Incremental Task Model Updates from Demonstration

Reymundo A. Gutierrez
University of Texas at Austin
ragtz@cs.utexas.edu

Vivian Chu
Georgia Institute of Technology
vchu@gatech.edu

Andrea L. Thomaz
University of Texas at Austin
athomaz@ece.utexas.edu

Scott Niekum
University of Texas at Austin
sniekum@cs.utexas.edu

***Abstract*—Fully specifying a task model such that a robot can perform a task in all situations and environments is intractable. Instead, we propose a novel algorithm that allows users to update a baseline model by providing demonstrations or corrections in the environment in which the robot operates. A set of model updates are proposed that make structural changes to a finite state automaton (FSA) representation of the task. These changes are instantiated through conversion into a state transition hidden Markov model (STARHMM). The STARHMM's probabilistic properties are then used to perform approximate Bayesian model selection to choose the best model update, if any. We implement and evaluate the model selection component on a simulated block sorting domain. Initial results show that this formulation can choose models that sufficiently incorporate new demonstrations, while remaining as simple as possible.**

I. Introduction

Given the size and complexity of robotics problems, any initial model of a task is likely incomplete and would require modification upon encountering previously unseen circumstances. For example, a robot may be deployed with a set of preprogrammed task models for common tasks that fail under certain conditions or use cases unknown to the engineers. Due to the intractability of enumerating all scenarios the robot may encounter, it is advantageous to have a mechanism that allows end users to update and adapt these models by providing demonstrations or corrections in the environment in which the robot operates. This leads to the following question: How can a task model be efficiently updated to incorporate new information that may or may not fit into the existing policy defined by the task model?

Learning from demonstration (LfD) provides an intuitive mechanism to program robots, allowing them to learn new skills from example executions provided by humans [2]. While much work has focused on learning single policies for skill execution [1, 12] or sequencing a set of learned policy primitives [13], recent efforts have focused on learning task models by jointly reasoning over action primitives and their sequencing while tackling the problem of incremental learning of such task models [7, 8, 10, 11, 14]. However, these methods either require an explicit reward function definition [7, 8] or utilize computationally expensive processes such as simulated evaluation [14] or Markov chain Monte Carlo (MCMC) sampling [10, 11]. Utilizing a parametric method with closed-form inference could remove the need for a reward function while decreasing the computational overhead.

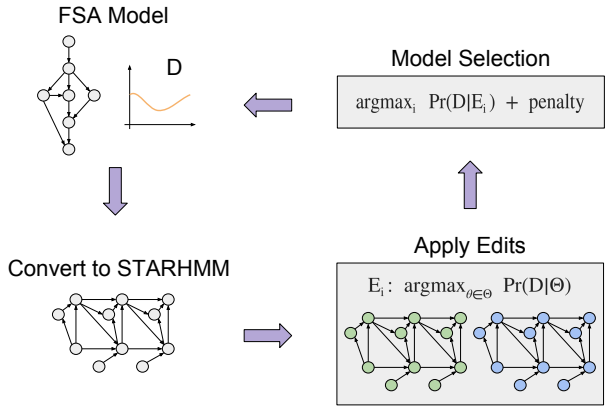


Fig. 1. The proposed incremental task model updates pipeline consists of first applying a set of edits given the demonstrations, and then comparing the edited model's likelihoods given the data. This work focuses on the model comparison step, with the edit application step replaced with a set of user-provided models.

In this work, we present a new method that utilizes corrective demonstrations to inform incremental changes to the model, which is represented as a finite-state automaton (FSA) with primitives defined by an initiation, a termination, a dynamics, and a policy model. A set of model updates are proposed that make specified structural changes to the FSA. These changes are instantiated through conversion into a state transition hidden Markov model (STARHMM) [9]. The STARHMM's probabilistic properties are then used to perform approximate Bayesian model selection, which chooses the best update to incorporate into the model. We implement and evaluate the model selection component on a simulated block sorting domain. Given demonstrations, the model selection component of the framework should be able to choose from a set of candidate models the simplest model that fits the data well. Initial results show that this method is able to select the simplest good-fit model from a set of candidate FSA updates.

II. Related Work

One way of representing complex skills is through finite state automata (FSA), which define the transitions between a set of primitives. Kappler et al. [6] learn associative skill memories (ASMs) which are sequenced through a manually-specified manipulation graph. ASMs utilize sensor traces as a feedback mechanism to enforce stereotypical behavior of the

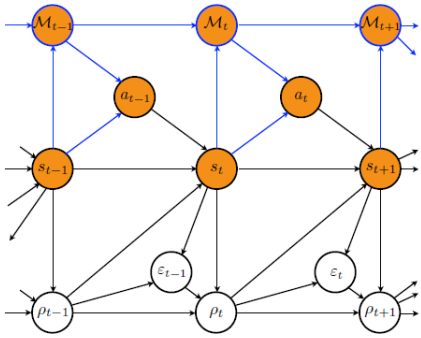


Fig. 2. Graphical representation of STARHMM [9]. While the STARHMM also has additional nodes representing high-level motor primitives (M), it is not used in this work.

