Incremental Task Modification via Corrective Demonstrations

Reymundo A. Gutierrez[1] Vivian Chu[2] Andrea L. Thomaz[3] and Scott Niekum[1]

***Abstract*— In realistic environments, fully specifying a task model such that a robot can perform a task in all situations is impractical. In this work, we present Incremental Task Modification via Corrective Demonstrations (ITMCD), a novel algorithm that allows a robot to update a learned model by making use of corrective demonstrations from an end-user in its environment. We propose three different types of model updates that make structural changes to a finite state automaton (FSA) representation of the task by first converting the FSA into a state transition auto-regressive 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 evaluate ITMCD Model Selection in a simulated block sorting domain and the full algorithm on a real-world pouring task. The simulation results show our approach can choose new task models that sufficiently incorporate new demonstrations while remaining as simple as possible. The results from the pouring task show that ITMCD performs well when the modeled segments of the corrective demonstrations closely comply with the original task model.**

I. Introduction

Robots deployed into real-world environments will inevitably encounter situations not covered by their initial task models and will need to incorporate new information to handle these unanticipated circumstances (e.g. an engineer cannot anticipate all conditions needed for a robot deployed with a set of preprogrammed household task models). Due to the intractability of enumerating all scenarios the robot may encounter, it is advantageous to have mechanisms that allow robots to interact with end users to update and adapt task models by providing corrective demonstrations in the environment in which the robot operates. This leads to the question: How can a task model be efficiently updated to incorporate new information when that information may or may not fit into the policy defined by the existing model?

Learning from demonstration (LfD) allows robots to learn new skills from example task executions provided by humans [1, 2]. While much work has focused on learning single policies for skill execution [3, 4] or sequencing a set of learned policy primitives [5, 6], recent efforts have focused on learning task models by jointly reasoning over action primitives and their sequences while tackling the problem of incremental learning of such task models [7]–[12]. However, these methods require an explicit reward function definition [8, 9], utilize computationally expensive processes such as a simulated evaluation [12] or Markov chain Monte Carlo (MCMC) sampling [10, 11], or require hand-coded segments [7]. Utilizing a parametric method with closed-form inference could remove the need for a reward function while decreasing the computational overhead.

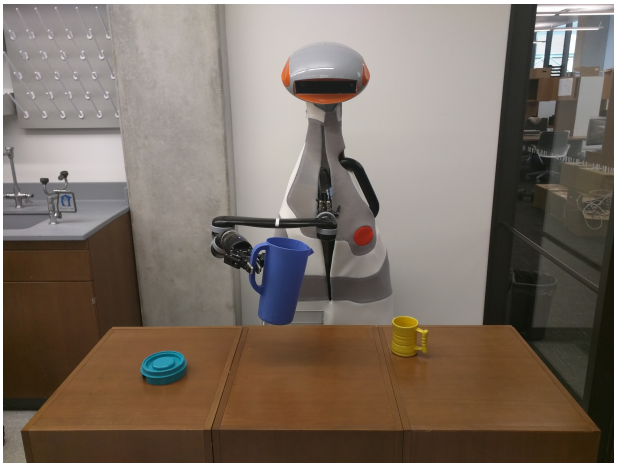


Fig. 1. ITMCD was evaluated on a pouring task, where the pitcher can be either open or closed. Starting from a task model that can complete the pouring task in the open condition, the objective is to learn the task model to operate in the closed condition given corrective demonstrations.

In this work, we present Incremental Task Modification via Corrective Demonstrations (ITMCD), a new method that utilizes corrective demonstrations to inform incremental changes to a task model represented as a finite-state automaton (FSA) whose primitives are specified by initiation, termination, dynamics, and policy models. Given a new demonstration, 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 auto-regressive hidden Markov model (STARHMM) [13]. 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 evaluate the ITMCD Model Selection step on a simulated block sorting domain and the full algorithm on a real-world pouring task. Given corrective demonstrations, ITMCD Model Selection is able to select the simplest good-fit model from a set of candidate FSA updates. Although the primitive representation used was sensitive to small variations in the modeled trajectories, ITMCD is able to appropriately add new primitives to the task model when the modeled segments of the corrective demonstrations comply with the original task model.

