Logical Hierarchical Hidden Markov Models For Modeling User Activities

Sriraam Natarajan*, Hung H.Bui+, Prasad Tadepalli*, Kristian Kersting#, Weng-Keen Wong*

* School of EECS, Oregon State University, Corvallis USA ***
+ AI Center, SRI International
# Fraunhofer IAIS, Schloss Birlinghoven, Germany†

Abstract. Hidden Markov Models (HMM) have been successfully used in applications such as speech recognition, activity recognition, bioinformatics etc. There have been previous attempts such as Hierarchical HMMs and Abstract HMMs to elegantly extend HMMs at multiple levels of temporal abstraction (for example to represent the user's activities). Similarly, there has been previous work such as Logical HMMs on extending HMMs to domains with relational structure. In this work we develop a representation that naturally combines the power of both relational and hierarchical models in the form of Logical Hierarchical Hidden Markov Models (LoHiHMMs). LoHiHMMs inherit the compactness of representation from Logical HMMs and the tractability of inference from Hierarchical HMMs. We outline two inference algorithms: one based on grounding the LoHiHMM to a propositional HMM and the other based on particle filtering adapted for this setting. We present the results of our experiments with the model in two simulated domains.

1 Introduction

Activity recognition is a problem that has long been the focus of researchers and has a wide range of applications from surveillance[8] to intelligent interfaces[11] to assisting the elderly[1]. Accordingly, several kinds of approaches ranging from domain-specific hand-coded solutions to the more general Hidden Markov Models(HMM) have been proposed for such activity recognition problems. Hidden Markov Models and their several extensions are among the most popular methods for activity recognition. The main advantage of HMMs and their extensions lies in the fact that they define a clear probabilistic semantics for the problem. Also, efficient inference algorithms have been proposed for HMMs making them very attractive for this problem. The different extensions of HMM like the Hierarchical HMM (HHMM) [5] and abstract HMMs (AHMM) [3] are being widely used in activity recognition for a variety of applications.

The main drawback of HMMs is that they do not take into account the complete structure of the problem. The objects in the domain may be related by specific relationships and these relationships govern the action that the user performs in the current state. For example, a user is more likely to send a paper

*** Sriraam is currently at University of Wisconsin, Madison
† Kristian was at CSAIL, MIT when the work was performed

that he is writing to his co-authors and not to random email addresses. Also, since the HMMs are inherently propositional, they do not allow for generalization among the objects of the domain. For instance, the models cannot be shared among multiple users of the desktop, as a separate HMM needs to be constructed for every user in a propositional setting. Also in several cases, the user might decompose his goal of submitting a proposal or writing a paper by a similar methodology of running experiments, writing the paper, writing the abstract, sending it to one's co-authors etc.

The goal of this work is to extend the HMM with a logical model that allows for generalization of the objects in the domain and a hierarchical model that allows for richer structure of the user's tasks. The use of logical models will allow us to specify the models at an abstract level that can later be instantiated with specific instances and to perform inference on them. Also, it allows parameter sharing between the objects of the same type thus requiring a smaller number of examples for learning. The hierarchical structure will enable us to elegantly represent the user's goal-subgoal decomposition and allows for efficient inference. Kersting et al.[7] earlier introduced a logical extension to HMMs. In this paper, we extend their work in 2 ways: allowing conditional transitions and more importantly incorporating hierarchies in the logical models.

Our first contribution is to introduce the Logical Hierarchical HMMs (LoHiHMMs) and outline their syntax and semantics. In many real-world applications, the user chooses his action based on some conditions in the environment. For example, a user who goes shopping might prefer the nearest store to shop from. If the product that he wants to buy is not available in that store, he might go to another one. If it was available, he might buy the product and return home. The availability of the product can be observed by looking at the inventory. But, it is not possible to observe the mental state of the user. Hence, some parts of the state space are completely observed while some others are not. In our model, we consider the current state as having two components: an observable component (called the *world* state) and an unobservable component (called the *user state*). Note that the names of world and user are specific to our applications but they mean the *observable* and the *unobservable* parts of the state space. One of our critical assumptions is that the world state is completely observable. This would make it possible to perform tractable inference (as we show later in the paper) in large domains. In our model, we naturally model such conditional transitions that take place based on the observable part of the state space.

Logical models have generally been unrolled to their ground formulations and inference was performed using these ground models. We present an algorithm for unrolling the LoHiHMM to the corresponding ground HMM which will make it possible to perform exact inference in the model. But, in very large domains, the state space of the ground HMM could be prohibitively large and can make the inference hard. HMM inference is quadratic in the number of states and a very large state space can make the inference impractical if not infeasible. Hence, in this work, we avoid complete unrolling of the LoHiHMM. Our second major contribution is to adapt particle filtering to this logical setting. The filter avoids

considering all the possible states by exploiting the conditions in the transitions and restricting the set of possible next states. We compare the performances of the exact inference on the ground HMM and the particle filter on 2 simulated domains: a grid world domain where the user has to navigate the grid to achieve his tasks and a kitchen domain where the user follows some recipes to cook his dishes. The results demonstrate that the particle filter performs comparably to that of the exact HMM with much less overhead.

The rest of the paper is organized as follows: the next section provides the background for the work. Section 3 first presents the LoHMM, its syntax and semantics and then extends them to the Logical Hierarchical HMMs. Section 4 outlines the inference algorithm based on particle filtering and the algorithm for unrolling it to a ground HMM. Section 5 presents the experimental results on the domains. The final section concludes the paper by reviewing the related work and outlining some areas of future research.

2 Background

In this section, we briefly review the background work namely, HMMs, Hierarchical HMMs, Particle Filtering and the previous work on Logical HMMs.

