agent traversed an empty space before reaching a, and then traversed another three empty spaces before reaching b. The rank of the Hankel matrix is equal to the automaton's number of states. An automaton with m states can be reconstructed from a rank m factorization of the Hankel matrix.

The problem is: given a rank m, find a rank factorization $H = WP$. H is the Hankel matrix. $H \in \mathbb{R}^{n\times d}$, $W \in \mathbb{R}^{n\times m}$, $P \in \mathbb{R}^{m\times d}$. Let $h_{\epsilon,S} = H[0,:]$ and $h_{P,\epsilon} = H[:,0]$. $H_\sigma = p(u\sigma v)$ – in other words, H_σ is a submatrix of H where all prefixes end with σ. Let Σ be an alphabet with elements σ (Σ corresponds to the propositions). H can be divided into $|\Sigma|$ submatrices H_σ and a submatrix H_ϵ (where ϵ corresponds to an empty string).

When W and P are obtained, we can derive the Weighted Automaton (WA) $\mathcal{W} = \langle m, \mathtt{I}, \mathtt{F}, (M_\sigma)_{\sigma\in\Sigma}\rangle$ corresponding to the Hankel matrix. m is the number of states, equal to the rank of the Hankel matrix. $\mathtt{I}$ is the $m \times 1$ vector of initial state probabilities, and $\mathtt{F}$ is the $m \times 1$ vector of final state probabilities. M_σ are the $m \times m$ transition weights from the current state to the next state for each proposition σ. $\mathcal{W}$ can be derived using the following equations. ($^+$ stands for the Moore-Penrose pseudoinverse).

$$\mathtt{I}^\top = h_{\epsilon,S}^\top P^+, \quad \mathtt{F} = W^+ h_{P,\epsilon}, \quad M_\sigma = W^+ H_\sigma P^+$$

Let $\mathbb{1}_I = [1, 0, \ldots, 0]$ and $\mathbb{1}_F = [0, \ldots, 0, 1]$. W and P are obtained from the optimization problem

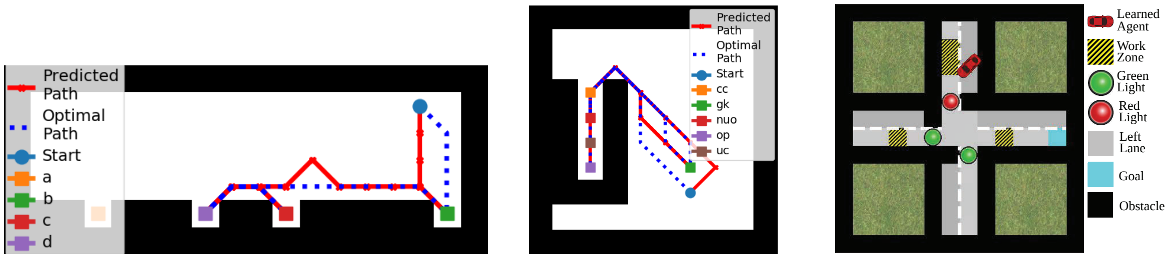
$$\begin{aligned} \underset{W,P}{\text{minimize}} \quad & \frac{1}{2}||H - WP||_F^2 + \alpha_s \sum_{\sigma\in\Sigma} ||W^+ H_\sigma P^+||_1 \\ & + \beta_s ||h_{\epsilon,S}^\top P^+ - \mathbb{1}_I||_1 + \gamma_s ||W^+ h_{P,\epsilon} - \mathbb{1}_F||_1 \\ \text{subject to} \quad & W \geq 0, P \geq 0 \end{aligned}$$

α_s, β_s, and γ_s are hyperparameter weights. The first term in the objective function corresponds to the matrix factorization; the second term is an L_1 regularization term on the transition weight matrices, and the third and fourth terms are constraints for the first and last states to be the initial and final states, respectively. The positivity constraints on W and P constrain the weights to be positive.

We use the open-source Sp2Learn toolbox (Arrivault et al. 2017) to process the data and generate the Hankel matrices. We use Tensorflow to perform the optimization problem above to factor the matrix. We can then obtain $\mathtt{I}$, $\mathtt{F}$, and M_σ.

