Fast Hierarchical Goal Schema Recognition

Nate Blaylock
Cycorp, Inc.
3721 Executive Center Dr. Ste 100
Austin, Texas, USA
blaylock@cyc.com

James Allen
Department of Computer Science
University of Rochester
PO Box 270262
Rochester, New York, USA
james@cs.rochester.edu

Abstract

We present our work on using statistical, corpus-based machine learning techniques to simultaneously recognize an agent's current goal schemas at various levels of a hierarchical plan. Our recognizer is based on a novel type of graphical model, a Cascading Hidden Markov Model, which allows the algorithm to do exact inference and make predictions at each level of the hierarchy in time quadratic to the number of possible goal schemas. We also report results of our recognizer's performance on a plan corpus.

Introduction

Much work has been done over the years in *plan recognition* which is the task of inferring an agent's goal and plan based on observed actions. *Goal recognition* is a special case of plan recognition in which only the goal is recognized.

For most applications, there are several properties required in order for goal recognition to be useful:

1. **Speed:** Most applications use goal recognition "online", meaning they use recognition results before the observed agent has completed its activity. Ideally, goal recognition should take a fraction of the time it takes for the agent to execute its next action. Plan recognition in the general case has been shown to be intractable (Kautz 1987).
2. **Early/partial prediction:** In a similar vein, applications need accurate goal prediction as early as possible in the observed agent's task execution. Even if a recognizer is fast computationally, if it is unable to predict the goal until after it has seen the last action in the agent's task, it will not be suitable for applications which need recognition results *during* task execution. If full recognition is not immediately available, applications can often make use of partial predictions.

In this paper, we build on our recent work (Blaylock & Allen 2005b) on using statistical, corpus-based techniques for goal recognition. Whereas that work was concerned only with recognition of an agent's top-level goal, we concentrate here on the more general task of *hierarchical goal recognition* — recognition of the chain of an agent's currently active top-level goal and subgoals. As an illustration, consider the tree shown in Figure 1 which represents an execution trace based on a hierarchical transition network (HTN) plan. Here we model goals and subgoals as parameterized action schemas from the SHOP2 HTN planner (Nau *et al.* 2003).

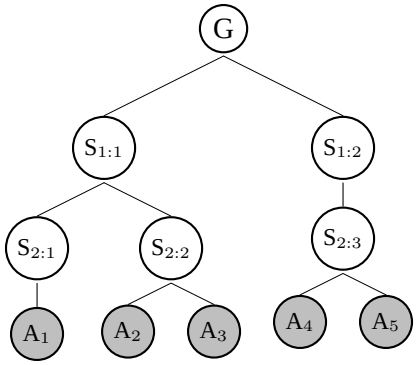

Figure 1: A Sample Hierarchical Plan Execution Tree

Here the root of the tree G is the agent's top-level goal. Leaves of the tree A_1–A_5 represent the atomic actions executed by the agent. Nodes in the middle of the tree represent the agent's various subgoals within the plan. For each executed atomic action, we can define a *goal chain* which is the subgoals which were active at the time it was executed. This is the path which leads from the atomic action to the top-level goal G. The goal chains associated with each atomic action in the tree in Figure 1 are shown in Figure 2. We cast hierarchical goal recognition as the recognition of the goal chain corresponding to the agent's last observed action.

Recognizing such goal chains can provide valuable information not available from a top-level goal recognizer. First, though not full plan recognition, hierarchical goal recognition provides information about which goal an agent is pursuing as well as a partial description of *how*.

Additionally, the prediction of subgoals can be seen as a type of partial prediction. As mentioned above, when a full prediction is not available, a recognizing agent can often make use of partial predictions. A hierarchical recognizer may be able to predict an agent's subgoals even when it is still not clear what the top-level goal is. This can allow a recognizer to potentially make predictions much earlier in an execution stream.