Hidden Markov models (HMMs) are used to model systems that follow a Markov process which is essentially a process with no memory. In a Markov process, the next state of the system is independent of the past states given its current state. In HMMs, there are 2 kinds of variables: a *state* variable that follows a Markov chain and an *observation* variable whose value is generated by the current state. As the name implies, in an HMM the states are assumed to be unobserved(hidden). Hence, one of the important tasks is to infer the distribution of the current state conditioned on the values of the observations (known as filtering). Formally, an HMM is defined using the 5-tuple $\langle S, \Pi, O, \Omega, I \rangle$, where S is a set of states, $\Pi(s'|s)$ is the next-state distribution, O is a set of observations, $\Omega(o|s)$ is the probability of observing o when in state s and I is the initial state distribution. The HMM starts in one of the states s chosen according to $I(s)$, at every step transitions to next states according to Π, and emits observation o with probability $\Omega(o|s)$.

It is clear that while considering the problem of activity recognition, the goal is to infer the distribution over the states given the current set of observations $p(s|o_{1:t})$. Hence the activity recognition problem is posed as performing inference in an HMM. Although a HMM is a very good choice for the problem, there are applications where the number of states can be arbitrarily large. It is also difficult to model situations where some states take more than one time-step to be executed. For instance, editing a particular document might take several time-steps and it is not possible to fix the number of time-steps in advance. In addition, the user's tasks could have a well-defined hierarchical structure that can be exploited if captured explicitly by the model. Though these can be captured by a HMM, the structure in the problem cannot be exploited efficiently.

Hierarchical HMMs[5] extend HMMs to include a hierarchical structure for the states. Each state in a HHMM is a self-contained probabilistic model. One

way to understand a Hierarchical HMM is to think of each state of the HHMM being an HHMM in itself. So when the HHMM transitions to the current state, the sub-HHMM is activated which in turn activates a HHMM at the lower level. This would mean that at each state, the HHMM will emit a sequence of observation symbols rather than a single symbol. The key difference to a normal HMM is the notion of *end* states. Each sub-HMM has a set of end states that can terminate the HMM at the current level and return the control to the calling state of the parent HMM.

Any HHMM can be converted to a HMM[10]. The state of the HMMs correspond to the state stack of the HHMMs($S^{1:D}$), where D is the number of levels of the HHMM. If the HHMM structure is a tree, it would imply that there are no repetitive substructures. The corresponding HMM then contains one state for every combination of the parent and child states. If the HHMM has repeated substructures, they have to be duplicated in the HMM resulting in a large model. We refer to [2],[10] and [5] for more details on converting a HHMM to a HMM.

Since in large HHMMs (or even large HMMs), performing exact inference might be impractical, sampling methods became popular. The key innovation in these methods is to focus the set of samples on high probability regions of the state space i.e., generate more samples in high posterior regions. Among the sampling methods, *particle filtering*[4] methods are very popular. The general approach to particle filtering works as follows: Samples are generated according to the prior distribution and propagated to the next step according to the transition distribution. The samples are then re-weighed according to the observed evidence probabilities and new samples are generated. We present the generating distributions later in the paper when we discuss the particle filter for the logical setting.

Logical HMMs: Given that the HMMs are inherently propositional, Kersting et al. introduced Logical HMMs [7] to combine ideas from SRL[6] and dynamic models. In their framework, the states of the HMM are abstract states rather than propositional states. An abstract state consists of a predicate name and a set of parameters that can be instantiated with constant ground values. The transition distribution in the logical HMM now consists of two kinds of distributions: an abstract transition distribution that specifies the probability of the next abstract state given the current abstract state and a selection distribution that specifies the probability of the instantiation of the parameters for the current state. In their work, the observations are generated from state transitions and the selection distribution is specified using a Naive Bayes function.

3 Logical Hierarchical Hidden Markov Models

In this section, we present our formalism of LoHiHMMs. Since this formalism extends the HMM in two dimensions, we present the logical extension (LoHMMs) as a first-step and later extend it to include hierarchies. We refer to our logical extension as LoHMMs and that of Kersting et al. as Logical HMMs. We contrast our formalism to theirs after we introduce our model.

3.1 LoHMMs

Consider the HMM presented in Figure 1. In this example, a user receives a document to edit through an email, edits the document and sends it back to the sender. The semantics is that when the conditions in the arcs are satisfied the HMM transitions to the next state with the probability indicated by the number on the arcs. In this section, we first define the states and transitions of this HMM and then formally define LoHMM. In our model, the state in the

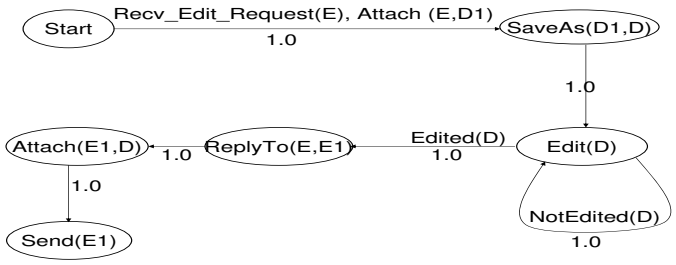

Fig. 1. A Logical Hidden Markov Model for editing a document. The logical conditions on the arcs indicate the conditions under which the transitions take place. The numbers on the arcs indicate the probability of the transition.

model consists of two components, the state of the user and the state of the environment. The state of the user is assumed to be a single ground predicate that indicates the activity of the user. The user is restricted to be in only one state at any given instant of time. The state of the environment could be a conjunction of several predicates. An example of user state is $edit(D)$, where *edit* is the logical predicate and D is the parameter (variable) associated with this state. The environment state could include information about the current time, the deadlines, the set of projects that the user is involved with or as in Figure 1 whether a request for edit has been received etc.