[1] Department of Computer Science, University of Texas at Austin, Austin, Texas 78712 Email: {ragtz,sniekum}@cs.utexas.edu

[2] School of Interactive Computing, Georgia Institute of Technology, Atlanta, Georgia 30332-0250 Email: vchu@gatech.edu

[3] Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, Texas 78712 Email: athomaz@ece.utexas.edu

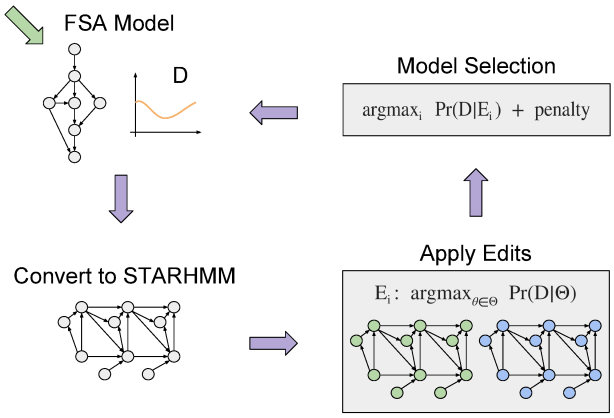


Fig. 2. The ITMCD algorithm starts with an FSA model and applies a set of edits given the demonstrations. Then, the edited model's likelihoods given the data are compared to select an update to the model.

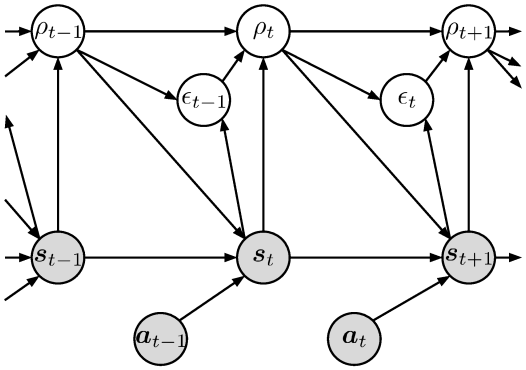


Fig. 3. Graphical representation of STARHMM [13]. The model includes states $\boldsymbol{s}$, actions $\boldsymbol{a}$, phases ρ and termination variables ϵ. The gray nodes are observed variables, while the white nodes are hidden variables.

II. Related Work

One method of representing complex skills needed by robots is through finite state automata (FSA), which define the transitions between a set of primitives. Kappler et al. [7] 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 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. [8] 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 [9]. In closely-related work, Riano and McGinnity [12] 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. Niekum et al. [10, 11] segment 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 for algorithm termination.

Interactive learning has been used in prior work to update an agent's task model. Akgun et al. [3] introduced a method for learning keyframe-based models of skills, and an interaction method to iteratively add/remove keyframes in the learned model in subsequent interactions. Embodied Queries is an active learning approach in which a robot queries a human to gain specific demonstrations or follow-up information about a recently learned skill [14]. In work by Sauser et al. [15], a teacher can provide tactile feedback to a robot during playback of a learned skill to adapt or correct object grasping behaviors. The Confidence-Based Autonomy (CBA) algorithm intelligently queries a human teacher for more demonstrations in low-confidence states [16]. Jain et al. [17] 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 use two different graphical representations to model and modify tasks: (1) finite state automaton (FSA) and (2) state transition auto-regressive hidden Markov model (STARHMM). We represent a task using an FSA, a directed graph in which nodes represent low-level primitives and edges indicate transitions between primitives. We convert this existing FSA into a STARHMM and use the probabilistic properties of the STARHMM to evaluate changes to the FSA. This next section briefly discusses these two models before continuing to our algorithm, ITMCD.

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. In 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 - describing when a primitive should end ($\mathbb{P}^{term}$), a state dynamics model - describing how the state changes with actions ($\mathbb{P}^{dyn}$), and a policy model - describing when to take what action (π). 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.

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

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) [13] 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 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.

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

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. 3.

