Dynamic Abstraction Networks

Victoria Manfredi, Sridhar Mahadevan *

Technical Report 2005-33
May 17, 2005

Department of Computer Science
140 Governors Drive
University of Massachusetts
Amherst, Massachusetts 01003-4601
{vmanfred, mahadeva}@cs.umass.edu

Abstract

We propose *dynamic abstraction networks*, a new type of hierarchical graphical model that combines state and temporal abstraction. Fundamental to our approach is the belief that temporal abstraction and state abstraction are intertwined, and should be modeled jointly in the same network. For the state abstraction component of the model we use the hierarchical hidden Markov model, while for the policy abstraction component we use a structure based on the abstract hidden Markov model. We present results on both real and synthetic activity data comparing the policy abstractions learned with and without the use of state abstraction.

Keywords: graphical models, hierarchy, activity modeling

1 Introduction

A hallmark of human intelligence is the ability to form spatial and temporal abstractions of the world. We can reason about whether or not to reach for an object, or go on vacation, or how best to prepare for retirement. Correspondingly, we are also effortlessly able to maintain state estimates of our location at multiple levels of abstraction: driving from New York City to Miami, we know whether we are in Georgia or Maryland, or near a particular exit. What is even more striking is that state abstraction is often intertwined with temporal abstraction, enabling us to not only jointly reason about one from the other, but also learn both simultaneously. This synchrony is perhaps one reason why we often report our state using temporal abstractions ("I'm in graduate school"), or our actions using state abstractions ("I'm vacationing in Italy"). We formalize these intuitions using the framework of graphical models, but other approaches are possible.

Previous work in graphical models has largely focused on studying temporal abstraction or state abstraction in isolation. We propose a new type of hierarchical graphical model, *dynamic abstraction networks* (DANs),

*This work was supported in part by the National Science Foundation under grant ECS-0218125. Any opinons, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. Victoria Manfredi was supported by a National Science Foundation graduate research fellowship.

that combines state and temporal abstraction. Introduced in [1], the term dynamic abstraction refers to the changing relevance of state variables over time. For example, traffic conditions are relevant while driving but irrelevant once at the destination. We extend the definition to include the idea that each abstract state has certain admissible abstract policies, generalizing the idea from Markov decision processes that each local state has certain admissible actions. DANs augment the state hierarchy of the hierarchical hidden Markov model (HHMM) [6] with a policy abstraction component based on the abstract hidden Markov model (AHMM) [3].

Jointly encoding spatial and temporal abstraction in a graphical model like the DAN permits us to do both joint inference and joint learning. For instance, suppose I know you have seen a Broadway play (abstract policy). I can infer using a DAN how probable it was that you were in New York City (abstract state). Intuitively, abstract policies are intricately tied to abstract states. For instance, New York City is both a temporal and a spatial abstraction: its geographical location permits you to both execute such abstract policies as visit the Metropolitan Museum of Art or attend a Broadway play, and to define such spatial abstractions as the five boroughs of New York City or the state of New York.

In the rest of this paper, we define the DAN model and present results comparing temporal abstractions learned with and without the use of state abstraction.