encoded skills. These sensor traces can also be used to evaluate the success and failure of the skills, allowing transition into recovery behaviors dictated by the manipulation graph. Alternatively, Konidaris et al. [7] utilize reward signals to discover the subgoals of a demonstration, which provide a segmentation of the demonstration and results in a skill chain. These skill chains are then merged to form skill trees. This work was later extended to learn skill trees autonomously [8]. The work by Riano and McGinnity [14] and Niekum et al. [10, 11] are most closely related to this work. Riano and McGinnity [14] discover FSAs to accomplish a task through an evolutionary algorithm that permutes the structure of the FSA, but requires many simulated executions to evaluate an FSA's fitness. Work by Niekum et al. [10, 11] segments demonstrations with a Beta Process Autoregressive hidden Markov model (BP-AR-HMM) and constructs an FSA using those segments. This approach allows corrective demonstrations to improve the FSA, but this MCMC sampling approach does not provide a mechanism to know when to terminate the algorithm.

Interactive learning has been used in prior work to update an agent's task model. Chernova and Veloso [3] utilize their Confidence-Based Autonomy (CBA) algorithm to intelligently query a human teacher for more demonstrations in low-confidence states. These demonstrations are then used to update the agent's policy. Jain et al. [5] introduce a method for iterative improvement of trajectories in order to incorporate user preferences. Each demonstration potentially provides new information regarding task constraints, which are then incorporated into the task model. However, these methods are currently limited to low-level skills and do not reason over the sequencing of multiple low-level skills.

III. Background

We utilize two different graphical representations to model and modify tasks: (1) finite state automatons (FSAs) and (2) state transition auto-regressive hidden Markov models (STARHMMs). We represent a task using a FSA, a directed graph in which nodes represent low-level primitives and edges indicate transitions between primitives. We convert the existing FSA into a STARHMM and use the probabilistic properties of the STARHMM to evaluate changes to the FSA. This next section briefly walks through the basics of the two models before continuing to our approach.

A. Finite State Automaton

A finite state automaton (FSA) is defined as a directed graph, with nodes representing primitives and edges representing valid transitions between primitives. For the purposes of this work, each node z_i within the FSA is a primitive defined by an initiation classifier - when the primitive should begin ($\mathbb{P}^{init}$), a termination classifier - when a primitive should end ($\mathbb{P}^{term}$), a state dynamics model - how the state changes with actions ($\mathbb{P}^{dyn}$), and a policy model - when to take what action ($\mathbb{P}^{pol}$). More concretely, z_i is defined below where $i \in \{1, \ldots, \kappa\}$, $\boldsymbol{s}_t \in \mathbb{R}^n$ is the observed state at time t, $\boldsymbol{a}_t \in \mathbb{R}^m$ is the action at time t, and κ is the number of nodes in the FSA.

$$z_i = \{\mathbb{P}_i^{init}(\boldsymbol{s}_t), \mathbb{P}_i^{term}(\boldsymbol{s}_t), \mathbb{P}_i^{dyn}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t), \mathbb{P}_i^{pol}(\boldsymbol{a}_t|\boldsymbol{s}_t)\} \quad (1)$$

The edges of the graph are encoded through the following two functions.

- $parent(i)$: returns the parents of primitive i
- $children(i)$: returns the children of primitive i

A full policy execution is computed by selecting a primitive and executing its policy at each time step. The primitive selection follows the structure of the graph such that if the current primitive is z_i, the most likely primitive from the set $Z = \{z_j | j \in \{\{i\} \cup children(i)\}\}$ is selected as the primitive in the next time step.

B. State Transition Auto-Regressive Hidden Markov Model

A state transition auto-regressive hidden Markov model (STARHMM) [9] is a probabilistic graphical model that captures the entry and exit conditions that represent the subgoals of multi-phase tasks. In addition to the state and action variables outlined in Section III, a STARHMM has hidden states, which we refer to as phases ($\rho_t \in \{1, \ldots, \kappa\}$), and termination states ($\epsilon_t \in \{0, 1\}$). Hidden phases are similar to nodes within an FSA in that they index a primitive. The termination state governs when a phase can transition.