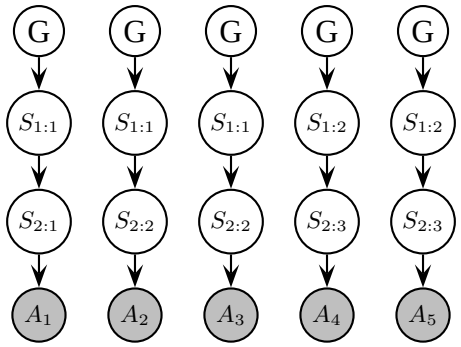

Figure 2: Goal Chains Corresponding to the Execution Tree in Figure 1

In this paper, we limit ourselves to hierarchical recognition of goal *schemas* (cf. (Blaylock & Allen 2005b)). As we discuss below, we have recently done work on extending this schema recognizer to do *instantiated recognition* (recognition of goal schemas and their instantiated parameter values), which we plan to report soon.

The goal recognizer we describe here uses a novel type of graphical model which we term a *Cascading Hidden Markov Model* (CHMM). We first describe this model and then describe our schema recognition algorithm. We then detail our recognition experiments and results, compare previous work and finally conclude and discuss future directions.

Cascading Hidden Markov Models

In our hierarchical schema recognizer, we utilize a *Cascading Hidden Markov Model (CHMM)*, which consists of D stacked state-emission HMMs ($H_{0,D-1}$). Each HMM (H_d) is defined by a 5-tuple $(\sigma_d, \kappa_d, \Pi_d, A_d, B_d)$ where σ_d is the set of possible hidden states; κ_d is the set of possible output states; $\Pi_d = \{\pi_{d:i}\}, i \in \sigma_d$ is the initial state probability distribution; $A_d = \{a_{d:ij}\}, i, j \in \sigma_d$ is the set of state transition probabilities; and $B_d = \{b_{d:ik}\}, i \in \sigma_d, k \in \kappa_d$ is the set of output probabilities.

The HMMs are stacked such that for each HMM (H_d), its output state is the hidden state of the HMM below it (H_{d+1}). For the lowest level (H_{D-1}), the output state is the actual observed output. In essence, at each timestep t, we have a chain of hidden state variables ($X_{0,D-1:t}$) connected to a single observed output O_t at the bottom level. An example of a CHMM is shown in Figure 3.

Here, the dth HMM (i.e., the HMM which starts with the hidden state $X_{d:1}$) is a typical HMM with the output sequence $O_{1,n}$. As we go up the CHMM, the hidden level becomes the output level for the level above it, and so forth.

We will now describe the differences between CHMMs and other hierarchical HMMs. We then discuss how inference is done with CHMMs — in particular, how the forward probability is calculated, as this is a key part of our recognition algorithm.

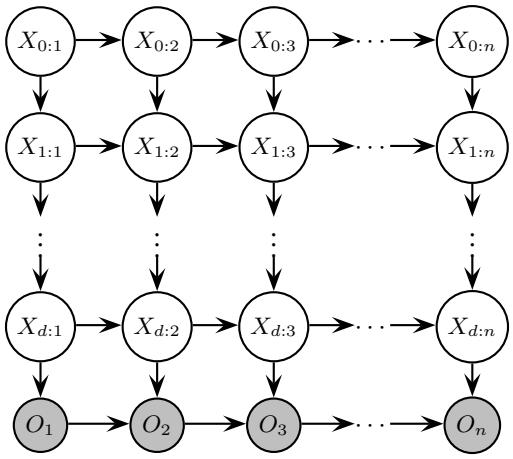

Figure 3: A Cascading Hidden Markov Model (CHMM)

Comparison to Hierarchical HMMs

Hierarchical HMMs (HHMMs) (Murphy & Paskin 2001) and the closely related Abstract HMMs (AHMMs) (Bui, Venkatesh, & West 2002) also represent hierarchical information using a limited-depth stack of HMMs. In these models, an hidden state can output either a single observation, or a string of observations. Each observation can also be associated with a hidden state at the next level down, which can also output observations, and so forth. In HHMMs, when a hidden state outputs an observation, control is transferred to that observation, which can also output and pass control. Control is only returned to the upper-level (and the upper-level moves to its next state) when the output observation has finished its output, which can take multiple timesteps.