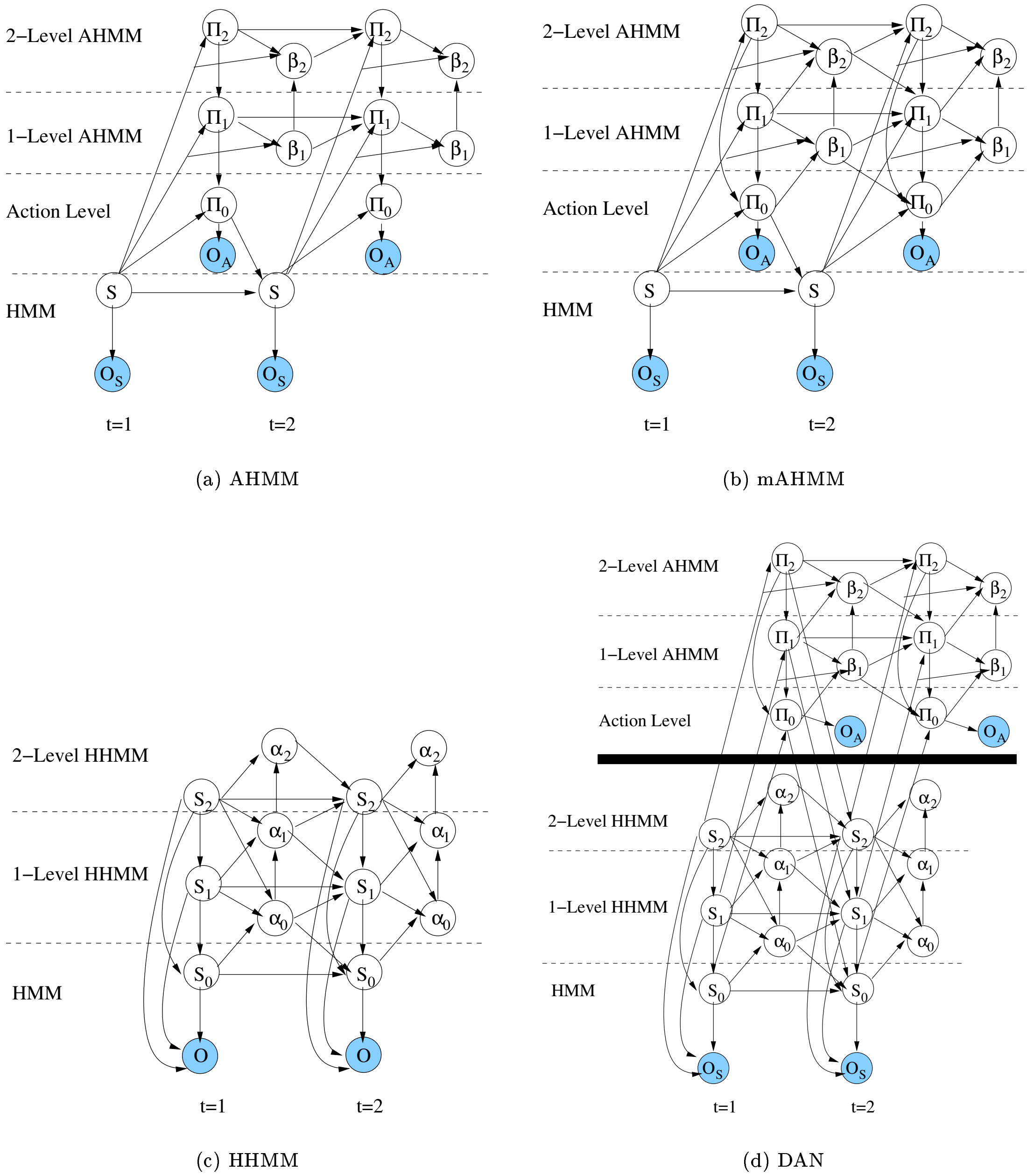
2 Dynamic Abstraction Networks

One of the simplest models used to infer hidden state from a series of observations is the hidden Markov model (HMM). Given the basic HMM, one extension is to add a policy layer that chooses among multiple HMMs, as in the AHMM [3]. Shown in Figure 1a, the bottom layer of the AHMM represents an HMM. Alternatively, as in [8], the bottom layer could also represent a Kalman filter. The second layer in the AHMM represents actions, Π_0. Which action can be executed at time t depends on the state and policy at time t. If actions are continuous, we can include an action observation node O_A as shown in Figure 1a. The third and higher layers in the AHMM represent successively more abstract policies, Π_i where $i > 0$. Policy termination nodes β_i indicate when a new policy at level i can be chosen. In summary, an AHMM specifies policies as mappings from states to probability distributions over actions, where higher-level policies are mappings from states to distributions over lower-level policies.