State: A state consists of two components: the *user state* which is a predicate that represents the current activity of the user and the *environment state* which is a set of predicates describing the environment. We aim to capture transitions between user states conditioned on some states of the environment. This will enable us to model the user's activity conditioned on the context of the execution. For example, if the document that the user is currently editing is a long document, the user might prefer to print the document, while if it is a short one, the user might just read it off the monitor. Observing the length of the document will enable us to predict the user's next action. Though there are several ways of modeling the conditions in the transitions between the states, we use a model that is similar to decision lists or the case statements in programming languages.

Table 1. Structure of an Logical transition in our model. The current user state is $A(X, Y)$ and transitions to one of the successor states based on the condition and logical transition probability.

$$A(X,Y) \mapsto \begin{cases} \text{if } C_1^1(X,Y) \wedge C_2^1(Y,Z)\text{... then} & \begin{cases} p_1^1 : & B_1^1(X,Z) \\ p_2^1 : & B_2^1(Y,Z) \end{cases} \\ \text{else if } C_1^2(X,Y) \wedge C_2^2(X,Y,Z)\text{... then} & \begin{cases} p_1^2 : & B_1^2(X,Z) \\ p_2^2 : & B_2^2(Y,Z) \end{cases} \\ else & ... \end{cases}$$

Associated with each state predicate we can define a set of observation predicates and an associated observation distribution. The observations in our model are similar to the state predicates. The observation consists of a predicate name and a set of variables. The observation may or may not consist of the same variables that are present in the state predicate.

The transitions in our model are called *logical transitions* and are of the form presented in Table 1. The transition occurs between the current state of the user (called as *source* user state) and the next state of the user (called as *target* user state) conditioned on some of the predicates of the environment. As shown in the figure, there could be several logical test predicates that can be conjunctions of several predicates (shown as C in the figure) based on which transition could take place.

The most important constraint is that exactly one branch of the transition will be always taken for the given state. Once a particular branch, say k, evaluates to true and the corresponding branch is taken, the target atom is chosen according to the transition distribution (called ''logical transition distribution") $\boldsymbol{p^k}$ for the current branch k. $\boldsymbol{p^k}$ is a well-defined distribution i.e, $\sum_{i \in S} p_i^k = 1$, where S is the set of target abstract states for the given logical transition. In the statement presented in Table 1 for instance, $p_1^1 + p_2^1 = 1$. An example transition is presented in Table 2. In this example, the user downloads a document P and if it is a long paper, the user prints the document with a probability 0.9 or reads it off the webpage with a probability 0.1. If not, he reads from the webpage.

The branches can be understood as similar to decision lists in programming languages. Given the instantiations of the current user state, the branch that first evaluates to true is taken. Once a branch is chosen, the logical transition distribution is used to choose the next user state predicate. The last branch has the key word *else* and has no conditions. It is a default one that will be taken if the other branch conditions fail to evaluate to true (for example see Table 2). This is to ensure that the conditions in at least one branch would evaluate to true and hence one branch will always be taken for the given user state.

Logical Transition: A logical transition consists of the current user state S as the source and a set of logical predicates of the environment. Corresponding to each predicate is a set of target user states and a corresponding set of probability

Table 2. Example of a Logical transition. The user prints the document if it is a long document or reads it off the webpage.

$$Download(P) \mapsto \begin{cases} \text{if } LongPaper(P) \wedge URL(Z,P) \text{ then} & \begin{cases} 0.9: & Print(P) \\ 0.1: & Read(P,Z) \end{cases} \\ \text{else if } URL(Z,P) & \begin{cases} 1.0: & Read(P,Z) \end{cases} \\ \text{else} & \begin{cases} 1.0: & DoNothing \end{cases} \end{cases}$$

distributions called the logical transition distributions. For each of the target user states, there is an associated instantiation distribution ($\boldsymbol{\mu}$) that specifies the probability of the values of the variables associated with the target state. We discuss the instantiation distribution later.

One potential issue is that there could exist several transitions that can be matched with the current instance. For example, ignoring the conditions there could be 2 different abstract transitions defined as follows:

$$A(X,Y) \longrightarrow B(X,Z)$$
$$A(x,Y) \longrightarrow C(Y,Z)$$

The second transition is more specific when compared to the first transition. If we use a substitution $\{\theta = X/x, Y/y\}$, we could match the two rules and there is a question of which transition to choose. In order to avoid multiple matching rules, we require that these 2 rules are ordered and combined to a single decision list as follows

$$A(X,Y) \begin{cases} \mathrm{I}f & (X=x) \quad 1.0 : C(Y,Z) \\ else & 1.0 : B(X,Z) \end{cases}$$

The idea is to encode the more specific instances as well in the transition. This is natural in our formalism as we always assume that only one branch can be true for any ground instantiation. This means that no two different logical transitions can exist with the same predicate name as its source.

Now, we define a generalized LoHMM to consist of set of states S, a set of observations O and a set of logical transitions Δ. This definition of the LoHMM is too general for many problems. If we assume that the logical predicates are unobserved along with the user states, the problem of user activity recognition includes performing inference over the environment states as well. Hence, we consider a restricted model in which we assume that the environment states are completely observed and the user state is assumed to be unobserved. This assumption is not unreasonable in many domains such as a desktop domain, video surveillance etc where the effects of the user's actions in the environment are completely observable while the user's mental states are not.