In contrast, CHMMs are much simpler. Each hidden state can only output a single observation, thus keeping the HMMs at each level in lock-step. In other words, in CHMMs, each level transitions at each timestep, whereas only a subset transition in HHMMs.

Below, we use CHMMs to represent an agent's execution of a hierarchical plan. As we will discuss there, mapping a hierarchical plan onto a CHMM results in a loss of information which could be retained by using an HHMM (cf. (Bui, Venkatesh, & West 2002)). This, however, is exactly what allows us to do tractable online inference, as we will show below. Exact reasoning in HHMMs has been shown to be exponential in the number of possible states (Murphy & Paskin 2001).

Computing the Forward Probability in CHMMs

An analysis of the various kinds of inference possible with CHMMs is beyond the scope of this paper. Here we only focus on the forward algorithm, which is used in our recognition algorithm below.[1]

[1]The Reader may wonder why we do not use the forward-backward algorithm. The forward-backward algorithm is essentially for post hoc analysis. For goal recognition, we are interested in making "online" predictions, i.e., predictions after each obser-

Typical HMMs In a typical HMM, the forward probability ($\alpha_i(t) = P(o_{1,t}, X_t = i|\Pi, A, B)$) describes the probability of the sequence of outputs observed up until time t ($o_{1,t}$) and that the current state X_t is i, given an HMM model (Π, A, B).

The set of forward probabilities for a given timestep T, ($\alpha(T) = \{\alpha_i(T), i \in \sigma\}$) can be efficiently computed using the well-known forward algorithm, which uses a state lattice (over time) to efficiently compute the forward probability of all intermediate states. This allows it to efficiently compute forward probabilities for the next timestep by simply using those from the previous timestep, using dynamic programming. Although the forward algorithm is likely familiar to most readers, we describe it here to allow us to more easily show how it is augmented for CHMMs below.

First, $\alpha(0)$ is initialized with the initial state probabilities (Π). Then, for each subsequent timestep t, individual forward probabilities are computed using the following formula:

$$\alpha_j(t) = \left[\sum_{i \in \sigma} \alpha_i(t-1) a_{ij}\right] b_{jo_t} \quad (1)$$

The complexity of computing the forward probabilities for a sequence of T observations is O($|\sigma|^2 T$) (where σ is the set of possible hidden states). However, as we will use the forward probability in making online predictions in the next section, we are more interested in the complexity for *extending* the forward probabilities to a new timestep (i.e., calculating $\alpha(t+1)$ given $\alpha(t)$). For extending to a new timestep, the runtime complexity is O($|\sigma|^2$), or quadratic in the number of possible hidden states.

Algorithm Overview In a CHMM, we want to calculate the forward probabilities for each depth within a given timestep: $\alpha(t) = \{\alpha_d(t)\}, d \in 0, D-1$, where $\alpha_d(t) = \{\alpha_{d:i}(t)\}, i \in \sigma_d$. This can be done one timestep at a time, cascading results up from the lowest level ($D-1$).

Initialization of each level occurs as normal — as if it were a normal HMM — using the start state probabilities in Π_d. For each new observation o_t, the new forward probabilities for the chain are computed in a bottom-up fashion, starting with $\alpha_{D-1}(t)$. At this bottom level, the new forward probabilities are computed as for a normal HMM using Equation 1.

We then move up the chain, computing one forward probability set at a time, using the results of the level below as observed output. However, we cannot use Equation 1 to calculate these forward probability sets ($\alpha_d(t)$), as the forward algorithm assumes that output is observed with certainty. While this was the case for level $D-1$ (where the output variable is o_t), for all other levels, the output state is actually also a hidden state ($X_{d+1:t}$), and thus uncertain.

We overcome this by first observing that, although we do not know the value of $X_{d+1:t}$ with certainty, if we have the forward probability distribution for that node ($\alpha_{d+1}(t)$), we can use it as a probability distribution over possible values for the state. As discussed above, we can compute the forward probability set at the bottom level $\alpha_{D-1}(t)$, which gives us a probability distribution over possible values of $X_{D-1:t}$. In order to calculate the forward probability at the next level up $\alpha_{D-2}(t)$ (as well as higher levels), we need to augment the forward algorithm to work for HMMs with uncertain output.