A key drawback of the AHMM is that because it lacks an explicit mechanism for defining abstract states, higher level policies are difficult to define and cannot share common state abstractions. Consequently, abstract policies must be defined over concrete states, greatly increasing the number of parameters that must be learned. We would therefore like to incorporate some method of state abstraction into the AHMM. While one approach might be to simply add a decision tree that encodes state abstraction, this does not enable us to jointly learn abstract states and policies. If we instead incorporate a state abstraction layer based on a graphical model like the HHMM, we are able to jointly learn both state and policy abstractions.

In comparison to the AHMM, the HHMM encodes no policies or actions. Instead, it encodes state abstractions as mappings from all higher-level states to a probability distribution over each lower-level state. While only policies at level i influence policies at level $i-1$ in the AHMM, states at level i as well as all higher-level states influence states at level $i-1$. The DBN representation of the HHMM defined in [10] is shown in Figure 1c; the bottom layer of the HHMM represents an HMM while higher layers represent successively more abstract states, S_i where $i > 0$. State termination nodes α_i indicate when a new state at level i can occur. Along with HMMs, HHMMs have been widely applied in sequence classification problems, such as in speech and robotics [10, 13].

We can think of DANs as a merging of a state hierarchy, represented by the HHMM, and a policy hierarchy represented by the modified version of the AHMM shown in Figure 1b. The modified AHMM (mAHMM) encodes two types of additional dependencies not represented in the AHMM. The first type of dependency is represented by edges from each level i policy node to all lower-level policy nodes. The second is represented by edges to and from policy nodes at level i to policy termination nodes at level $i+1$. We added these dependencies because empirically we found that without them the AHMM had difficulty learning to execute



(a) AHMM (b) mAHMM

(c) HHMM (d) DAN

Figure 1: DBN representation of (a) abstract hidden Markov model, (b) modified abstract hidden Markov model, (c) hierarchical hidden Markov model, and (d) dynamic abstraction network. We emphasize that (d) is just one realization of a dynamic abstraction network, and other configurations are possible.

a unique policy with high probability in a given state (see the next section). This may be a consequence of the type of data that was used. We note that by adding these dependencies we violate the conditional independence theorem for AHMMs in [3] which permitted them to design a fast Rao-Blackwellized particle filtering inference method for the AHMM. We merge the mAHMM and HHMM by adding edges from state nodes at time t on level i to policy and policy termination nodes at time t on level i and from policy nodes at time t on level i to state nodes at time $t+1$ on level i gives us the DAN shown in Figure 1d. Formally, we define a K-level DAN as comprised of an intertwined state hierarchy and policy hierarchy defined as follows.

Policy Hierarchy. A policy hierarchy H_π with K levels is given by the ordered tuple $H_\pi = (O_A, \Pi_0, \Pi_1, \ldots, \Pi_K)$.

- O_A is the set of action observations. O_A is only necessary if actions are continuous or partially observable.
- Π_0 is the set of primitive (one-step) actions.
- Π_i is the set of abstract policies at level $1 \leq i \leq K$, defined in more detail below.

State Hierarchy. A state hierarchy H_s with K levels is given by the ordered tuple $H_s = (O_S, S_0, S_1, \ldots, S_K)$.

- O_S is the set of state observations.
- S_j is the set of abstract states at levels $0 \leq j \leq K$, defined in more detail below.