Definition 1 (LoHMM). *A LoHMM consists of*

- *a set of states S which has two components: the user state S_u that is a single predicate and is unobserved and the environment state S_w that can possibly consist of many predicates and is completely observed*[1]
- *a set of observations O and the associated observation distributions*
- *a set of logical transitions Δ.*

Note that the assumption of the state being composed of the observable environment states and the unobserved user state predicate makes the inference process easier. This will enable us to ignore the changes to the environment predicates and just focus on the changes to the user state. The key thing to note here is that the logical conditions are the observable part of the state space. Hence given a current user state, it would be easy to evaluate the different logical predicates conditioned on the current user state. The LoHMM can be understood as defining a distribution over the set of ground predicates (interpretations). The ground interpretation consists of two parts: the user activity interpretation that is not observed and the environment interpretations that are observed and are fixed and do not change with time. The transitions can be understood as a stochastic mapping between two ground interpretations that depends on certain parts of the state.

Global Variables: There could be some variables in the LoHMM whose value should not change as the HMM evolves. For instance, consider the example in Figure 1. The user, after editing the document, has to reply to the email in which he received the request to edit. This means that the value of the variable E once set to the value of the email that has the request should not be changed. In our model, some of the variables are declared as global and their values do not change over time.

Substitution: Substitution in LoHMMs is defined exactly as in first-order logic. A substitution θ is defined as assignment of values to all the variables in the LoHMM. The set of variables is the union of all the variables in all the predicates of all the user states and the environment states.

Ground HMM: A ground HMM is a HMM constructed by substituting the values for all the variables of a LoHMM. We refer to the user state that has been substituted for its variables as an instantiated user state. The state-space of this ground HMM consists of two components: the instantiated user states and the global list. The transition distribution specifies a distribution over the next ground user states and the values of the global variables.

Theorem 1 (Existence of a ground HMM). *A ground HMM can be constructed for every LoHMM.*

The proof is trivial and follows from the theorem in Kersting et al [7] and we omit the proof for brevity. The motivation behind stating the existence of a HMM

[1] The names user state and environment state are chosen for the activity recognition purposes only. They mean the unobservable and observable part of the state space in general.

is that, since a HMM defines a unique distribution over the trajectories, it follows that the LoHMM also defines a unique distribution over the trajectories. This ensures that there is a well-defined model-theoretic semantics for the LoHMM. In fact, HMMs are a special case of the LoHMMs with the states having an arity of zero, no conditional branches and no instantiation probability. Hence LoHMMs generalize HMMs.

We now present the relationship to the Logical HMMs of Kersting et.al [7]. In their model, there was an abstract transition distribution $\boldsymbol{p}$, which chooses the target predicate to transition to. In our model, first the logical conditions are evaluated, the corresponding branch is taken and then the target predicate is chosen according to the distribution $\boldsymbol{p^k}$. Their model does not allow for conditional branches. Thus their model can be considered as a special case of our LoHMMs where the default ($else$) branch is always taken. Recall that since they restrict their states to be predicates as well, the conditions cannot be captured in their Logical HMMs. Yet another difference is the presence of the global variables in our model. In their model, a variable has to be present in all the states for it to retain its value. Declaring a variable as global is useful in particular for the hierarchical setting as we explain in the next section.

3.2 LoHiHMMs

In many real-world situations, users solve difficult problems by decomposing them into a set of smaller ones. For example, proposal writing might involve writing the project description, preparing the budget, and then getting signatures from proper authorities. The tasks to be completed by the user have a natural hierarchical structure. To capture this kind of knowledge using LoHMMs, the state predicate should include the user's current task at all the levels of the hierarchy. This is not very elegant and it would be difficult for the domain expert to specify this kind of LoHMM; also, the inference process can be very difficult due to the explosive state space. A representation that could capture such a hierarchical structure cleanly can be used to perform efficient inference. In this section we introduce the Logical Hierarchical HMMs.

Recall that in a LoHMM, there are logical (horizontal) transitions between user states while in the Logical Hierarchical HMM (LoHiHMM) in addition to the horizontal transitions, there are vertical (task-subtask) transitions between parent and child tasks[2]. It should be mentioned that LoHiHMMs impose the hierarchical structure on the user's states and not on the environment state predicates as we are interested in modeling the user's activities and not how the environment evolves.

Instantiation Probability and Global Variables: The distribution that is used for grounding the state predicates to their ground values is called the *Instantiation Distribution* and denoted by μ. The constraints in the abstract transition serve as hard constraints for this distribution. In addition, in the hierarchical setting, the variable instantiation of the parent state should stay the same in all transitions made by its child states, i.e., the variables of the

[2] We refer to the states of the LoHiHMM as tasks and subtasks

Table 3. The syntax of the selection transition statements. The keywords *parent* and *child* denote the parent and child task respectively. The other parts of the statements are similar to LoHMM

$$parent:\ A(X,Y) \mapsto \begin{cases} \text{if } C_1^1(X,Y) \wedge C_2^1(X,Y,Z)... \text{ then} & \begin{cases} sp_1^1 : child: & B(X,Z) \\ sp_2^1 : child: & C(Y,Z) \end{cases} \\ ... & \\ else & ... \end{cases}$$

parent state should be"global". Their values cannot change once instantiated until that state finishes its execution. The global list thus includes variables that are declared global along with the values of the parent variables.

Selection Probability: The selection probability specifies the probability distribution over the child tasks given the parent tasks. We call the transition between a parent task and a child task a *selection transition* and the syntax is presented in Table 3.