Changing the forward algorithm to handle uncertain output is straightforward, and needs only to include a weighted sum over output probabilities as shown here:

$$\alpha_{d:j}(t) = \left[\sum_{i \in \sigma_d} \alpha_{d:i}(t-1) a_{ij}\right] \left[\sum_{k \in \sigma_{d+1}} \alpha_{d+1:k}(t) b_{jk}\right] \quad (2)$$

Adding this additional summation does not change the complexity of the forward algorithm. For updating each level, it remains quadratic in the number of hidden states $O(|\sigma_d|^2)$.

Recognition Algorithm

From the previous discussion, it may be quite apparent where we are going with this. A quick comparison of the goal chains in Figure 2 and the CHMM in Figure 3 shows a remarkable resemblance. Our hierarchical goal schema recognizer models an agent's plan execution with a CHMM, with one level for each subgoal level in the hierarchical plan. The use of CHMMs requires us to have a plan tree of constant depth, which is not usually not a valid assumption, including for the corpus we use below. Given a variable depth corpus, we use a padding scheme to extend all leaves to the deepest level. Space constraints preclude us from a discussion of the details of this scheme and how it is reflected in the prediction algorithm. The interested reader is referred to (Blaylock 2005) for details.

After each observed action, predictions are made at each subgoal level using what we will call *augmented forward probabilities*. Early experiments[2] based purely on the CHMM forward probabilities gave only mediocre results. In order to improve performance, we added observation-level information to the calculations at each level by making both transition and output probabilities context dependent on the current and last observed action (bigram). The idea was that this would tie upper-level predictions to possible signals present only in the actual actions executed (as opposed to just some higher-level, generic subgoal). This is similar to what is done in probabilistic parsing (Charniak 1997), where lexical items are included in production probabilities to provide better context.

The only change we made was to the transition probabilities (A_d) and output probabilities (B_d) at each level. Thus, instead of the transition probability $a_{d:ij}$ being $P(X_{d:t} = j|X_{d:t-1} = i)$, we expand it to be conditioned on the observed actions as well:

$$a_{d:ij} = P(X_{d:t} = j|X_{d:t-1} = i, O_t, O_{t-1})$$

vation, which the forward-backward algorithm does not lend itself to.

[2]Space precludes us from reporting these here. See (Blaylock 2005) for details.

Total Sessions	5000
Goal Schemas	10
Action Schemas	30
Ave Actions/Session	9.5
Subgoal Schemas	28
Ave Subgoal Depth	3.8
Max Subgoal Depth	9

Table 1: The Monroe Corpus

Similarly, we added bigram information to the output probabilities ($b_{d:ik}$):

$$b_{d:ik} = P(X_{d:t} = i | X_{d+1:t} = k, O_t, O_{t-1})$$

We will now describe how the CHMM is trained, and then how predictions are made. We then analyze the runtime complexity of the recognition algorithm.

Training the CHMM

As a CHMM is really just a stack of HMMs, we need only to estimate the transition probabilities (A_d), output probabilities (B_d) and start state probabilities (Π_d) for each depth d. To estimate these, we used the Monroe corpus, which is an artificially-generated HTN plan corpus in an emergency management domain. It was generated by randomizing the SHOP2 planner (Nau *et al.* 2003) and using it to produce plans given stochastically-generated goals and start states (Blaylock & Allen 2005a). Its vital statistics can be found in Table 1. We converted the corpus into sets of goal chains, which we used the to estimate the different probability distributions.

Predictions

At the start of a recognition session, a CHMM for the domain is initialized with start state probabilities from the model. Upon observing a new action, we calculate the new forward probabilities for each depth using the CHMM forward algorithm described above.

Using the forward probabilities, n-best predictions are made separately for each level. The n most likely schemas are chosen, and their combined probability is compared against a confidence threshold τ.[3] If the n-best probability is greater than τ, a prediction is made. Otherwise, the recognizer does not predict at that level for that timestep.