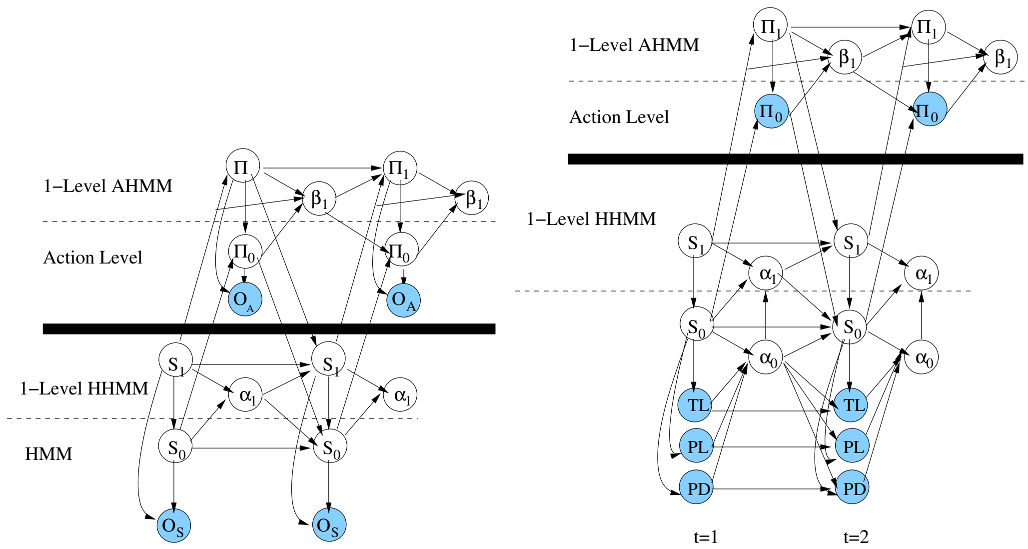
Abstract Policies. At level i, each abstract policy $\pi_i \in \Pi_i$ is given by the tuple $\pi_i =< S_{\pi_i}, B_{\pi_i}, \beta_{\pi_i}, \sigma_{\pi_i} >$.

- *Selection set.* $S_{\pi_i} \subset S_i$ is the set of states in which π_i can be executed.
- *Termination probability.* $B_{\pi_i} \subset S_i$ is the set of states in which π_i can terminate. $\beta_{\pi_i} : B_{\pi_i} \rightarrow (0, 1]$ is the probability that policy π_i terminates in state B_{π_i}. Note that for all termination states that are not also selection states, policy π_i terminates with probability one.
- *Selection probability.* $\sigma_{\pi_i} : \Pi_{i+1:K} \times S_{\pi_i} \rightarrow [0, 1]$ is the probability with which a policy $\pi_i \in \Pi_i$ can be initiated when executing abstract policies $\pi_{i+1} \in \Pi_{i+1}, \ldots, \pi_K \in \Pi_K$ in state $s_{\pi_i} \in S_{\pi_i}$.

Abstract States. At level j, each abstract state $s_j \in S_j$ is given by the tuple $s_j =< \Pi_{s_j}, \alpha_{s_j}, \sigma_{s_j} >$. Note that an abstract state may be part of more than one higher level state; for instance, states at level j may be letters while states at level $j+1$ are words.

- *Entry set.* $\Pi_{s_i} \subset \Pi_i$ is the set of policies which enter state s_i.
- *Exit probability.* $\alpha_{s_j} : S_j \rightarrow (0, 1]$ is the probability that the agent exits state $s_j \in S_j$ when in states $s_{j+1} \in S_{j+1}, \ldots, s_K \in S_K$.
- *Transition probability.* $\sigma_{s_j} : S_{j+1:K} \times \Pi_j \times S_j \rightarrow [0, 1]$ is the probability that the agent transitions to state $s_j^{t+1} \in S_j$ from state $s_j^t \in S_j$ when in parent states $s_{j+1}^t \in S_{j+1}, \ldots, s_K^t \in S_K$ and executing policy $\pi_j^t \in \Pi_j$.

Figure 1d shows a 2-level DAN. For the policy hierarchy, Π nodes encode the *selection sets* S_π and the *selection probabilities* σ_π, while β nodes encode the *termination probabilities* β_π. For the state hierarchy, S nodes encode the *entry sets* Π_s and the *transition probabilities* σ_s, while α nodes encode the *exit probabilities* α_s. At level i, choosing policy $\pi_i \in \Pi_i$ depends in part on state $s_i \in S_i$ but executing $\pi_i \in \Pi_i$, by choosing policy $\pi_{i-1} \in \Pi_{i-1}$, depends in part on state $s_{i-1} \in S_{i-1}$. That is, choosing a policy at level i depends on the state at level i, but executing a policy at level i depends on the state at level $i-1$.