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, the goal of this work is to take a model of a task and make use of corrective demonstrations to inform incremental changes to this model. We present the Incremental Task Modification via Corrective Demonstrations (ITMCD) algorithm, which 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. The high-level depiction of this algorithm can be seen in Fig. 2.

We assume that there exists an original task model FSA. Given a new situation in which this task model is not able to be applied successfully, we want to allow an end-user to provide corrective demonstrations that show how the task should be achieved in this new setting. Given a set of such corrective demonstrations, we generate a set of candidate corrections to the task model, corresponding to structural changes to the FSA: node modification, node addition, and edge addition. A candidate correction is a set of decisions over these edit types, covering a range of transformations needed to correct different modeling errors. For example, a small change such as modifying the initiation conditions of a node, 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 is 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 [13]. Then ITMCD chooses the most likely correction through approximate Bayesian model comparison. The remainder of this section details each of these steps.

A. FSA-STARHMM Conversion

In order to instantiate the model updates needed for model comparison, the original 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 $\boldsymbol{s_t}$.

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

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

The dynamics of each primitive are represented as linear Gaussian models

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

where $\boldsymbol{A}_i \in \mathbb{R}^{n \times n}$, $\boldsymbol{B}_i \in \mathbb{R}^{n \times m}$, and $\boldsymbol{\Sigma}_i \in \mathbb{R}^{n \times n}$ are specific to each primitive z_i.The policy of each primitive π_i can be derived with any algorithm that can be trained on trajectory segments (e.g. dynamic motion primitives [4]).

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

$$\mathbb{P}(\epsilon_t|\boldsymbol{s}_{t+1}, \rho_t = i) = \begin{cases} 1 - \mathbb{P}_i^{term}(\boldsymbol{s}_{t+1}) & \epsilon_t = 0 \\ \mathbb{P}_i^{term}(\boldsymbol{s}_{t+1}) & \epsilon_t = 1 \end{cases} \tag{7}$$

$$\mathbb{P}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t, \rho_t = i) = \mathbb{P}_i^{dyn}(s_{t+1}|s_t, a_t) \tag{8}$$

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

$$\begin{aligned} \mathbb{P}(\rho_{t+1} = j|\boldsymbol{s}_{t+1}, \rho_t, \epsilon_t = 1) &= \begin{cases} p & j \in T(\rho_t) \\ 0 & j \notin T(\rho_t) \end{cases} \\ p &= \frac{\mathbb{P}_j^{init}(\boldsymbol{s}_{t+1})\mathbb{P}(\rho_{t+1} = j|\rho_t)}{\sum_{k \in T(i)} \mathbb{P}_k^{init}(\boldsymbol{s}_{t+1})\mathbb{P}(\rho_{t+1} = j|\rho_t)} \end{aligned} \tag{9}$$

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. Note that the candidate corrections do not specify which subset of nodes/edges are being modified. The decisions over the edit types define the free parameters $\boldsymbol{\Gamma} = \{\boldsymbol{\Theta}, T\}$ for the model learning procedure, where $\boldsymbol{\Theta}$ is the set of node model parameters and T is a function that returns the set of allowable transitions for each phase.

1) 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 $\boldsymbol{\theta}_i = \{\boldsymbol{\omega}_i^{init}, \boldsymbol{\omega}_i^{term}, \boldsymbol{A}_i, \boldsymbol{B}_i, \boldsymbol{\Sigma}_i\}$ 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 ($\boldsymbol{\Theta} = \boldsymbol{\theta}^{n_{mod}} \cup \boldsymbol{\theta}^{n_{add}} \cup \boldsymbol{\theta}^{e_{add}}$ 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 $\tau^{e_{add}}(i) = \{\}$.

2) 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. The union of the set of prior demonstrations and the new set of corrective demonstrations is used to train the STARHMMs according to the candidate correction, using the Expectation-Maximization algorithm for STARHMMs [13] to update the parameters $\boldsymbol{\Theta}$ with transitions governed by T.