A STARHMM models the state dynamics with the distribution $\mathbb{P}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t, \rho_t)$, which is similar to the FSA, except now it also depends on the hidden phase, ρ. The phase transitions ($\rho_t \rightarrow \rho_{t+1}$) depend on the current state $\boldsymbol{s}_{t+1}$, the previous phase ρ_t, and the termination status of the previous phase ϵ_t. These dependencies are modeled with the distribution $\mathbb{P}(\rho_{t+1}|\boldsymbol{s}_{t+1}, \rho_t, \epsilon_t)$. Phase transition can only occur when the previous phase has terminated, which constrains the phase transition distribution in the following manner.

$$\mathbb{P}(\rho_{t+1}|\boldsymbol{s}_{t+1}, \rho_t, \epsilon_t = 0) = \begin{cases} 1 & \rho_{t+1} = \rho_t \\ 0 & \rho_{t+1} \neq \rho_t \end{cases} \quad (2)$$

When $\epsilon_t = 1$ (i.e. when p_t has terminated), phase transitions are governed by the initiation distribution

$\mathbb{P}(\rho_{t+1}|\boldsymbol{s}_{t+1}, \rho_t, \epsilon_t = 1)$. Finally, phase termination ϵ_t depends on the current phase ρ_t and the next state $\boldsymbol{s}_{t+1}$. This is modeled by the distribution $\mathbb{P}(\epsilon_t|\boldsymbol{s}_{t+1}, \rho_t)$. The incorporation of this auxiliary variable allows for the explicit modeling of each phase's state-dependent exit conditions. The graphical representation of this model can be seen in Fig. 2.

Given the above distribution definitions, the probability of observing a sequence of states $\boldsymbol{s}_{1:N+1}$, actions $\boldsymbol{a}_{1:N}$, phases $\rho_{1:N+1}$, and phase terminations $\epsilon_{0:N-1}$ is

$$\begin{aligned}\mathbb{P}(\boldsymbol{s}_{1:N+1}, \boldsymbol{a}_{1:N}, \rho_{1:N+1}, \rho_{0:N-1}) = \mathbb{P}(\epsilon_0, \rho_1, \boldsymbol{s}_1) \cdot \\ \prod_{t=1}^{N} \mathbb{P}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t, \rho_t)\mathbb{P}(\boldsymbol{a}_t) \prod_{t=2}^{N} \mathbb{P}(\rho_t, \epsilon_{t-1}|\boldsymbol{s}_t, \rho_{t-1})\end{aligned} \tag{3}$$

where $\mathbb{P}(\epsilon_0, \rho_1, \boldsymbol{s}_1) = \mathbb{P}(\boldsymbol{s}_1, \epsilon_0)\mathbb{P}(\rho_1|\boldsymbol{s}_1, \epsilon_0)$ and $\mathbb{P}(\rho_t, \epsilon_{t-1}|\boldsymbol{s}_t, \rho_{t-1}) = \mathbb{P}(\rho_t|\boldsymbol{s}_t, \rho_{t-1}, \epsilon_{t-1})\mathbb{P}(\epsilon_{t-1}|\boldsymbol{s}_t, \rho_{t-1})$.

IV. Approach

As described earlier, our goal is to take a model of a task and make use of corrective demonstrations to inform incremental changes to this model. Our approach uses the following two-step method for incremental task model updates: (1) searching for model updates defined by a set of candidate corrections and (2) selecting the best model from this set.

Given a set of demonstrations for an error condition induced by an incomplete or incorrect FSA, we generate a set of candidate corrections to the model. We make candidate corrections according to a predefined set of edit types, which correspond to structural changes of the FSA. These edit types are: node modification, node addition, and edge addition. A candidate correction is then defined as a set of decisions over these edit types, which cover a range of transformations needed to correct different modeling errors. For example, a small change to the policy may only require modifying an existing node, while learning new sub-skills would require additional nodes. In this way, each candidate correction defines a search direction in the space of possible FSAs. For each candidate correction, we find an associated model update through a search procedure that learns the parameters of a STARHMM that was created by converting the FSA into a STARHMM. The search is constrained to modifications specific to the candidate correction (e.g. allowing only node modification). We use the corrective demonstrations to instantiate these models through the Expectation-Maximization (EM) algorithm for STARHMMs [9]. Then, the system chooses the most likely correction through approximate Bayesian model comparison. The overall pipeline can be seen in Fig. 1.

A. FSA-STARHMM Conversion

In order to instantiate the model updates needed for model comparison, the FSA must be converted to an equivalent STARHMM. The initiation and termination classifiers of the FSA are modeled using logistic regression, where $\boldsymbol{\omega}_i^{init} \in \mathbb{R}^d$ and $\boldsymbol{\omega}_i^{term} \in \mathbb{R}^d$ are the weights for the initiation and termination classifiers and $\boldsymbol{\phi}(\boldsymbol{s_t}) \in \mathbb{R}^d$ is the feature vector for state $\boldsymbol{s_t}$.

$$\mathbb{P}_i^{init}(\boldsymbol{s}_t) = \frac{1}{1 + e^{-\boldsymbol{\omega}_i^{init\mathsf{T}}\boldsymbol{\phi}(\boldsymbol{s}_t)}} \tag{4}$$

$$\mathbb{P}_i^{term}(\boldsymbol{s}_t) = \frac{1}{1 + e^{-\boldsymbol{\omega}_i^{term\mathsf{T}}\boldsymbol{\phi}(\boldsymbol{s}_t)}} \tag{5}$$