(a) Lab/Entry DAN

(b) Taxi DAN

Figure 2: Models used in experiments.

We note that due to the structure of the AHMM, for all $1 \leq i < K$ where K is the number of levels in the AHMM, if $\beta_i^t = continue$ then all policies at levels i and higher must remain the same and $\beta_{i+1:K}^t = continue$. Thus for all $j \geq i$, the edge from S^t to β_{j+1}^t, the edge from Π_{j+1}^t to β_{j+1}^t, the edge from S^t to Π_i^{t+1}, and the edge from Π_{j+1}^t to Π_j^t, can be removed due to context-specific independence (CSI) properties [2]. Similarly, if $\beta_i^t = end$ then the edge from β_i^t to β_{j+1}^t and the edge from Π_j^t to Π_i^{t+1} can be removed since the next policy no longer depends on the current policy. These CSI properties permit the design of a fast inference algorithm, Rao-Blackwellized particle filtering, for the AHMM [3]. Other work [7] has shown that using variational methods for approximate inference in the AHMM results in a significant speedup in inference. A multiagent extension to the AHMM that maintains the CSI properties to have an associated Rao-Blackwellized particle filtering algorithm for inference was proposed in [12].

One of the key ideas for developing this model is that abstract states are useful for learning abstract policies. Consequently, abstract states are fed into all policy levels including the actions. In the results section we show empirically that DANs learn "cleaner" policies than AHMMs.

3 Experiments

We used three datasets: the *Lab* and *Entry* activity data from [11] and data generated using the *Taxi* domain in [5]. The *Lab* data consists of six types of sequences collected in the hallway of a laboratory. There are three instances of each type; we use two of each for training and the third for testing. The primary feature of interest for this data was that the sequences overlapped significantly. The *Entry* data consists of eight types of sequences collected in the entryway of a building. There are ten instances of each type; we use nine of each for training and the tenth for testing. The primary feature of interest for this data was that the sequences periodically crossed each other. Given the sequences of (x, y) locations, we artificially created actions for both datasets by computing the x and y differences between the locations at times t and $t+1$.

The *Taxi* domain [5] consists of a five-by-five grid with four taxi terminals, R, G, Y, and B, see Figure 3. The goal of the agent is to pick up a passenger from one terminal, and deliver her to another one (possibly

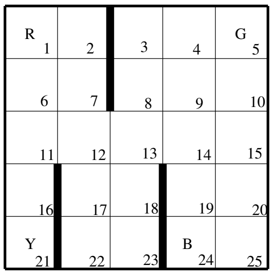


Figure 3: Grid for the Taxi domain.

the same one). There are six actions: north (N), south (S), east (E), west (W), pick up passenger (PU), and put down passenger (PD). 80% of the time N, S, E, and W work correctly; for 10% of the time the agent goes right and 10% of the time the agent goes left. The agent's state consists of the taxi location (TL), the passenger location (PL), and the passenger destination (PD). Note that $PL = 1$ when the passenger has been picked up and $PD = 1$ when the passenger has been delivered. We generated 1000 training sequences from a reinforcement learning agent trained in this domain. Each sequence is the trajectory of states visited and actions taken in one episode as the trained agent uses its learned policy to reach the goal.

We used Expectation-Maximization [4] to learn the parameters for 1-level AHMM, mAHMM, HHMM and DAN models for the *Lab* data and *Entry* data. We learned only 1-level mAHMM and DAN models for the *Taxi* data. Figure 2 shows the DAN models used in the experiments; note that the HHMM components of the DANs have been slightly modified to suit the datasets. Bayes Net Toolbox [9] was used to implement and train all models.