The syntax is similar to the LoHMM syntax except that we now use special keywords namely, *parent* and *child* to identify the respective tasks. The rule that is presented in Table 3 is interpreted as: *"If the current user task is A(X,Y), then it chooses one of the possible next child tasks by evaluating the branch predicates (shown as C_i^j in the rule) and then chooses the child user task (example B or C) based on the selection distribution"*. This rule is similar to the transition rule in LoHMMs except that in this case, it defines a task-subtask transition of a HMM. The branch predicates are handled in a manner similar to a decision list such that the first satisfied branch is always taken. The selection probability is well defined, i.e., the sum of the probabilities of the different child abstract tasks for a given branch is 1 ($\sum_i sp_i^j = 1$). Note the presence of a default branch that is chosen if all the other branch conditions fail to evaluate to true.

Transition Probability: The transitions are similar to the LoHMMs with one difference: the transitions must include the task of the parent. Hence, the horizontal transition is conditioned on the instantiated value of the parent tasks. The syntax of the horizontal transition is presented in Table 4. The statement is similar to the ones presented for the LoHMMs except that these statements include the parent tasks (shown using the keyword *pa*). These statements can be interpreted as follows: *"If the current parent task is A(X,Y) and the current child task is B(X,Z), then it chooses one of the possible logical transitions by evaluating the branch predicates (shown as C_i^j in the rule) and chooses the abstract child task based on the transition distribution"*. For instance, the next child could be of the form $C(x,y,z,w)$ with probability $p_1^1.\mu(w)$, where p_1^1 is the abstract transition probability for $C(X,Y,Z,W)$ and $\mu(w)$ is the instantiation probability for w. Note that the variables in the parent (X,Y) will be "global", since its binding will stay the same until the current parent terminates.

Table 4. The syntax of the horizontal transition statements. The keywords *pa*, *ch* and *nch* denote the parent the current child and the next child abstract tasks respectively. The other parts of the statements are similar to LoHMM

$$pa:\ A(X,Y);\ ch:\ B(X,Z) \mapsto \begin{cases} \text{if } C_1^1(X,Y,Z) \wedge C_2^1(X,Y,Z)... \text{ then} & \begin{cases} p_1^1 : nch : C(X,Y,Z,W) \\ p_2^1 : nch : D(X,Y,Z,W) \end{cases} \\ ... & \\ else & ... \end{cases}$$

As with the selection transition of LoHiHMM, the set of branch predicates for a particular parent-child combination transition rule is part of a decision-list so that the first branch that evaluates to true is taken. The conditions on the transitions could also serve as hard constraints on the instantiation distributions. Similar to the LoHMMs, the transition distribution is well defined i.e., $\sum_i p_i^j = 1$.

Ending Probability: If the LoHiHMM at a lower level terminates at some task, the control would return to the parent task. For example, it is possible that the user might take a break from running the experiments and instead work on the abstract. It is possible that the current task might terminate in more than one task. Hence there is a necessity to define a distribution over the possible end tasks at a particular level. We specify the ending probability as a function β from the set of ground tasks (at any given level) to $[0, 1]$.

Execution semantics: The execution semantics of the LoHiHMM is as follows: When control reaches the current user state, it chooses a child state to transition to based on the selection distribution and the instantiation distribution. Once all the lower level HHMMs terminate, the control reaches back to the current state and it chooses a horizontal transition as with the case of LoHMMs. The main intuition is that at each level we have a LoHiHMM and the selection distribution chooses the LoHiHMM to execute. Each vertical transition (selection of a child) also has logical conditions that need to be evaluated to choose the child. LoHiHMMs are to LoHMMs what HHMMs are to HMMs. They can be viewed as either incorporating hierarchies to LoHMMs or adding logical models to Hierarchical HMMs. Similar to the LoHMMs, it is possible to state and prove the existence of a ground Hierarchical HMM for every LoHiHMM.

It must be mentioned that since a HHMM exists for every LoHiHMM and since a HMM exists for every HHMM, there exists a HMM for every LoHiHMM but with a prohibitively large state space. LoHiHMM provides 2 further significant advantages: first, it is easier for the domain expert to elucidate the knowledge as it is difficult to encode all the relationships and the task hierarchy into the CPTs of the HMM. Second, learning can be easier since the parameters are shared in the LoHiHMM.

4 Inference

In this section we present two methods for inference in these models. The first is to unroll the LoHiHMM to a ground HMM and use a sparse matrix package. Our second method avoids explicit unrolling and adapts particle filtering for the logical setting. Although we present the algorithm for unrolling to a ground HMM, we do not do that automatically in our experiments and instead use an equivalent hand-crafted HMM for comparison.

4.1 Particle Filter with Logical querying

Particle Filter[4] samples the next state based on the evidence and computes the weight of the samples. The state at time t is denoted by x_t and the observation at time step t is denoted by y_t. Samples are drawn according to the optimal proposal distribution $P(x_t|x_{t-1}^i, y_t)$ given that i is the current level of the goal stack where,

$$P(x_t|x_{t-1}^i, y_t) = \frac{P(y_t|x_t)P(x_t|x_{t-1})}{\sum_{x_t} P(y_t|x_t)P(x_t|x_{t-1})} \tag{1}$$

and the weight w_t is given by,

$$w_t \propto P(y_t|x_{t-1}^i) = \sum_{x_t} P(y_t|x_t)P(x_t|x_{t-1})$$

The states are then re-sampled based on the weights of the samples. In this section, we modify the algorithm for the LoHiHMMs. The main bottleneck is computing the summation over all the next states (x_t)[3] in the equation 1. In a logical model, the number of possible ground states can be very large. Thus it might not even be possible to enumerate all the states for the summation to be computed. For instance, in a desktop there could be a very large number of files and emails. It is impractical to enumerate all possible combinations of files that can be attached to an email. Hence we propose to use a method that avoids enumeration of the complete state space. In our model, given the value of the previous sample and the values of the elements in the global list, a query answering procedure would return the set of possible next states. The intuition is that the conditions in the abstract transitions would greatly reduce the set of possible next states.