The dynamics and policy of each primitive are represented as linear Gaussian models

$$\mathbb{P}_i^{dyn}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t) = \mathcal{N}(\boldsymbol{A}_i\boldsymbol{s}_t + \boldsymbol{B}_i\boldsymbol{a}_t, \boldsymbol{\Sigma}_i^{dyn}) \tag{6}$$

$$\mathbb{P}_i^{pol}(\boldsymbol{a}_t|\boldsymbol{s}_t) = \mathcal{N}(\boldsymbol{V}_i\boldsymbol{s}_t, \boldsymbol{\Sigma}_i^{pol}) \tag{7}$$

where $\boldsymbol{A}_i \in \mathbb{R}^{n\times n}$, $\boldsymbol{B}_i \in \mathbb{R}^{n\times m}$, $\boldsymbol{V}_i \in \mathbb{R}^{m\times n}$, $\boldsymbol{\Sigma}_i^{dyn} \in \mathbb{R}^{n\times n}$, and $\boldsymbol{\Sigma}_i^{pol} \in \mathbb{R}^{m\times m}$ are specific to each primitive z_i.

Each phase in the STARHMM indexes a primitive in the FSA. Using the FSA models, we parameterize the termination and state transition distributions of the STARHMM as

$$\mathbb{P}(\epsilon_t|\boldsymbol{s}_{t+1}, \rho_t = i) = \begin{cases} 1 - \mathbb{P}_i^{term}(\boldsymbol{s}_{t+1}) & \epsilon_t = 0 \\ \mathbb{P}_i^{term}(\boldsymbol{s}_{t+1}) & \epsilon_t = 1 \end{cases} \tag{8}$$

$$\mathbb{P}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t, \rho_t = i) = \mathbb{P}_i^{dyn}(s_{t+1}|s_t, a_t) \tag{9}$$

thus directly mapping the termination classifier and dynamics model of each primitive in the FSA to the termination distribution and state transition distribution of the associated phase in the STARHMM, respectively.

The initiation distribution $\mathbb{P}(\rho_{t+1}|\boldsymbol{s}_{t+1}, \rho_t, \epsilon_t = 1)$ takes on different parameterizations depending on the edit type during model learning and the structure of the FSA during model selection. These parameterizations all take the following form

$$\begin{aligned}\mathbb{P}(\rho_{t+1} = j|\boldsymbol{s}_{t+1}, \rho_t, \epsilon_t = 1) &= \begin{cases} p & j \in T(\rho_t) \\ 0 & j \notin T(\rho_t) \end{cases} \\ p &= \frac{\mathbb{P}_j^{init}(\boldsymbol{s}_{t+1})\mathbb{P}(\rho_{t+1} = j|\rho_t)}{\sum_{k\in T(i)} \mathbb{P}_k^{init}(\boldsymbol{s}_{t+1})\mathbb{P}(\rho_{t+1} = j|\rho_t)}\end{aligned} \tag{10}$$

where T is a function that takes a phase and returns the set of allowable transitions. Sections IV-B and IV-D define T for the model learning and selection steps respectively. The distribution $\mathbb{P}(\rho_{t+1}|\rho_t)$ defines the prior probability of transitioning from ρ_t to ρ_{t+1}. In the general case, this distribution can be estimated by keeping a running count of all transitions seen. In the simplest case, the allowable transitions can all be given an equal prior probability. With this parameterization, the phase transition distribution for ρ_t can be constructed from the initiation classifiers of the primitives in $T(\rho_t)$. In other words, the transition probabilities for primitives in $T(\rho_t)$ are governed by the initiation classifiers of all primitives in $T(\rho_t)$, while all primitives not in $T(\rho_t)$ have a transition probability of zero. With these definitions, a traversal of the FSA corresponds to a ρ sequence assignment in the STARHMM.

B. Candidate Correction Application

A candidate correction is defined as a set of decisions over the following edit types: node modification, node addition, and edge addition. Specifically, a candidate correction can allow node modification; allow the addition of $K \geq 1$ new nodes; and/or allow the addition of new edges. This leads to a total of $4K + 3$ possible candidate corrections. The decisions over the edit types define the free parameters $\mathbf{\Gamma} = \{\mathbf{\Theta}, T\}$ for the model learning procedure, where $\mathbf{\Theta}$ is the set of node model parameters and T is a function that returns the set of allowable transitions for each phase.