The transition weights of the learned WA are not constrained to add to one, so they do not correspond to probabilities. The WA will also not include propositions that are not present in the data strings (such as the obstacle proposition). Therefore the learned WA is better suited as a prior rather than as the primary means of determining TM.

We learn automata with number of states ranging from 4 to $2P$. We have observed that for every domain tested, the optimization loss drops by one or two orders of magnitude



(a) Lunchbox domain (b) Cabinet domain (c) Driving domain

Figure 4: Example instances of three domains

when the number of states reaches the correct number. We therefore use the optimization losses to create a prior on the number of states (defined as α in Sec. 4.1). If the optimization loss for a certain number of states F is c_F, then the prior for F states is $-\log(c_F/c_{F-1})$. We also use the transition weights as prior values for $\overline{\beta}$ in the Bayesian model.

5 Experiments & Results

5.1 Generating Expert Data

Linear Temporal Logic We use linear temporal logic (LTL) to formally specify tasks (Clarke, Grumberg, and Peled 2001). Formulae ϕ have the syntax grammar

$$\phi := p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \bigcirc \phi \mid \phi_1 \,\mathcal{U}\, \phi_2$$

where p is a *proposition* (a boolean-valued truth statement that can correspond to objects or goals in the world), $\neg$ is negation, $\vee$ is disjunction, $\bigcirc$ is "next", and $\mathcal{U}$ is "until". The derived rules are conjunction ($\wedge$), implication ($\implies$), equivalence ($\leftrightarrow$), "eventually" ($\Diamond\phi \equiv \texttt{True}\,\mathcal{U}\,\phi$) and "always" ($\Box\phi \equiv \neg\Diamond\neg\phi$) (Baier and Katoen 2008). $\phi_1 \,\mathcal{U}\, \phi_2$ means that ϕ_1 is true until ϕ_2 is true, $\Diamond\phi$ means that there is a time where ϕ is true and $\Box\phi$ means that ϕ is always true.

Generating Data We use SPOT (Duret-Lutz et al. 2016) and Lomap (Ulusoy et al. 2013) to convert LTL formulae into FSAs. Every FSA that we consider has a goal state G, which is the desired final state of the agent, and a trap state T, which is an undesired terminal state. We generate a set of environments in which obstacles and other propositions are randomly placed. Given the FSA and an environment, we run Dijkstra's shortest path algorithm to create expert trajectories that we use as data for imitation learning.

LSTM Baseline: We compare the performance of LVIN to an LSTM network, which we take to be a generic method for dealing with time-series data. The first layer of the network is a 3D CNN with 1024 channels. The second layer is an LSTM with 1024 hidden units. The hidden units do not directly correspond to logic states or a TM.

5.2 Environments

Lunchbox Domain The lunchbox domain (Fig. 4a) is an 18×7 gridworld where the agent must first pick up either a sandwich a or a burger b and put it in a lunchbox d, and then pick up a banana c and put it in the lunchbox d. The specification is $\Diamond((a \vee b) \wedge \Diamond(d \wedge \Diamond(c \wedge \Diamond d))) \wedge \Box\neg o$.

	Training		Test	
	LVIN	LSTM	LVIN	LSTM
Lunchbox				
Set size	500	9000	1800	1800
Success Rate	99.60%	91.44%	**99.94%**	82.44%
Cabinet				
Set size	550	6000	1800	1800
Success Rate	100.00%	93.60%	**100.0%**	89.58%
Driving				
Set size	500	6000	1800	1800
Success Rate	100.0%	58.54%	**100.0%**	58.60%

Table 1: Training and test performance of LVIN vs. LSTM

Cabinet Domain The cabinet domain is a 10×10 gridworld where the agent must open a cabinet. First it must check if the cabinet is locked (cc). If the cabinet is locked (lo), the agent must get the key (gk), unlock the cabinet (uc), and open it (op). If the cabinet is unlocked (uo), then the agent can open it (op). The specification is $\Diamond(cc \wedge \Diamond((uo \wedge \Diamond op) \vee (lo \wedge (\Diamond(gk \wedge \Diamond(uc \wedge \Diamond op)))))) \wedge \Box\neg o$. Because many of the propositions lie in nearly the same point in space (e.g. checking the cabinet, observing that it is unlocked, and opening the cabinet), we define a "well" (as shown in Fig. 4b) that contains the relevant propositions in separate grid spaces but represents a single point in space in the real world.