It is important to note that using this prediction algorithm means that it is possible that, for a given timestep, subgoal schemas may not be predicted at all depths. It is even possible (and actually occurs in our experiments described below) that the depths at which predictions occur can be discontinuous, e.g., a prediction could occur at levels 4, 3, and 1, but not 2 or 0. We believe this to be a valuable feature as subgoals at different levels may be more certain than others.

[3]Although it would be possible to set a separate threshold for each depth, our results below are based on using a single threshold for all levels.

Complexity

The runtime complexity of the recognizer for each new observed timestep is the same as that of forward probability extension in the CHMM: $O(D|S|^2)$, where D is depth of the deepest possible goal chain in the domain (not including the observed action), and S is the set of possible subgoals (at any level). Thus the algorithm is linear in the depth of the domain and quadratic in the number of possible subgoals in the domain.

Experiments

We now report on our experiments using the hierarchical goal schema recognizer. For the experiments, we used 4500 plan sessions from the Monroe corpus for training and the remaining 500 for testing. This is the same data used in our experiments on top-goal schema recognition (Blaylock & Allen 2005b) and allows us to make comparisons below.

Before we describe the experiments and their results, however, we briefly describe the metrics we use to report results.

Metrics

We report results for individual subgoal depths, as well as totals.[4] For each depth, we use the same metrics introduced in our previous work (Blaylock & Allen 2005b) to measure results. These were introduced to try to measure the general requirements of goal recognizers described above. *Precision* and *recall* report the number of correct predictions divided by total predictions and total prediction opportunities, respectively. *Convergence* and *convergence point* stem from the fact that, oftentimes, the recognizer will be unsure very early on in a session, but may at some point 'converge' on the correct answer, predicting it from that point on until the end of the plan session. *Convergence* measures the percentage of plan sessions where the correct answer was converged upon.[5] For those plan sessions which converge, *convergence point* reports the average action observation after which it converged, divided by the average number of actions for the converged sessions. This is an attempt to measure how *early* in the plan session the recognizer was able to zero in on the correct answer. In hierarchical recognition, some subgoals only span one timestep (e.g., they only result in one executed atomic action), in which case, it does not make sense to report convergence or a convergence point. For all levels, we only report convergence and convergence point for subgoals which correspond to at least two timesteps.

Results

The results of the experiment are shown in Table 2. Overall, the results are very encouraging with 81.9% precision and 52.3% recall for 1-best prediction which jumps to 94.9%

[4]Predictions on "filler" subgoals inserted to make the plan trees of constant depth were not counted here. See (Blaylock 2005) for details.

[5]This essentially measures how many *last* predictions were correct, i.e., whether we *ended* predicting the right answer.

	1-best ($\tau = 0.7$)				2-best ($\tau = 0.95$)			
level	**prec.**	**recall**	**conv.**	**conv. pt**	**prec.**	**recall**	**conv.**	**conv. pt**
0	85.6%	58.6%	100%	5.2/10.2	90.7%	62.0%	100%	4.9/10.2
1	84.3%	54.8%	71.8%	2.9/6.1	96.1%	77.3%	99.0%	2.3/5.6
2	89.3%	46.3%	45.8%	3.4/4.7	93.0%	64.3%	84.4%	3.5/4.8
3	74.8%	42.8%	41.2%	2.7/3.5	97.6%	80.1%	99.0%	3.5/4.5
4	78.7%	53.5%	61.8%	3.3/3.7	97.0%	73.2%	100%	3.2/3.8
5	59.3%	46.1%	6.2%	3.8/4.2	99.1%	77.1%	100%	2.0/3.9
6	69.3%	69.3%	0.0%	N/A	100%	100%	100%	1.0/4.0
7	95.2%	95.2%	N/A	N/A	100%	100%	N/A	N/A
8	100%	100%	N/A	N/A	100%	100%	N/A	N/A
Total	81.9%	52.3%	65.0%	3.8/6.8	94.9%	71.4%	95.7%	3.3/6.1

Table 2: Results of the Hierarchical Schema Recognition Experiment

	1-best ($\tau = 0.7$)				2-best ($\tau = 0.95$)			
level	**prec.**	**recall**	**conv.**	**conv. pt**	**prec.**	**recall**	**conv.**	**conv. pt**
top	95.6%	55.2%	96.4%	5.4/10.2	99.4%	58.7%	99.8%	5.4/10.3

Table 3: (For Comparison) Results of Top-level Goal Schema Recognition (Blaylock & Allen 2005b)

precision and 71.4% recall for 2-best prediction.[6] In the 2-best case, 95.7% of sessions converged on the right answer. On average, this was after a little more than half of the actions had been observed.