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 Viterbi 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 [8]: 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

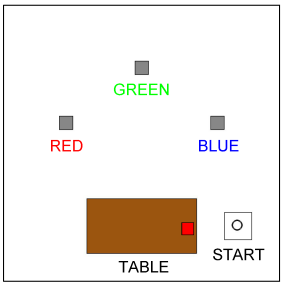


Fig. 4. The simulation environment allows a user to give block sorting demonstrations. A block appears on the table, which the user then grasps with the cursor and moves it to the appropriate bin.

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 π_i 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.

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 Viterbi algorithm on each model's associated STARHMM, with the allowed transitions following the FSA structure. This corresponds to:

$$T(i) = \{i\} \cup children(i) \tag{10}$$

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), shown below:

$$AIC = 2k - 2\ln(L) \tag{11}$$

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

$$k = \sum_{i=1}^{\kappa} \eta_i(|\boldsymbol{\omega}_i^{init}| + |\boldsymbol{\omega}_i^{term}|) + |\boldsymbol{A}_i| + |\boldsymbol{B}_i| + |\boldsymbol{\Sigma}_i| \tag{12}$$

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,

$$\eta_i = |parent(i)| + \mathbb{1}(|children(i)| \geq 1) \tag{13}$$

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 ITMCD 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.

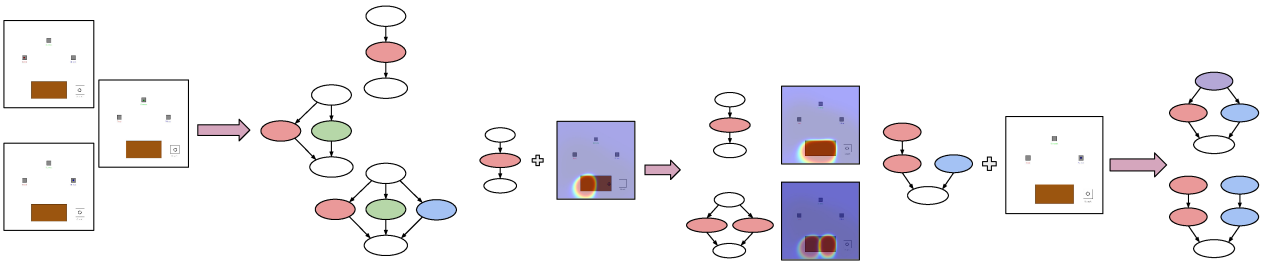
1) User-Defined: Under this method, a user is asked to provide a semantic breakdown of the task whereby a set of requisite steps are described. For each of these steps, the user provides examples of the state configurations that describe the step's goal. The user provides demonstrations for the execution 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 [18]. 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.

2) Automated: Under this method, the user provides demonstrations of the full execution of the task, which are then used to train a STARHMM. An initial estimate of the segmentation is needed to start the learning procedure. Kroemer et al. [13] accomplish this through spectral clustering using a similarity metric defined by contact distribution kernels, effectively utilizing the making and breaking of object contacts to define an initial segmentation into primitives. This method assumes a one-to-one mapping from contact configurations to primitives, which does not always hold. Instead, we make use of a Gaussian Mixture Model (GMM) over state, action pairs to produce the initial segmentation. The FSA structure is defined by inferring the demonstrations' primitive traversal and following the merging procedure outlined in Section IV-C.

F. New Primitive Initialization

The node addition edit type requires a procedure to initialize the model parameters of the new primitives. We accomplish this by first estimating which points of the demonstration are outside the current model by performing anomaly detection utilizing the Gaussian components of the state, action GMM learned during the initial model construction. A new GMM with K components is then learned over the anomalous data, providing the segmentation estimates for the new primitives. These new Gaussian components are added to the existing set for subsequent rounds of node addition.



(a) Node Addition (b) Node Modification (c) Edge Addition

Fig. 5. We evaluate ITMCD Model Selection with three experiments. (a) ITMCD should select the simplest color sort model (e.g. for green sort, the second model is best). (b) and (c) Given an FSA and a corrective demonstration, ITMCD selected between two plausible model updates. In (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 (c), an initial model that can only sort blue blocks placed in the gripper and must be updated by either modifying its current red grasp primitive or adding a new grasp primitive.