For the *Lab* and the *Entry* data, the state and action observations were treated as two-dimensional Gaussians and initalized with randomly chosen points from the data. The remaining distributions were multinomials, and except for the β_1, α_1, Π_1, and S_1 distributions, were initialized randomly. For both the *Lab* and the *Entry* data we set $|\Pi_1| = 8$, $|\Pi_0| = 4$, $|S_1| = 8$, $|S_0| = 30$, $|\beta_1| = 2$, and $|\alpha_1| = 2$. For the *Taxi* data, all distributions were multinomials and except for the β_1, α_0, α_1, Π_1, S_0, and S_1 distributions, were initialized randomly. For the *Taxi* data we set $|S_1| = 5$, $|S_0| = 25$, $|TL| = 25$, $|PL| = 5$, $|PD| = 5$, $|\Pi_1| = 6$, $|\Pi_0| = 6$, $|\alpha_0| = 2$, $|\alpha_1| = 2$, and $|\beta_1| = 2$.

For all experiments, higher level states and policies were biased to change more slowly than lower level states and policies; e.g., the floor you are on should not change more frequently than the room you are in. For the *Taxi* data, this was done by initializing the β_1, α_0, α_1, Π_1, S_0, and S_1 distributions as follows, where $i = 0, 1$.

$$P(\beta_1^t \mid \Pi_0^t, \Pi_1^t, S_1^t) = \begin{cases} 0.95 & \text{if } \beta_1^t = \text{continue} \\ 0.05 & \text{otherwise} \end{cases}$$

$$P(\alpha_0^t \mid S_0^t, O_S^t) = \begin{cases} 0.95 & \text{if } \alpha_0^t = \text{continue} \\ 0.05 & \text{otherwise} \end{cases}$$

$$P(\alpha_1^t \mid \alpha_0^t, S_0^t, S_1^t) = \begin{cases} 1 & \text{if } \alpha_1^t = \alpha_0^t = \text{continue} \\ 0.5 & \text{if } \alpha_1^t = \text{continue and} \\ & \quad \alpha_0^t = \text{end} \\ 0 & \text{otherwise} \end{cases}$$

$$P(\Pi_1^{t+1} \mid \Pi_1^t, S_1^t, \beta_1^t) = \begin{cases} 1 & \text{if } \beta_1^t = \text{continue and } \Pi_1^{t+1} = \Pi_1^t \\ 1/|\Pi_1| & \text{if } \beta_1^t = \text{end} \\ 0 & \text{otherwise} \end{cases}$$

$$P(S_0^{t+1} \mid \Pi_0^t, S_0^t, S_1^{t+1}, \alpha_0^t, \alpha_1^t) = \begin{cases} 1 & \text{if } \alpha_0^t = \text{continue and } S_0^{t+1} = S_0^t \\ 1/|S_0| & \text{if } \alpha_0^t = \text{end} \\ 0 & \text{otherwise} \end{cases}$$

$$P(S_1^{t+1} \mid \Pi_1^t, S_1^t, \alpha_1^t) = \begin{cases} 1 & \text{if } \alpha_1^t = \text{continue and } S_1^{t+1} = S_1^t \\ 1/|S_1| & \text{if } \alpha_1^t = \text{end} \\ 0 & \text{otherwise} \end{cases}$$

For the *Lab* and *Entry* data, we used the same initializations for β_1, Π_1, and S_1. Since the DAN used for the *Lab* and *Entry* data (Figure 2a) did not include α_0^t, we instead initialized α_1^t and S_0 to be,

$$P(\alpha_1^t \mid S_0^t, S_1^t) = \begin{cases} 0.95 & \text{if } \alpha_1^t = \text{continue} \\ 0.05 & \text{otherwise} \end{cases}$$

$$P(S_0^{t+1} \mid \Pi_0^t, S_0^t, S_1^{t+1}, \alpha_1^t) = \begin{cases} 1 & \text{if } \alpha_1^t = \text{continue and } S_0^{t+1} = S_0^t \\ 1/|S_0| & \text{if } \alpha_1^t = \text{end} \\ 0 & \text{otherwise} \end{cases}$$