We will now discuss the various levels in detail, first looking at the results for predicting top-level goal schemas (level 0) and then the other levels.

Top-level Results Results at the top level are also encouraging with 85.6% precision and 58.6% recall for 1-best prediction and 90.7% precision and 62.0% recall for 2-best. For comparison, we reprint the results of our top-level recognizer on the same data set in Table 3.

For recall, convergence, and convergence point, the two recognizers perform fairly equivalently, both in 1-best and 2-best prediction. Precision, however, is markedly lower in the hierarchical recognizer, for both the 1-best and 2-best cases. Whereas precision is 95.6 percent for 1-best in the flat recognizer, it drops to 85.7 percent for the hierarchical recognizer. A similar drop in precision from 99.4 percent to 91.5 percent is shown in the 2-best case.

Although there seem to be several factors involved in this drop, it is perhaps most important to mention two. First is the loss of true bigram information within the hierarchical recognizer. In the hierarchical recognizer, the top-level goal is predicted based on predictions at the next immediate subgoal level (level 1) as opposed to directly from the action observation level as is the top-level goal recognizer. Converting a plan tree into a sequence of goal chains looses explicit information about the actual previous subgoal.

[6]Due to the lack of common test sets and reporting metrics in the field, it is difficult to gauge these results wrt. related work. We hope our recent work on creating plan corpora (Blaylock & Allen 2005a) and reporting metrics (Blaylock & Allen 2005b) will help alleviate this problem.

Secondly, and most importantly, a direct comparison of algorithm performance is difficult because the hierarchical recognizer is doing much more than simple top-level goal classification as was done in the top-level goal recognizer. Arguably, we could improve performance by using the hierarchical recognizer for the subgoal levels and then the flat recognizer for top-level recognition, although this then looses the generalization that the hierarchical recognizer can also handle cases where several top-level goals are pursued serially.

Other Levels Results at lower levels for 1-best prediction are on average not as good as those at the top level. The foremost reason is that there is actually more competition at lower levels. At lower levels, more subgoals are possible, whereas only the 10 top-level schemas are possible at level 0. Also, there are several lower-level subgoals per level throughout a goal session. Only one top-level goal makes the transition probabilities much simpler at the top level as well (basically transition probabilities are 1 between the same schemas and 0 between any others). This seems to especially account for the very low convergence numbers for levels 5 and 6 (6.2% and 0.0%, respectively), where there were only a few data points and these were recognized well at the start of their interval, but not at the end. (In 2-best prediction both of these move to 100% convergence.)

That said, in the 1-best case, recognition results are fairly good for levels 1, 2, 7, and 8, although there is a trough between them. A partial explanation is that, at higher levels, there are less competitors. Thus, as we move to lower levels, things become harder to predict. At the same time, the lower we go, the closer to the observed output, and thus closer to certain information. Thus, the last two levels have very good precision and recall because they are so closely related to the observed action. (Levels 7 and 8 contained no subgoals

which span more than one timestep, hence convergence and convergence point are not reported.)

It appears that in the middle (e.g., levels 3-6), the recognizer tends to not to distinguish well among close competitors. That this is the case can be shown by looking at the 2-best case, where all levels move to the 90's or 100 percent for precision and also improve dramatically in recall.

Related Work

Relatively little work has been done on hierarchical goal schema recognition. Pynadath (Pynadath & Wellman 2000) casts hierarchical schema recognition as parsing using probabilistic state-dependent grammars. Online recognition is performed by converting the grammar into a dynamic belief network (DBN) with a special update algorithm. For partially observable states, however, the runtime complexity is quadratic in the number of states consistent with observation, which grows exponentially with the number of unobservable state nodes.