V. Experiments

This evaluation is conducted in two phases. The first is aimed at evaluating ITMCD Model Selection across the range of edit types and is conducted on a simulated block sorting domain. The second phase evaluates the full ITMCD algorithm on a pouring task performed by a physical robot, which is designed to induce a node addition correction.

A. Simulation Experiments

The simulation environment consists of a table, three bins, and a gripper start area. A user can move 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 Fig. 4. 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 is 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 are used to construct the FSAs using the first of the two methods outlined in Section IV-E.

1) Node Addition Experiment: For the first experiment, we constructed three models: 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 corrective 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, ITMCD Model Selection was used to select the appropriate model for each demonstration. The FSAs used for this experiment can be seen in Fig. 5(a). The Akaike information criterion (AIC) scores for each model and demonstration are shown in Fig. 6(a). As can be seen, the correct model was selected for each demonstration. Specifically, ITMCD chose the simplest FSA that encodes each demonstration.

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

2) Node Modification Experiment: 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 corrective 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, ITMCD Model Selection 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 Fig. 5(b). The AIC scores for each model and demonstration are shown in Fig. 6(b). ITMCD opted to expand the sort primitive initiation classifier (LR) in each case since the increase in likelihood achieved when adding a new primitive was not enough to

counterbalance the increase in parameters.

3) Edge Addition Experiment: For the third experiment, an initial model was constructed that can sort red blocks but could only sort blue blocks if they were placed in the gripper. An expert corrective demonstration was provided that moved a blue block from the table to the blue bin and returned the gripper to the start location. Then, ITMCD Model Selection 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 Fig. 5(c). The AIC scores for each model are shown in Fig. 6(c). As can be seen, ITMCD 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.

B. Robot Experiments

Having demonstrated the ability ITMCD Model Selection to make use of corrective demonstrations appropriately in simulation for node/edge addition and node modification, next we validate the full ITMCD algorithm with an experiment on a physical robot.

1) Platform: The demonstrations were gathered on the Poli mobile manipulator through kinesthetic teaching, see Fig. 1. Poli has a pan/tilt head with a Kinect v2 for perception and a 6 degree-of-freedom (DoF) Kinova JACO arm with a Robotiq 2-finger adaptive gripper. The arm is gravity compensated to aid in gathering kinesthetic demonstrations.

2) Task: The robot learns to pour from a pitcher to a mug and return the pitcher to the table upon completion (Fig. 1). The pitcher can be in one of two conditions: lid closed or lid open. Under the closed condition, the lid must be removed before the pouring can continue. This creates a need for correction, since the robot first learns its model of the task having only seen the lid open. Thus when encountering a closed lid, there is need for a corrective demonstration. We evaluate the ability of ITMCD to successfully modify the initial model to account for this new portion of the skill.

The state space of this environment consists of the pairwise distances between the gripper $\boldsymbol{e}$, pitcher $\boldsymbol{p}$, and mug $\boldsymbol{m}$; the vertical distances between the table $\boldsymbol{t}$ and the gripper, pitcher, and mug; 3 histogram features $\boldsymbol{h} = (h_0, h_1, h_2)$ over the dot product between the pitcher normals and table normal; and the distance between the gripper fingertips k: $\boldsymbol{S} = (\|\boldsymbol{e}-\boldsymbol{p}\|, \|\boldsymbol{e}-\boldsymbol{m}\|, \|\boldsymbol{p}-\boldsymbol{m}\|, \|\boldsymbol{t}-\boldsymbol{g}\|_z, \|\boldsymbol{t}-\boldsymbol{p}\|_z, \|\boldsymbol{t}-\boldsymbol{m}\|_z, \boldsymbol{h}, k)$. The action at each time step is the gripper velocity with respect to both Cartesian coordinates and the fingertip distance: $\boldsymbol{A} = (v_x^g, v_y^g, v_z^g, v_k^g)$.

3) Data Collection: We collected a set of 5 demonstrations of the full pouring task under the closed lid condition. From this we created a set of 5 *initial* demonstrations of the scenario where the lid is already open, by manually trimming each demonstration to start after the lid has been opened. Then each of the 5 full demonstrations can be considered *corrective* demonstrations to a model learned from the trimmed demonstrations that have never seen the closed lid condition. These set of demonstrations were used to construct 25 datasets by selecting sets of 4 *initial* demonstrations and 4 *corrective* demonstrations.