Driving Domain The driving domain (Fig. 4c) is a 14×14 gridworld where the agent must obey three "rules of the road" – prefer the right lane over the left lane (l): $\Box\Diamond\neg l$; stop at red lights (r) until they turn green (h): $\Box(r \Rightarrow (r \;\mathcal{U}\; h))$; and reach the goal ($g$) while avoiding obstacles (o): $\Diamond g \wedge \Box\neg o$. Unlike the other domains, this domain has a time-varying element (the red lights turn green); it also has an extra action – "do not move" – since the car must sometimes wait at the red light.

Performance We ran experiments using an Intel i9 processor and an Nvidia 1080Ti GPU. The simplest environment takes ~ 1.4 hours to train; the most complicated takes ~ 7.5 days to train. The KL divergence for all environments shows a typical training pattern in which the divergence rapidly decreases before flattening out. Runtimes and loss curves for all environments can be found in Araki et al. (2019a).

Performance of LVIN (shorthand for our model) vs. the LSTM network is shown in Table 1. We measure "success rate" as the proportion of trajectories where the agent satisfies the environment's specification. LVIN achieves virtually perfect performance on every domain with relatively little data. The LSTM network achieves fairly high performance on the lunchbox and cabinet domains, but has poor performance in the time-varying driving domain. On top of achieving better performance than the LSTM network, the LVIN model also has an interpretable output that can be modified to change the learned policy.

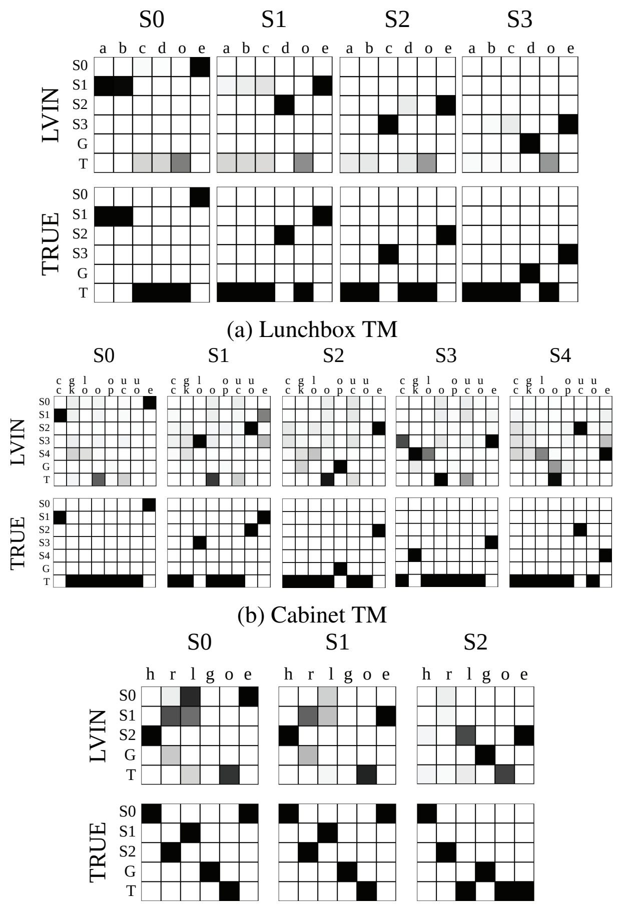
The LVIN model requires much less data than the LSTM network for two reasons. One is that the LVIN model can take advantage of the spectral learning prior to reduce the amount of data needed to converge to a solution, whereas the LSTM network cannot use the prior. The second is that since the LVIN model is model-based, once it learns an accurate model of the rules it can generalize to unseen permutations of the environment better than the LSTM network, which in a sense only interpolates between data points.