EDIT TYPES: Each edit type defines its own set of free parameters $\boldsymbol{\gamma} = \{\boldsymbol{\theta}, \tau\}$. In the following, κ is the current number of nodes and the notation $\boldsymbol{\theta}_i = \{\boldsymbol{\omega}_i^{init}, \boldsymbol{\omega}_i^{term}, \boldsymbol{A}_i, \boldsymbol{B}_i, \boldsymbol{\Sigma}_i^{dyn}\}$ is used to denote the set of model parameters in a STARHMM for node z_i. We describe each edit type in detail below. The full set of free parameters for a candidate correction is the union over the edit type free parameters ($\mathbf{\Theta} = \boldsymbol{\theta}^{n_{mod}} \cup \boldsymbol{\theta}^{n_{add}} \cup \boldsymbol{\theta}^{e_{add}}$ and $T(i) = \tau^{fsa}(i) \cup \tau^{n_{mod}}(i) \cup \tau^{n_{add}}(i) \cup \tau^{e_{add}}(i)$, where $\tau^{fsa}(i) = \{i\} \cup children(i)$).

Node Modification: Allowing node modification sets all current node model parameters as free ($\boldsymbol{\theta}^{n_{mod}} = \{\boldsymbol{\theta}_i | i \in \{1, \ldots, \kappa\}\}$). If node modification is not allowed, there are no free node model parameters ($\boldsymbol{\theta}^{n_{mod}} = \{\}$). In both cases, $\tau^{n_{mod}}(i) = \{\}$.

Node Addition: Allowing node addition creates a set of free parameters for each new node ($\boldsymbol{\theta}^{n_{add}} = \{\boldsymbol{\theta}_i | i \in \{\kappa+1 \ldots, \kappa+K\}\}$) and allows all transitions to and from these new nodes. Concretely, if $i \leq \kappa$ then $\tau^{n_{add}}(i) = \{j | j \in \{\kappa + 1, \ldots, \kappa + K\}\}$; if $i > \kappa$ then $\tau^{n_{add}}(i) = \{j | j \in \{1, \ldots, \kappa + K\}\}$. If node addition is not allowed, $\boldsymbol{\theta}^{n_{add}} = \{\}$ and $\tau^{n_{add}}(i) = \{\}$.

Edge Addition: Allowing edge addition does not create new free node model parameters ($\boldsymbol{\theta}^{e_{add}} = \{\}$) but allows for potential transitions between all current nodes ($\tau^{e_{add}}(i) = \{j | j \in \{1, \ldots, \kappa\}\}$). Several new edges could be added, provided the demonstration dictates their necessity. If edge addition is not allowed, $\boldsymbol{\theta}^{e_{add}} = \{\}$ and $\tau^{e_{add}}(i) = \{\}$.

MODEL LEARNING: For each candidate correction, a new STARHMM is instantiated with its parameters initialized according to the FSA-STARHMM conversion procedure outlined in Section IV-A. In the full pipeline, the set of demonstrations will be used to train the STARHMMs according to the free parameters of the candidate correction. Expectation-Maximization is used to update the parameters $\mathbf{\Theta}$ with transitions governed by T. In this work, the model learning step is replaced with a set of user-defined models. This allows for the evaluation of the subsequent model selection steps.

C. FSA Updates

For each of the STARHMMs learned in the previous step, the corresponding FSA can be constructed by first replacing the initiation, termination, and dynamics models of all primitives in the current FSA with the corresponding models in the new STARHMM as defined in Section IV-A. Then, the new STARHMM is used to infer the maximum likelihood primitive sequence given the corrective demonstrations by setting T according to the candidate correction (Section IV-B) and running the max-product algorithm. The resultant sequence is a combination of primitives in the current FSA and the new learned primitives. After removing redundancies in the sequence (e.g. $1, 2, 2, 2, 3, 3 \rightarrow 1, 2, 3$), a new FSA is defined by iteratively merging the sequence with the current FSA using a process similar to the one outlined by Konidaris et al. [7]; elements in the sequence are merged with the nodes they index, with new edges and nodes defined through the unmerged elements. As an example, suppose that the subsequence $2, 7, 4$ appears in the maximum likelihood sequence and 7 is a new primitive. This corresponds to a new node being added to the FSA with parent z_2 and child z_4. The policies $\mathbb{P}_i^{pol}$ for the new primitives added to the FSA are learned by training the policy models on the segments of the demonstrations assigned to each primitive by the maximum likelihood sequences.

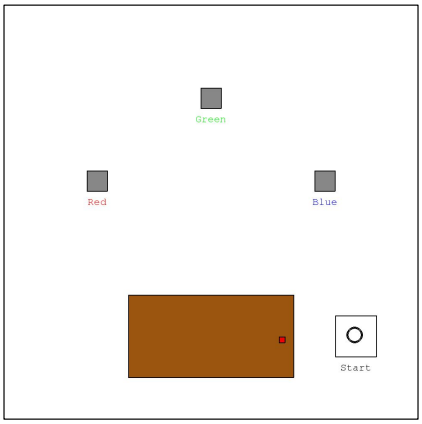


Fig. 3. The simulation environment allows a user to give demonstrations.