Given a particular ground state and the values of the global list, the selection procedure would first determine the set of logical transitions that have the current user state predicates as the source and obtain the set of the possible next user state predicates (task-subtask combinations) based on the logical conditions. For each of the next predicates, possible instantiations are considered using the instantiation distribution. The probability of the next ground state is the product of the selection probability and the instantiation probability. In some cases, if we are only interested in the current user state predicate (for instance,

[3] In the case of LoHiHMMs, x_t corresponds to the user's goal stack

whether the user is composing an email or editing a document), we ignore the instantiation distribution and sample only from the selection distribution.

For instance, consider the example presented in Figure 1. Let us assume that the user just completed editing the document say $d1$. So when the particle filter queries for possible next states, the condition $edited(d1)$ is satisfied. Since the email E is in the global list, the selection procedure returns the next state as replying to email E. This greatly reduces the computation that is otherwise needed to sum over all the states.

Our inference procedure can be understood as a lazy evaluation procedure where the HMM is constructed on the fly based on the values that have been assigned to the variables and the conditions that are satisfied. The main goal of this procedure is to avoid the explicit construction of a ground HMM. This is because as we have pointed out earlier, the number of objects in the real-world is very large which in turn leads to huge transition and observation matrices. But considering the objects lazily based on the current state and the values of the variables, we are able to perform effective inference in real time.

4.2 Constructing a ground HHMM

It is clear that the LoHiHMMs extend the LoHMMs similar to the way in which Hierarchical HMMs (HHMMs) extend HMMs. Hence it is conceivable that a LoHiHMM can be unrolled into a ground HHMM. The basic idea of grounding to a HHMM is as follows: once the current abstract state at level d has been instantiated with the corresponding values subject to the constraints, it chooses a child abstract state at level $d+1$ based on the logical conditions and the selection distribution. The child state then chooses its instantiation based on the instantiation distribution and the global list of variables. It then chooses its child state at depth $d+2$ recursively. If the current state at d is a primitive state, it chooses the next state at the subsequent time-step based on the conditions and transition distribution. If the child state terminates, the control returns to the parent state, which makes an abstract transition to the next state at the same level based on the conditions and the transition distribution.

The key thing to note is that the process is very similar to the execution of a HHMM. Since at any time a particular condition can only be true for any particular transition, we obtain a HHMM at the ground level. We now proceed to define this ground HHMM and the process of unrolling. The states of the ground HHMM at level d correspond to the possible ground instantiations of the atoms of all the possible predicates at level d. Let us consider the selection of a ground child state given the parent state. Assume that the current abstract state is $A(X,Y)$ and the selection transition is defined by the rule in Table 3. Then, for all substitutions of X,Y,Z, i.e, $\forall\theta = X/x, Y/y, Z/z$

$$If\ BP_1(\theta) = True, \begin{cases} \mathrm{P}(B\theta \mid A\theta) = sp_1^1 \cdot \mu(\theta) \\ \mathrm{P}(C\theta \mid A\theta) = sp_2^1 \cdot \mu(\theta) \end{cases}$$

$$else...$$

Hence the idea is to consider all the substitutions of the current parent states and evaluate the branch conditions and choose the next child ground state based on the selection and instantiation distributions. The above step takes place if the state $A(X, Y)$ is instantiated at the previous time-step and it has to choose the ground child state in the current time-step.

If the current state is $B(X, Y)$ and the child state finishes executing, it will make an abstract transition to the next state at the same level. For example, consider the transition in Table 4 and assume that the current state at level d is $B(x, z)$ and then child tasks at level $d' > d$ are completed. $B(x, z)$ then makes an abstract transition. For all substitutions of X,Y,Z,W $i.e, \forall\theta = X/x, Y/y, Z/zW/w$

$$If\ BP_1(\theta) = True, \begin{cases} \mathrm{P}(C\theta \mid B\theta, A\theta) = p_1^1 \cdot \mu(\theta) \\ \mathrm{P}(D\theta \mid B\theta, A\theta) = p_2^1 \cdot \mu(\theta) \end{cases}$$

$$else...$$

Here the current state considers all the substitutions, evaluates the branch condition and chooses the next state based on the abstract transition and instantiation distributions. Since it has been shown that a HMM exists for every HHMM [5], we can use any standard HMM inference with a sparse matrix representation for performing inference.

5 Experiments

In this section, we present our experiments with the particle filter and unrolled (hand-crafted) HMM in 2 simulated domains: a grid world and a cooking domain.

5.1 Doorman Domain

In this domain, the user is in a gridworld where each grid cell has 4 doors that the user has to open to navigate to the adjacent cell. The LoHiHMM for the figure is presented in Figure 2.*a*. The highest level goals of the user are to *Gather* a resource or to *Attack* an enemy. To *gather* a resource, the user has to *collect* the resource and *deposit* it at the corresponding location. Similarly, to *destroy* an enemy, the user has to kill the *dragon* and *destroy* the castle. There are different kinds of resources, namely *food* and *gold*. Each resource can be stored only in a storage of its own type (i.e, *food* is stored in *granary* and *gold* is stored in *bank*). These serve as constraints in the LoHiHMM and are shown as conditions on the arcs in the Figure 2.*a*. There are 2 locations for each of the resources and its storage. Similarly there are 2 kinds of enemy *red* and *blue*. The user has to kill the *dragon* of a particular kind and *destroy* the castle of the same kind. The actions that the user can perform are to move in 4 directions, open the 4 doors, pick up, put down and attack.