5.3 Interpretability

Our method learns an interpretable model of the rules of an environment in the form of a transition matrix (TM). Learned vs. true TMs are shown in Fig. 5 (we leave out the goal and trap states of the TMs, since they are trivial). The plots show values of the learned variational parameter $\widehat{\beta}^{F^*}$. Therefore the plots do not show the values of the actual TM but rather the values of the prior of the TM, giving an idea of how "certain" the model is of each transition.

Fig. 5a shows the TM of the lunchbox domain. Each matrix corresponds to the transitions associated with a given automaton state. Columns correspond to propositions (e is the empty proposition) and rows correspond to the probability that, given current state f and proposition p, the next state is f'. Inspection of the learned TM shows that in the initial state $S0$, picking up the sandwich or burger (a or b) leads to state $S1$. In $S1$, putting the sandwich/burger into the lunchbox (d) leads to $S2$. In $S2$, picking up the banana c leads to $S3$, and in $S3$, putting the banana in the lunchbox d leads to the goal state G.

The other domains can be interpreted similarly, and most of them closely match their expected TMs (see Araki et al. (2019a) for more examples). One exception is the driving domain (Fig. 5c). In the expected TM, the initial state $S0$ transitions to a lower-reward state $S1$ when the car enters the left lane (indicating that the left lane is allowed but unideal); $S0$ transitions to $S2$ when the car is at a red light, and then back to $S0$ when the green light h turns on. Our model learns a different TM – but due to the nature of the TM, it can still be interpreted. Unlike in the "true" TM, in the learned TM, the green light acts as a switch – the agent cannot reach the goal state unless it has encountered the green light. This is an artifact of the domain, since the agent always passes a green light before reaching the goal. The red light leads from $S0$ to $S1$, which is a lower-reward duplicate of $S0$. The agent will wait for the red light to turn green because it thinks it must encounter a green light before it can reach the goal. Regarding the left lane, the TM places significant weight on a transition to low-reward $S1$ when in $S0$, which discourages the agent from entering the left lane. Therefore although not as tidy as the true TM, the learned TM is still interpretable.



(a) Lunchbox TM

(b) Cabinet TM

(c) Driving TM

Figure 5: Learned vs true TMs for three domains

5.4 Manipulability Experiments on Jaco Arm

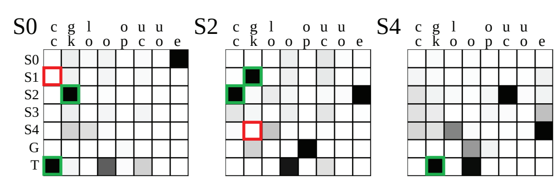


Figure 6: Cabinet TM modifications. Red indicates that a transition has been deleted; green that one has been added.

Our method allows the learned policy to be manipulated to produce reliable new behaviors. We demonstrate this ability on a real-word platform, a Jaco arm. The Jaco arm is a 6-DOF arm with a 3-fingered hand and a mobile base. An Optitrack motion capture system was used to track the hand and manipulated objects. The system was implemented using ROS (Quigley et al. 2009). The Open Motion Planning

	Lunchbox				Cabinet	
	ϕ_{l1}	ϕ_{l2}	ϕ_{l3}	ϕ_{l4}	ϕ_{c1}	ϕ_{c2}
Successes out of 20	20	20	19	19	20	17

Table 2: Performance of Jaco robot in executing learned lunchbox and cabinet tasks

Library (Şucan, Moll, and Kavraki 2012) was used for motion planning. The motion capture system was used to translate the positions of the hand and objects into a 2D grid, and an LVIN model trained on simulated data was used to generate a path satisfying the specifications.

We modified the learned TMs of the lunchbox and cabinet domains. We call the original lunchbox specification ϕ_{l1}. We tested three modified specifications – pick up the sandwich first, then the banana (ϕ_{l2}); pick up the burger first, then the banana (ϕ_{l3}); and pick up the banana, then either the sandwich or the burger (ϕ_{l4}). These experiments are analogous to the ones in Araki et al. (2019b) and are meant to show that though significantly less information was given to our model in the learning process, it can still perform just as well.