D. Model Selection

Each corrective demonstration induces a most likely primitive sequence in each of the learned FSAs. The likelihoods of these sequences are computed by running the max-product algorithm on each model's associated STARHMM, with the allowed transitions following the FSA structure. This corresponds to

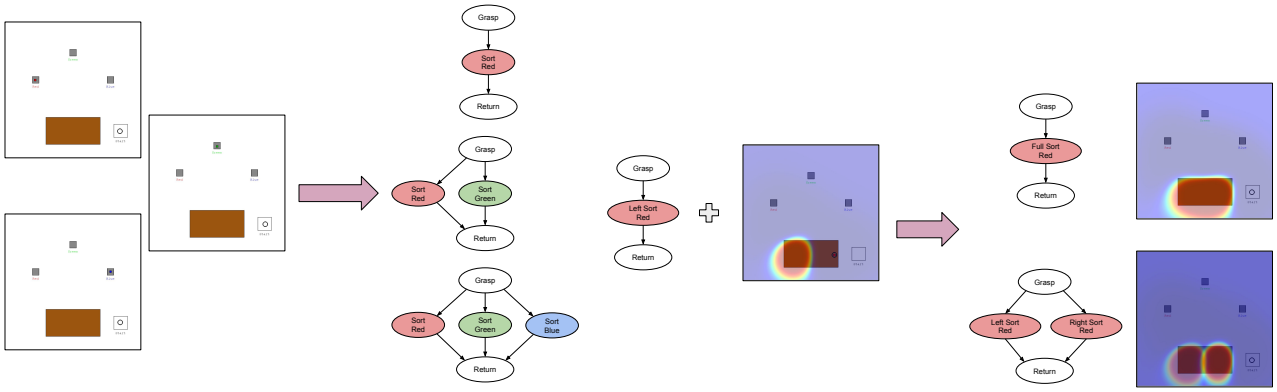
$$T(i) = \{i\} \cup children(i) \tag{11}$$

In order to avoid over-fitting, the likelihoods must be balanced with model complexity. Though we are currently exploring full Bayesian model comparison, we presently use the approximation given by the Akaike Information Criterion (AIC), defined below

$$AIC = 2k - 2\ln(L) \tag{12}$$

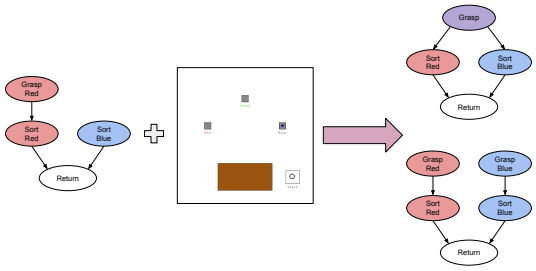
where k is the number of parameters and L is the likelihood. The model with the minimum AIC score is the best fit model. The number of parameters k is computed as follows

$$k = \sum_{i=1}^{\kappa} \eta_i(|\boldsymbol{\omega}_i^{init}| + |\boldsymbol{\omega}_i^{term}|) + |\boldsymbol{A}_i| + |\boldsymbol{B}_i| + |\boldsymbol{\Sigma}_i^{dyn}| \tag{13}$$



(a) Node Addition

(b) Node Modification



(c) Edge Addition

Fig. 4. The model selection component of the incremental update pipeline was evaluated using three experiments. In experiment 1 (a), the system should select the simplest model that fits each color sort demonstration (e.g. for green sort, the second model is best). In experiment 2 (b) and 3 (c), the system selected between two plausible model updates given the prior FSA plus a new demonstration. In experiment 2 (b), an initial model that can only sort from half the table must be updated by either modifying its current sort primitive or adding a new sort primitive. In experiment 3 (c), an initial model that can only sort blue blocks if they are placed in the gripper must be updated by either modifying its current red grasp primitive or adding a new grasp primitive.

This corresponds to summing the number of transition and dynamics parameters of an FSA's corresponding STARHMM. The η_i parameters are weighting factors that account for the number of times the ith primitive is used in the construction of the phase transition distribution. With the above T definition,

$$\eta_i = |parent(i)| + \mathbb{1}(|children(i)| \geq 1) \tag{14}$$

Thus, the number of parameters is proportional to the number of nodes and edges in the FSA.

E. Initial FSA Construction

The above sections outlined the task model update pipeline assuming a preexisting FSA. Thus, a method is required that can construct an initial FSA to start the process. We propose two methods to initialize the task model update pipeline. After initialization with either of these methods, the learning procedure is run to fine-tune the parameters.

USER-DEFINED: Under this method, a user is asked to provide a semantic breakdown of the task whereby a set of requisite steps are described. For each of these steps, the user provides examples of the state configurations that describe the step's goal. The user provides demonstrations for the execution of each step as well as their sequential ordering.