4 Results

We compared how well the trained models for each of the datasets inferred level 1 policies and states on test sequences. Table 1 shows the entropy, $-\sum_x p(x)\log p(x)$, for the filtered and smoothed posterior distributions $P(\Pi_1^t \mid O_S^{1:t}, O_A^{1:t})$, $P(\Pi_1^t \mid O_S^{1:T}, O_A^{1:T})$, $P(S_1^t \mid O_S^{1:t}, O_A^{1:t})$, and $P(S_1^t \mid O_S^{1:T}, O_A^{1:T})$ averaged over all timesteps t for all test sequences for the *Lab* and *Entry* datasets. This metric evaluates how certain each model is in its inference: i.e., on a given timestep for a given sequence, is one policy more probable or are all policies equally probable? Ideally, the models should identify at least some policies as significantly more probable than other policies; equivalently, the relevant posterior should have low entropy. Table 1 shows that for inferring level 1 policies, on both datasets, the DAN has the lowest entropy while the AHMM has the highest entropy. For inferring level 1 states, the DAN has higher entropy than the HHMM for the Lab data but lower entropy for the Entry data. We note that the average entropy for the state posteriors is lower than that for policy posteriors. This may be a consequence of the spatial nature of the *Lab* and *Entry* data.

While Table 1 indicates that the DAN is learning policy and state abstractions that on average permit it to identify the hidden abstract policies of unseen sequences with more certainty than the AHMM or mAHMM, and to identify the hidden abstract states with about the same certainty as the HHMM, a natural question to ask is exactly how certain is it. Figure 4 shows the probability of each level 1 policy for sequence 5 of the Lab data for the AHMM, mAHMM and DAN, and the probability of each level 1 state for the HHMM and DAN. We found that on average the most likely policy inferred by the DAN had probability 0.908 for the *Lab* data and 0.840 for the *Entry* data with variances of 0.052 and 0.042 respectively while the most likely policy inferred by the mAHMM had probability 0.854 for the *Lab* data and 0.723 for the *Entry* data with variances of 0.053 and 0.068 and the most likely policy inferred by the AHMM had probability 0.293 for the

	Model	Lab-F	Lab-S	Entry-F	Entry-S
Policy	AHMM	1.8621	1.8252	1.8792	1.8666
	mAHMM	0.3722	0.3217	0.6912	0.6058
	DAN	0.1548	0.1183	0.3970	0.3591
State	HHMM	0.0526	0.0251	0.1506	0.0672
	DAN	0.0761	0.0300	0.1380	0.0782

Table 1: Entropy of the filtered (F) and smoothed (S) policy and state posterior distributions, averaged over all timesteps for all test sequences for the *Lab* and *Entry* data.