Bui et al. (Bui, Venkatesh, & West 2002) use Abstract Hidden Markov Models (AHMMs) (described above) to model plan execution for recognition. Recognition is done using a DBN, but because this is intractable, Bui uses a method called Rao-Blackwellization (RB) to estimate one group of variables and do exact inference on the others. Given knowledge about the state of the world as well as the ending time of upper-level subgoals, this method can make inference with AHMMs tractable.

The recognizer was used in a system which tracked human behavior in an office building at three abstract levels, representing individual offices at the bottom level, then office groups, then finally the entire building. Policies at each level were defined specific to each region (for example the policy (behavior) of using the printer in the printer room). In this model, only certain policies are valid in a given state (location), which helps reduce the ambiguity. The domain was modeled such that lower-level policies become impossible as the agent moves to another room, which makes it fairly clear when they then terminate.

Although the algorithm was successful for this tracking task, it is unclear how effective estimation of policy termination would be in general (e.g., when most policies are valid in most states).

Our recognizer, on the other hand, explicitly simplifies away from the issue of subgoal ending times by forcing each level of the CHMM to transition at every time step. In doing so, we lose certain information about tree structure, which is retained in the approach of Bui et al. — specifically, knowledge about the previous subgoal. Consider again the plan tree in Figure 1 along with its corresponding CHMM (in Figure 2). At depth 1 (i.e., the first level below the top-level goal), two subgoals are executed: $S_{1:1}$ and $S_{1:2}$. This transition occurs after timestep 3, and at this point the information that the preceding subgoal was $S_{1:1}$ can be used in making predictions at timestep 4. In subsequent timesteps, however, we lose this information because of the Markovian assumption. Thus, at timestep 5, the HMM at level 1 thinks that the previous subgoal was $S_{1:2}$, although the last *actual* subgoal was $S_{1:1}$. This gives us a win in runtime, however, and, with the use of bigram information from observed actions, does not seem to affect recognition much in the Monroe domain.

Conclusions and Future Work

The recognizer we have described here has two nice features (which correspond to the two desired traits of goal recognizers described above). First, recognition is fast and scalable, running in time quadratic to the number of possible goal schemas. Second, the recognizer supports partial goal recognition, allowing it to make predictions earlier in the agent's task execution. It supports n-best prediction at each subgoal level as well as non-prediction at subgoal levels about which it is not yet confident.

We have recently incorporated a variant of our top-level goal parameter recognizer (Blaylock & Allen 2005b) into the schema recognizer to create an *instantiated* hierarchical goal recognizer which we plan to report soon.

Acknowledgments

We would like to thank the reviewers for their helpful comments. This material is based upon work supported by a grant from DARPA under grant number F30602-98-2-0133; two grants from the National Science Foundation under grant number IIS-0328811 and grant number E1A-0080124; ONR contract N00014-06-C-0032; and the EU-funded TALK project (IST-507802). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the above-mentioned organizations.

References

Blaylock, N., and Allen, J. 2005a. Generating artificial corpora for plan recognition. In *User Modeling 2005*.

Blaylock, N., and Allen, J. 2005b. Recognizing instantiated goals using statistical methods. In Kaminka, G., ed., *Workshop on Modeling Others from Observations*.

Blaylock, N. J. 2005. Towards tractable agent-based dialogue. Technical Report 880, University of Rochester, Department of Computer Science. PhD thesis.

Bui, H. H.; Venkatesh, S.; and West, G. 2002. Policy recognition in the Abstract Hidden Markov Model. *Journal of Artificial Intelligence Research* 17.

Charniak, E. 1997. Statistical techniques for natural language parsing. *AI Magazine* 18(4).

Kautz, H. A. 1987. A formal theory of plan recognition. Technical Report 215, University of Rochester, Department of Computer Science. PhD thesis.

Murphy, K. P., and Paskin, M. A. 2001. Linear time inference in hierarchical HMMs. In *NIPS-01*.

Nau, D.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research* 20:379–404.

Pynadath, D. V., and Wellman, M. P. 2000. Probabilistic state-dependent grammars for plan recognition. In *UAI-2000*.