Each step corresponds to a primitive in the FSA, whose structure is defined by the user-provided ordering. The initiation and termination classifiers are initialized by training the logistic regression models under the positive-unlabeled paradigm described by Elkan and Noto [4]. The goal state demonstrations of each step are used as positive examples for that primitive's termination model, while its initiation model uses its parents' goal state demonstrations as positive examples. All demonstrations can be used as unlabeled examples for all initiation and termination classifiers. This leaves the first primitive's initiation model undefined. To resolve this, all points within a specified distance of the first points of each demonstration for that primitive are used as positive examples. Finally, the dynamics and policy models are trained using the states and actions recorded during demonstration.

AUTOMATED: Under this method, the user provides demonstrations of the full execution of the task. The making and breaking of object contacts can be used to define an initial segmentation into primitives by spectral clustering using a similarity metric defined by contact distribution kernels, as outlined by Kroemer et al. [9]. The FSA structure is defined by inferring the demonstrations' primitive traversal and following the merging procedure outlined in Section IV-C. Exploration of this technique is a topic of future work.

V. EXPERIMENTS

This evaluation is aimed at the model selection component of our approach, which was conducted on a simulated block sorting domain. The simulation environment consists of a table, three bins, and a gripper start area. A user can move

		Task Model		
		R	RG	RGB
Demo	Red	**-17716.53**	-10533.59	-3693.94
	Green	∞	**-2492.32**	3859.02
	Blue	∞	7299.87	**5190.70**

(a) Node Addition AIC Scores

		Task Model			
		L	R	LR	L/R
Demo	Left	-	-10364.88	**-13878.30**	-11640.95
	Right	-5775.34	-	**-10873.87**	-8152.87

(b) Node Modification AIC Scores

		Task Model		
		RB*	RB	R/B
Demo	Blue	34880.08	**-771.55**	-380.01

(c) Edge Addition AIC Scores

Fig. 5. In experiment 1 (a), the system selected the smallest model that encodes each color sort demonstration. In experiment 2 (b), the system opted to expand the initiation classifier to encompass the whole table (LR) instead of adding a new primitive (L/R). In experiment 3 (c), the system optde to add an edge (RB) to model the blue sort demonstration instead of adding a new primitive (R/B).

the gripper and grasp blocks using the mouse. Blocks appear on the table with a random color within a pre-specified range of red, green, and blue (i.e for a red sort task, the block color is sampled from a red color distribution). The simulated environment can be seen in Figure 3. The state space of this environment consists of the gripper location $\boldsymbol{e} = (e_x, e_y)$, block location $\boldsymbol{b} = (b_x, b_y)$, block color $\boldsymbol{c} = (r, g, b)$, and the distances between the gripper and each of the bins ($\boldsymbol{n}^r = (n_x^r, n_y^r)$, $\boldsymbol{n}^g = (n_x^g, n_y^g)$, $\boldsymbol{n}^b = (n_x^b, n_y^b)$) and block: $\boldsymbol{S} = (\boldsymbol{e}, \boldsymbol{b}, \boldsymbol{c}, \|\boldsymbol{e} - \boldsymbol{n}^r\|, \|\boldsymbol{e} - \boldsymbol{n}^g\|, \|\boldsymbol{e} - \boldsymbol{n}^b\|, \|\boldsymbol{e} - \boldsymbol{b}\|)$. The action at each time step was the displacement of the gripper and a binary variable k indicating if the gripper is closed: $\boldsymbol{A} = (\Delta\boldsymbol{e}, k)$.

For each of the experiments described below, expert demonstrations were used to construct the FSAs using the first of the two methods outlined in Section IV-E.

A. Experiment 1: Node Addition

For the first experiment, three models were constructed: the first can only sort red blocks; the second can only sort red and green blocks; and the third can sort red, green, and blue blocks. One of the authors provided a sorting demonstration for each color, moving a block from the table to the bin of that color and returning the gripper to the start location. Then, the model selection component of our approach was used to select the appropriate model for each demonstration. The FSAs used for this experiment can be seen in Figure 4(a).

The AIC scores for each model and demonstration are shown in Figure 5(a). As can be seen, the correct model was selected for each demonstration. Specifically, the simplest FSA that encodes each demonstration was chosen.

B. Experiment 2: Node Modification

For the second experiment, two initial models were constructed: red block sorting from the left side of the table; and red block sorting from the right side of the table. In other words, they were trained such that the initiation of the sort primitive only covers half of the table. For the left sort model, an expert demonstration was provided that moved a red block from the right side of the table to the bin and returned the gripper to the start location. Then, the model selection component of our approach was used to select between two plausible model updates: expanding the sort initiation to cover the entire table (node modification) or adding a primitive to sort from the right side. We also ran a symmetric experiment for the right sort model. The FSAs used in this experiment can be seen in Figure 4(b).

The AIC scores for each model and demonstration are shown in Figure 5(b). As can be seen, the system opted to expand the sort primitive initiation classifier (LR) in each case since the increase in likelihood achieved when adding a new primitive wasn't enough to counterbalance the increase in parameters.