4) Results: For each of the 25 datasets, the *initial* trajectories with the already open lid were used to construct an FSA, using the second of the two methods outlined in IV-E. The *corrective* closed lid trajectories are then passed as input to ITMCD. All 25 experimental conditions resulted in node addition. The resultant FSA models fall into 3 categories:

- Correct: chain of new primitives for lid removal (4/25)
- Partially correct: correct update if one sequence is removed during graph construction (9/25)
- Incorrect: loops in graph due to primitive confusion/repetition (12/25)

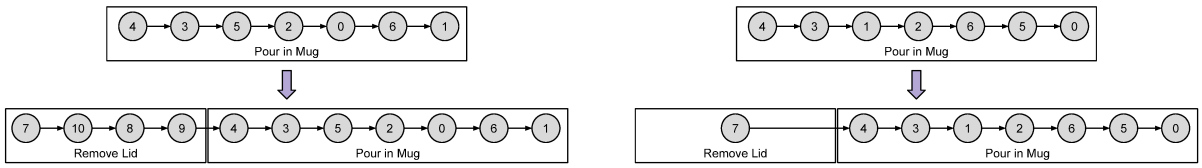
Some examples of correct and incorrect updated FSAs can be seen in Fig. 7. Though the correct models always add primitives to the beginning of the chain, different numbers of nodes are added.

Upon inspecting the demonstration sets that result in correct vs. partial vs. incorrect updates, we find a pattern. Recall that our *initial* demonstrations set is created by trimming the longer closed-lid *corrective* demonstrations. We find that if the *corrective* closed-lid demonstrations represent the same set of demonstrations trimmed to create the *initial* open-lid demonstrations, a successful model update is achieved 4 out of 5 times. If the *corrective* closed-lid demonstrations contain one demonstration that was not used to construct the open lid set, the resultant model either results in primitive confusion (12/21) or can be corrected if this one demonstration is removed (9/21). This suggests a bias in the model towards classifying small changes in the trajectories as part of a new primitive. Given that ITMCD achieves good performance if the modeled segments of the corrective demonstrations comply with the current task model, its overall performance could be improved by increasing the generalizability of the learned primitives. Explicitly modeling the execution trajectories over time, as opposed to single time step updates encoded in the dynamics, could be one way to achieve this.

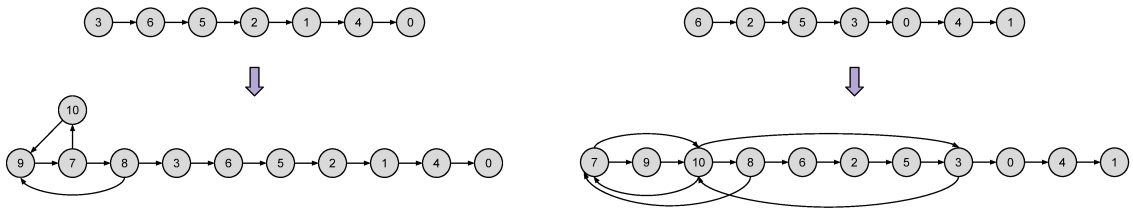
VI. Conclusion

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 the Incremental Task Modification via Corrective Demonstrations (ITMCD) algorithm that discovers such model updates through iterative constrained search and selection. Given corrective demonstrations, model updates are found that make specified changes to the task model, after which the best among these updates is automatically selected for incorporation into the original task model. In this work, we evaluated ITMCD Model Selection with simulated tasks and began an investigation of the full algorithm within a real-world task designed to induce node addition.