We also modified the learned cabinet TM (ϕ_{c1}) – if we know that the cabinet is locked, we have the agent pick up the key first before checking the cabinet (ϕ_{c2}). The modifications to the TM are shown in Fig. 6. The TM must be modified so that the agent will get the key (gk) before checking the cabinet (cc). So in the initial state $S0$, cc is set to go to the trap state so that the agent will avoid it; gk is set to transition to $S2$, indicating to the agent that it should get the key first. In $S2$, we then modify the TM so that gk is no longer the goal but rather cc is – in other words, the agent will then head to the cabinet and check it. Finally, in $S4$, once the agent has checked the cabinet, it must unlock the cabinet and it does not need to get the key, so we set gk to the trap state so the agent will be certain to unlock the cabinet and not try to get the key. These modifications, as shown in our experiments, successfully change the behavior of the agent to always get the key before checking the cabinet.

Each specification was tested 20 times on our experimental platform; as shown in Table 2 there were only a few failures, and these were all due to mechanical failures of the Jaco arm, such as the manipulator dropping an object or losing its grasp on the cabinet key.

6 Conclusion

This work addresses a challenge in the field of imitation learning, which is how to learn a model of the expert's behavior that is interpretable and manipulable. Our solution is to represent the rules of the environment as a high-level automaton; by using a nonparametric Bayesian model to learn the automaton, we can infer the structure of the automaton from expert trajectories without being given any information about the automaton. We demonstrate the effectiveness of our technique on several domains, showing how the learned transition matrices can be interpreted and manipulated to produce predictable new behaviors.

Acknowledgments

We are grateful to the many people who gave us invaluable advice on this project, including Leslie Kaelbling, Scott Linderman, Keyon Vafa, and David Blei. This material is supported by NSF grant 1723943, ONR N00014-18-1-2830, and the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001. K.V. is supported by an NSF Graduate Research Fellowship. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

References

Abbeel, P., and Ng, A. Y. 2004. Apprenticeship learning via inverse reinforcement learning. *ICML '04 International Conference on Machine Learning*.

Araki, B.; Vodrahalli, K.; Leech, T.; Vasile, C.-I.; Donahue, M.; and Rus, D. 2019a. Additional investigations of deep bayesian nonparametric learning of rules and plans from demonstrations with a learned automaton prior. Available at https://kiranvodrahalli.github.io/research/publications/aaai20_supp.pdf.

Araki, B.; Vodrahalli, K.; Leech, T.; Vasile, C. I.; Donahue, T.; and Rus, D. 2019b. Learning to plan with logical automata. *Robotics: Science and Systems 2019*.

Arrivault, D.; Benielli, D.; Denis, F.; and Eyraud, R. 2017. Sp2learn: A toolbox for the spectral learning of weighted automata. In *International Conference on Grammatical Inference*, 105–119.

Baier, C., and Katoen, J. 2008. *Principles of model checking*. MIT Press.

Burke, M.; Penkov, S.; and Ramamoorthy, S. 2019. From explanation to synthesis: Compositional program induction for learning from demonstration. *arXiv:1902.10657*.

Chen, Z.; Linderman, S. W.; and Wilson, M. A. 2016. Bayesian nonparametric methods for discovering latent structures of rat hippocampal ensemble spikes. In *2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP)*, 1–6. IEEE.

Clarke, E. M.; Grumberg, O.; and Peled, D. 2001. *Model Checking*. MIT Press.

Codevilla, F.; Miiller, M.; López, A.; Koltun, V.; and Dosovitskiy, A. 2018. End-to-end driving via conditional imitation learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 1–9. IEEE.

Damianou, A., and Lawrence, N. 2013. Deep gaussian processes. In *Artificial Intelligence and Statistics*, 207–215.

Daumé III, H.; Langford, J.; and Marcu, D. 2009. Search-based structured prediction. *Journal of Machine Learning* 75:297–325.