C. Experiment 3: Edge Addition

For the third experiment, an initial model was constructed that can sort red blocks but can't fully sort blue blocks. Specifically, the model couldn't grasp blue blocks, but could sort them if they were placed in the gripper. An expert demonstration was provided that moved a blue block from the table to the blue bin and returned the gripper to the start location. Then, the model selection component of our approach was used to select between two plausible model updates: adding an edge from the grasp to sort blue primitive or adding a new grasp blue primitive. The FSAs used in this experiment can be seen in Figure 4(c).

The AIC scores for each model are shown in Figure 5(c). As can be seen, the system opted to add a new edge (RB) since the increase in likelihood achieved when adding a new primitive wasn't enough to counterbalance the increase in parameters.

VI. Conclusion

As robots enter unstructured real-world environments, they will require a way to incrementally update and adapt their task models in order to account for unforeseen scenarios. We introduce an approach to discover such model updates through iterative constrained search and selection, covering a range of transformations needed to correct different modeling errors. Given demonstrations, model updates are found that make specified changes to the task model, after which the best among these updates is selected for incorporation into the original task model. The selection component of this pipeline was evaluated in this work.

We demonstrated that given demonstrations of un-modeled behavior, the proposed model selection framework chooses the simplest task model that fits the demonstrations. This then served as an update to the original task model. Each of the experiments resulted in the selection of the simplest good-fit model. Thus, our experiments indicate that STARHMM representations of FSA task models can be used to perform this model selection.

The immediate next step is the implementation of the automated candidate correction model learning procedure, as well as testing the full pipeline in more complex domains. The current formulation must use the entire set of objects in each of the of the initiation, termination, and dynamics models. This will lead to computational inefficiencies as the state space grows. Thus, it would be useful to modify the STARHMM to select the relevant object subset for each model. In addition, incorporating policy into the STARHMM would allow more direct learning and adaptation of the policy models.

ACKNOWLEDGMENTS

This work has taken place in the Socially Intelligent Machines (SIM) lab and the Personal Autonomous Robotics Lab (PeARL) at The University of Texas at Austin. SIM and PeARL research is supported in part by the National Science Foundation (IIS-1638107).

REFERENCES

[1] Baris Akgun, Maya Cakmak, Karl Jiang, and Andrea L Thomaz. Keyframe-based learning from demonstration. *International Journal of Social Robotics*, 4(4):343–355, 2012.

[2] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57 (5):469–483, 2009.

[3] Sonia Chernova and Manuela Veloso. Interactive policy learning through confidence-based autonomy. *Journal of Artificial Intelligence Research*, 34(1):1, 2009.

[4] Charles Elkan and Keith Noto. Learning classifiers from only positive and unlabeled data. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 213–220. ACM, 2008.

[5] Ashesh Jain, Brian Wojcik, Thorsten Joachims, and Ashutosh Saxena. Learning trajectory preferences for manipulators via iterative improvement. In *Advances in neural information processing systems*, pages 575–583, 2013.

[6] Daniel Kappler, Peter Pastor, Mrinal Kalakrishnan, Manuel Wüthrich, and Stefan Schaal. Data-Driven Online Decision Making for Autonomous Manipulation. In *Robotics: Science and Systems*, 2015.

[7] George Konidaris, Scott Kuindersma, Roderic Grupen, and Andrew Barto. CST: Constructing Skill Trees by Demonstration. *Doctoral Dissertations, University of Massachuts, Ahmrest*, 2011.

[8] George Konidaris, Scott Kuindersma, Roderic A Grupen, and Andrew G Barto. Autonomous Skill Acquisition on a Mobile Manipulator. In *AAAI*, 2011.

[9] Oliver Kroemer, Christian Daniel, Gerhard Neumann, Herke Van Hoof, and Jan Peters. Towards learning hierarchical skills for multi-phase manipulation tasks. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 1503–1510. IEEE, 2015.

[10] Scott Niekum, Sarah Osentoski, George Konidaris, and Andrew G Barto. Learning and generalization of complex tasks from unstructured demonstrations. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5239–5246. IEEE, 2012.

[11] Scott Niekum, Sachin Chitta, Andrew G Barto, Bhaskara Marthi, and Sarah Osentoski. Incremental Semantically Grounded Learning from Demonstration. In *Robotics: Science and Systems*, volume 9, 2013.

[12] Peter Pastor, Heiko Hoffmann, Tamim Asfour, and Stefan Schaal. Learning and generalization of motor skills by learning from demonstration. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 763–768. IEEE, 2009.

[13] Peter Pastor, Mrinal Kalakrishnan, Ludovic Righetti, and Stefan Schaal. Towards associative skill memories. In *Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on*, pages 309–315. IEEE, 2012.

[14] Lorenzo Riano and T Martin McGinnity. Automatically composing and parameterizing skills by evolving finite state automata. *Robotics and Autonomous Systems*, 60 (4):639–650, 2012.