We showed that given demonstrations of un-modeled behavior, ITMCD Model Selection 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,



(a) Example of Correct FSA Updates



(b) Example of Incorrect FSA Updates

Fig. 7. ITMCD was evaluated on a real-world pouring task. When the corrective demos represent the same set of demos trimmed to create the initial open-lid demos (a), a correct model update is found 4 of 5 times. When this condition is not met (b), incorrect model updates are made due to primitive confusion. The numbers in the nodes above are the primitive indices assigned by the ITMCD. They are included here to highlight the preservation of the previously learned structure.

our experiments indicate that STARHMM representations of FSA task models can be used to perform this model selection. When evaluating ITMCD on a real-world task, we found that it correctly adds primitives to account for new steps in the task when the modeled segments of the corrective demonstrations closely follow the learned primitives in the current task model. These results suggest that a useful avenue for investigation is improving the primitive representation for greater generalizability, as this would serve to increase ITMCD's overall performance. Future work will seek to address this issue as well as investigate learning within real-world tasks that explicitly require node modification and edge addition.

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. This research is supported in part by the National Science Foundation (IIS-1638107, IIS-1724157) and the Office of Naval Research (#N000141410003).

References

[1] S. Chernova and A. L. Thomaz, "Robot learning from human teachers," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 8, no. 3, pp. 1–121, 2014.

[2] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.

[3] B. Akgun, M. Cakmak, K. Jiang, and A. L. Thomaz, "Keyframe-based learning from demonstration," *International Journal of Social Robotics*, vol. 4, no. 4, pp. 343–355, 2012.

[4] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal, "Learning and generalization of motor skills by learning from demonstration," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, 2009, pp. 763–768.

[5] P. Pastor, M. Kalakrishnan, L. Righetti, and S. Schaal, "Towards associative skill memories," in *Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on*. IEEE, 2012, pp. 309–315.

[6] S. Manschitz, J. Kober, M. Gienger, and J. Peters, "Learning movement primitive attractor goals and sequential skills from kinesthetic demonstrations," *Robotics and Autonomous Systems*, vol. 74, Part A, pp. 97 – 107, 2015.

[7] D. Kappler, P. Pastor, M. Kalakrishnan, M. Wüthrich, and S. Schaal, "Data-Driven Online Decision Making for Autonomous Manipulation," in *Robotics: Science and Systems*, 2015.

[8] G. Konidaris, S. Kuindersma, R. Grupen, and A. Barto, "CST: Constructing Skill Trees by Demonstration," *Doctoral Dissertations, University of Massachuts, Ahmrest*, 2011.

[9] G. Konidaris, S. Kuindersma, R. A. Grupen, and A. G. Barto, "Autonomous Skill Acquisition on a Mobile Manipulator," in *AAAI*, 2011.

[10] S. Niekum, S. Osentoski, G. Konidaris, and A. G. Barto, "Learning and generalization of complex tasks from unstructured demonstrations," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 5239–5246.

[11] S. Niekum, S. Chitta, A. G. Barto, B. Marthi, and S. Osentoski, "Incremental Semantically Grounded Learning from Demonstration," in *Robotics: Science and Systems*, vol. 9, 2013.

[12] L. Riano and T. M. McGinnity, "Automatically composing and parameterizing skills by evolving finite state automata," *Robotics and Autonomous Systems*, vol. 60, no. 4, pp. 639–650, 2012.

[13] O. Kroemer, C. Daniel, G. Neumann, H. Van Hoof, and J. Peters, "Towards learning hierarchical skills for multi-phase manipulation tasks," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1503–1510.

[14] M. Cakmak and A. L. Thomaz, "Designing robot learners that ask good questions," in *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*. ACM, 2012, pp. 17–24.

[15] E. L. Sauser, B. D. Argall, G. Metta, and A. G. Billard, "Iterative learning of grasp adaptation through human corrections," *Robotics and Autonomous Systems*, vol. 60, no. 1, pp. 55–71, 2012.

[16] S. Chernova and M. Veloso, "Interactive policy learning through confidence-based autonomy," *Journal of Artificial Intelligence Research*, vol. 34, no. 1, p. 1, 2009.

[17] A. Jain, B. Wojcik, T. Joachims, and A. Saxena, "Learning trajectory preferences for manipulators via iterative improvement," in *Advances in neural information processing systems*, 2013, pp. 575–583.

[18] C. Elkan and K. Noto, "Learning classifiers from only positive and unlabeled data," in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2008, pp. 213–220.