Duret-Lutz, A.; Lewkowicz, A.; Fauchille, A.; Michaud, T.; Renault, E.; and Xu, L. 2016. Spot 2.0 — a framework for LTL and ω-automata manipulation. In *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA'16)*. Springer.

Fox, E.; Sudderth, E. B.; Jordan, M. I.; and Willsky, A. S. 2011a. Bayesian nonparametric inference of switching dynamic linear models. *IEEE Transactions on Signal Processing* 59(4):1569–1585.

Fox, E. B.; Sudderth, E. B.; Jordan, M. I.; and Willsky, A. S. 2011b. A sticky HDP-HMM with application to speaker diarization. *Annals of Applied Statistics* 5(2 A):1020–1056.

Gordon, D.; Fox, D.; and Farhadi, A. 2019. What Should I Do Now? Marrying Reinforcement Learning and Symbolic Planning. *arXiv:1901.01492*.

Hasanbeig, M.; Abate, A.; and Kroening, D. 2018. Logically-correct reinforcement learning. *arXiv preprint arXiv:1801.08099*.

Ho, J., and Ermon, S. 2016. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, 4565–4573.

Icarte, R. T.; Klassen, T. Q.; Valenzano, R.; and McIlraith, S. A. 2018. Teaching multiple tasks to an rl agent using ltl. *International Conference on Autonomous Agents and Multiagent Systems*.

Johnson, M. J.; Duvenaud, D.; Wiltschko, A. B.; Datta, S. R.; and Adams, R. P. 2016. Composing graphical models with neural networks for structured representations and fast inference. *Advances in neural information processing systems* 29:2946–2954.

Karkus, P.; Hsu, D.; and Lee, W. S. 2017. Qmdp-net: Deep learning for planning under partial observability. In *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc. 4694–4704.

Li, X.; Ma, Y.; and Belta, C. 2017. Automata guided hierarchical reinforcement learning for zero-shot skill composition. *arXiv:1711.00129*.

Linderman, S. W.; Johnson, M. J.; Miller, A. C.; Adams, R. P.; Blei, D. M.; and Paninski, L. 2017. Bayesian Learning and Inference in Recurrent Switching Linear Dynamical Systems. *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics* 54:914–922.

Paxton, C.; Raman, V.; Hager, G. D.; and Kobilarov, M. 2017. Combining neural networks and tree search for task and motion planning in challenging environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 6059–6066.

Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; and Ng, A. Y. 2009. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, 5. Kobe, Japan.

Rhinehart, N.; McAllister, R.; and Levine, S. 2018. Deep Imitative Models for Flexible Inference, Planning, and Control. *arXiv:1810.06544*.

Ritchie, D.; Horsfall, P.; and Goodman, N. D. 2016. Deep amortized inference for probabilistic programs. *arXiv preprint arXiv:1610.05735*.

Ross, S.; Gordon, G.; and Bagnell, J. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* 15:627–635.

Shah, A.; Kamath, P.; Li, S.; and Shah, J. 2018. Bayesian Inference of Temporal Task Specifications from Demonstrations. *Neural Information Processing Systems (NIPS)* 411–420.

Shah, A.; Li, S.; and Shah, J. 2019. Planning with uncertain specifications. *arXiv:1906.03218*.

Şucan, I. A.; Moll, M.; and Kavraki, L. E. 2012. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine* 19(4):72–82. http://ompl.kavrakilab.org.

Tamar, A.; Wu, Y.; Thomas, G.; Levine, S.; and Abbeel, P. 2016. Value iteration networks. In *Advances in Neural Information Processing Systems 29*, 2154–2162.

Taylor, S.; Kim, T.; Yue, Y.; Mahler, M.; Krahe, J.; Rodriguez, A. G.; Hodgins, J.; and Matthews, I. 2017. A deep learning approach for generalized speech animation. *ACM Transactions on Graphics (TOG)* 36(4):93.

Ulusoy, A.; Smith, S. L.; Ding, X. C.; Belta, C.; and Rus, D. 2013. Optimality and robustness in multi-robot path planning with temporal logic constraints. *The International Journal of Robotics Research* 32(8):889–911.