The states of the LoHiHMM are the goal-subgoal combinations of the user. The world state (observable) consists of the current square that the user is in and the current door that is open. The observations are the user actions.

The observation distribution is generated using a Boltzmann distribution on the number of actions to the goal. The results are presented in Figure 2.b. The particle filtering method is compared against the ground HMM on 50 different runs where the user chooses the top-level goal and starts acting towards achieving it. The mean reciprocal rank (mean of the inverse of the rank of the current state) was measured and presented. A MRR of 1 indicates that the correct state has always been ranked as 1 by the algorithm.

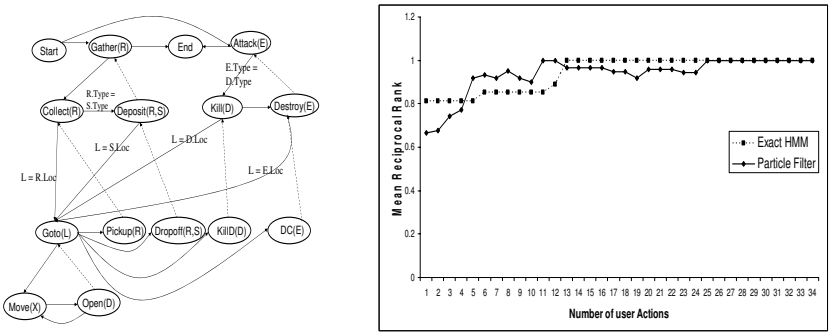

Fig. 2. (a)LoHiHMM for the gridworld domain. (b)Results. The y-axis presents the Mean Reciprocal Ratio of the correct user state

As can be seen, the grounded HMM (being an exact inference) method has a slightly better performance than the particle filter. More precisely, when a sub-goal has been achieved (or at the start of the run), particle filter can sample a few of the incorrect subgoals. Once a few observations are obtained, the particle filter performs as well as the ground HMM (in fact even better before the first sub-goal has been achieved). On seeing a few observations, it converges to the right state as evident from the slight dip in performance. On the other hand, while the performance of the exact HMM inference is marginally better, it has a large number of states. In this domain, the HMM had about 144 ground states and about 2200 observations. Thus the transition matrix is of the size 144×144 while the observation matrix is of the size 144×2200. It took about 3ms on an average to perform inference using the ground HMM and a sparse matrix representation while it was nearly 0ms using the filter. In larger domains, such as a real-time desktop assistant or video surveliance, it is likely that this state space can grow quickly and it could become infeasible for performing inference.

5.2 Kitchen Domain

The other domain is a kitchen domain where the user has to cook some dishes. The user has 2 kinds of higher-level goals: one in which he could prepare a recipe which contains a main dish and a side dish and the second in which he could use some instant food to prepare a main dish and a side dish. There are 2 kinds of main dishes and 2 kinds of side dishes that he could prepare from the recipe.

Similarly, there are 2 kinds of main dishes and 2 kinds of side dishes that he could prepare from instant food. The LoHiHMM is presented in Figure 3.*a*.

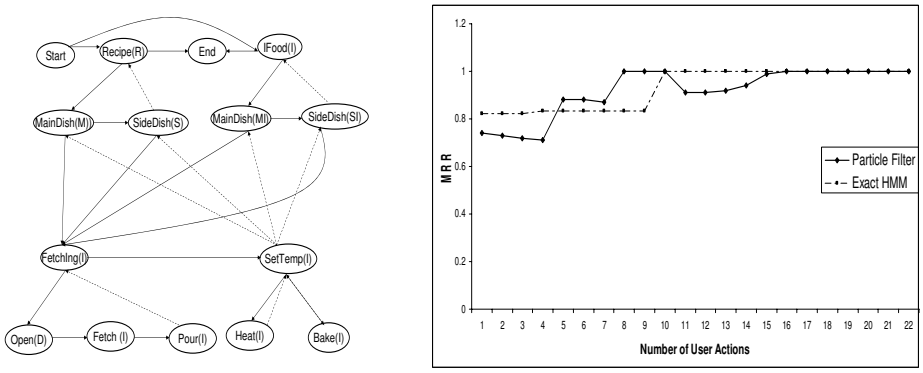

Fig. 3. (a)The kitchen LoHiHMM. (b)Results of the particle filter and the ground HMM inference

There are 2 shelves with 3 ingredients each. The shelves have doors that must be opened before fetching ingredients and only one door can be open at a time. The observable part of the state consists of the contents of the bowl, the ingredient on the table, the mixing state and temperature state of the ingredient (if it is in the bowl) and the door that is open. The user's actions are: open the doors, fetch the ingredients, pour them into the bowl, mix, heat and bake the contents of the bowl, or replace an ingredient back to the shelf.

As with the previous domain, we present the Mean Reciprocal Ratio of the correct state when using the particle filter and the exact inference using ground HMM in this domain. The 2 algorithms were executed for 50 runs where the user chooses the high-level goal randomly. In this domain, the performance of the particle filter is better than the grid world domain. As can be observed, the particle filter samples more correct states in the beginning and hence has a better performance compared to the exact inference method. But like the previous domain, once a sub-goal has been achieved, the filter samples some of the incorrect states thus having a small dip in the MRR but then recovers by using the observations to converge quickly on the true state. The ground HMM had a larger number of states(64) and observations (> 700) respectively leading to large transition and observation matrices and took about 2.5ms per inference query when compared to nearly 0ms for the particle filter.

6 Conclusion and Future Work

In this work, we motivated the need for the combination of the logical and hierarchical versions of HMMs for performing activity recognition. To this effect, we have developed and presented Logical Hierarchical HMMs. We outlined the

syntax and semantics of the conditional branching for the vertical and the horizontal transitions that depend on certain attributes of the environment. We also outlined a particle filter algorithm to perform efficient inference on these models and presented a procedure for unrolling the LoHiHMMs to HMMs. We then evaluated our algorithms on 2 simulated domains and showed that the particle filter can perform efficiently while sacrificing a small amount of accuracy.

Zettlemoyer et al. [13] consider the problem of filtering in relational HMMs and propose the idea of logical particle filter for this case. In their model, the states are conjunctions of propositions and/or functions. The observations and the actions are propositions (i.e., unparameterized). The basic idea is to construct a set of partitions (hypotheses) where each of the partitions represents a set of states such that every state in a hypothesis has the same transition probability. The hypotheses at the current time step are split into a set of mutually exclusive ones, the transition probability is then applied and the observations are used to specialize the hypothesis. The expectation over the states in a hypothesis is computed analytically while the hypotheses themselves are sampled. The main bottleneck lies in the construction of these hypothesis. It is not clear how to construct these mutually exclusive hypothesis in the presence of function symbols for the state space. We sacrifice expressiveness for efficiency in our work and consider only predicate symbols for the state space, hence making the particle filter simpler. Natarajan et.al [12] propose the use of relational hierarchies for specifying prior knowledge to an assistant. We consider a representation that is not specific to the assistant setting but propose a general method for extending the HMMs in the relational and hierarchical settings.

One of the important directions for future research is to improve the accuracy of the particle filter. One possible solution that we are currently exploring is the use of Rao-Blackwellization (similar to that of Zettlemoyer et al. but in a simpler setting using predicates for the states) to analytically marginalize out part of the state space and then sample the rest. It is possible to sample the predicates at the next time-step while inferring the values of the variables exactly. This would lead to improved inference while not increasing the size of the state space drastically. Recently Milch and Russell [9] presented an Metropolis-Hastings based MCMC algorithm for relational structures. The algorithm considers states as partial description of the world and uses context specific independences among the state variables to factor out the acceptance probabilities. It remains an interesting future work to explore the use of MCMC for the inference in LoHiHMMs. Such a direction would provide more insights into the relative merits of particle filtering and MCMC in relational domains. We are currently working on automating the grounding of the LoHiHMMs to a ground HMM.

Yet another important direction is to evaluate on real-world domains. Currently the logical version of the model (LoHMM) is being deployed and evaluated on a real-world assistant (name withheld for blind review) for activity recognition and initial results are promising. Our future goal is to use the LoHiHMMs for the same purpose. In many real-world domains, the variables can possibly take infinite values (such as filenames of the documents). There is a need for

the logical representations to handle these infinite variables. We are exploring methods that can make our particle filter to handle these infinite variables in a principled way. The principle advantage of Logical models lie in the fact that they can exploit parameter tying while learning and our future goal is to design efficient learning algorithms for LoHiHMMs.

Acknowledgements: This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA), through the Department of the Interior, NBC, Acquisition Services Division, under Contract Nos. NBCHD030010, FA8750-07-D-0185/0004 and FA8650-06-C-7606. Kristian was paritally supported by a Fraunhofer ATTRACT fellowship. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA, or the Air Force Research Laboratory (AFRL).

References

1. J. Boger, P. Poupart, J. Hoey, C. Boutilier, G. Fernie, and A. Mihailidis. A decision-theoretic approach to task assistance for persons with dementia. In *IJCAI*, 2005.
2. H. Bui, D. Phung, and S. Venkatesh. Hierarchical hidden markov models with general state hierarchy. In *Proceedings of AAAI 04*.
3. H. Bui, S. Venkatesh, and G. West. Policy recognition in the abstract hidden markov models. *JAIR*, 17, 2002.
4. Arnaud Doucet, Nando De Freitas, and Neil Gordon, editors. *Sequential Monte Carlo methods in practice*. 2001.
5. S. Fine, Y. Singer, and N. Tishby. The hierarchical hidden markov model: Analysis and applications. *Machine Learning*, 32:41–62, 1998.
6. L. Getoor and B. Taskar. *Introduction to Statistical Relational Learning*. MIT Press, 2007.
7. K. Kersting, L. De Raedt, and T. Raiko. Logical hidden markov models. *JAIR*, 25:425–456, 2006.
8. M. Leo, T. D'Orazio, and P. Spagnolo. Human activity recognition for automatic visual surveillance of wide areas. In *VSSN '04: Proceedings of the ACM 2nd international workshop on Video surveillance & sensor networks*, 2004.
9. Brian Milch and Stuart Russell. General-purpose mcmc inference over relational structures. In *Proceedings of the 22nd Annual Conference on Uncertainty in Artificial Intelligence (UAI-06)*, 2006.
10. K. Murphy and M. Paskin. Linear time inference in hierarchical HMMs. In *Proceedings of Neural Information Proceesing Systems*, 2001.
11. K. Myers, P. Berry, J. Blythe, K. Conleyn, M. Gervasio, D. McGuinness, D. Morley, A. Pfeffer, M. Pollack, and M. Tambe. An intelligent personal assistant for task and time management. In *AI Magazine*, 2007.
12. S. Natarajan, P. Tadepalli, and A. Fern. A relational hierarchical model for decision-theoretic assistance. In *Proceedings of 17th Annual International Conference on Inductive Logic Programming*, 2007.
13. L. S. Zettlemoyer, H. M. Pasula, and L. P. Kaelbling. Logical particle filtering. In *Proceedings of the Dagstuhl Seminar on Probabilistic, Logical, and Relational Learning*, 2007.