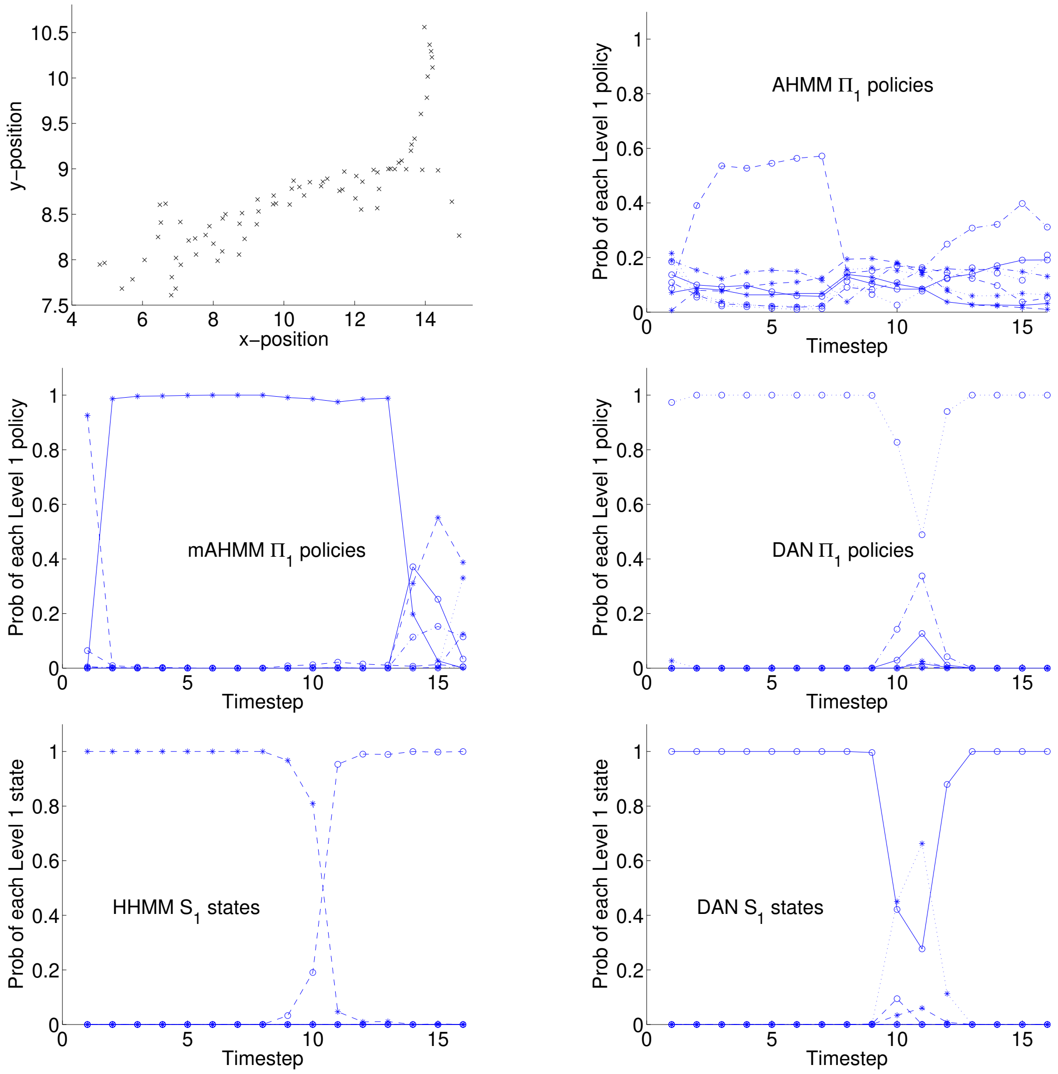


Figure 4: Distributions over level 1 policies and states for sequence 5 of the *Lab* data. Top left graph plots all of the lab data.

Lab data and 0.250 for the *Entry* data with variances of 0.014 and 0.021. Similarly we found that on average the most likely state inferred by the DAN had probability 0.948 for the *Lab* data and 0.941 for the *Entry* data with variances of 0.032 and 0.021 respectively while the most likely state inferred by the HHMM had probability 0.985 for the *Lab* data and 0.927 for the *Entry* data with variances of 0.001 and 0.021.

The influence of the abstract state on the learned abstract policies and vice versa is also apparent from Figure 4. Notice the dip in probability around timestep 11 for the DAN's most likely policy and state graphs in Figure 4. Timestep 11 corresponds to the juncture seen at location $(13, 9)$ in the graph in Figure 4 that plots all of the *Lab* data. At that point in the data there are two different directions in which the agent can go, seen as two overalapping sequences beginning to diverge. For the DAN, while it maintains the same policy for the entire sequence with lower probability at the juncture, it briefly switches to a different state at the junction point. In comparison, the HHMM chooses one state until the juncture, and then switches to and maintaines a different state after. This indicates the type of state abstractions learned are affected by the availability of action and policy information.

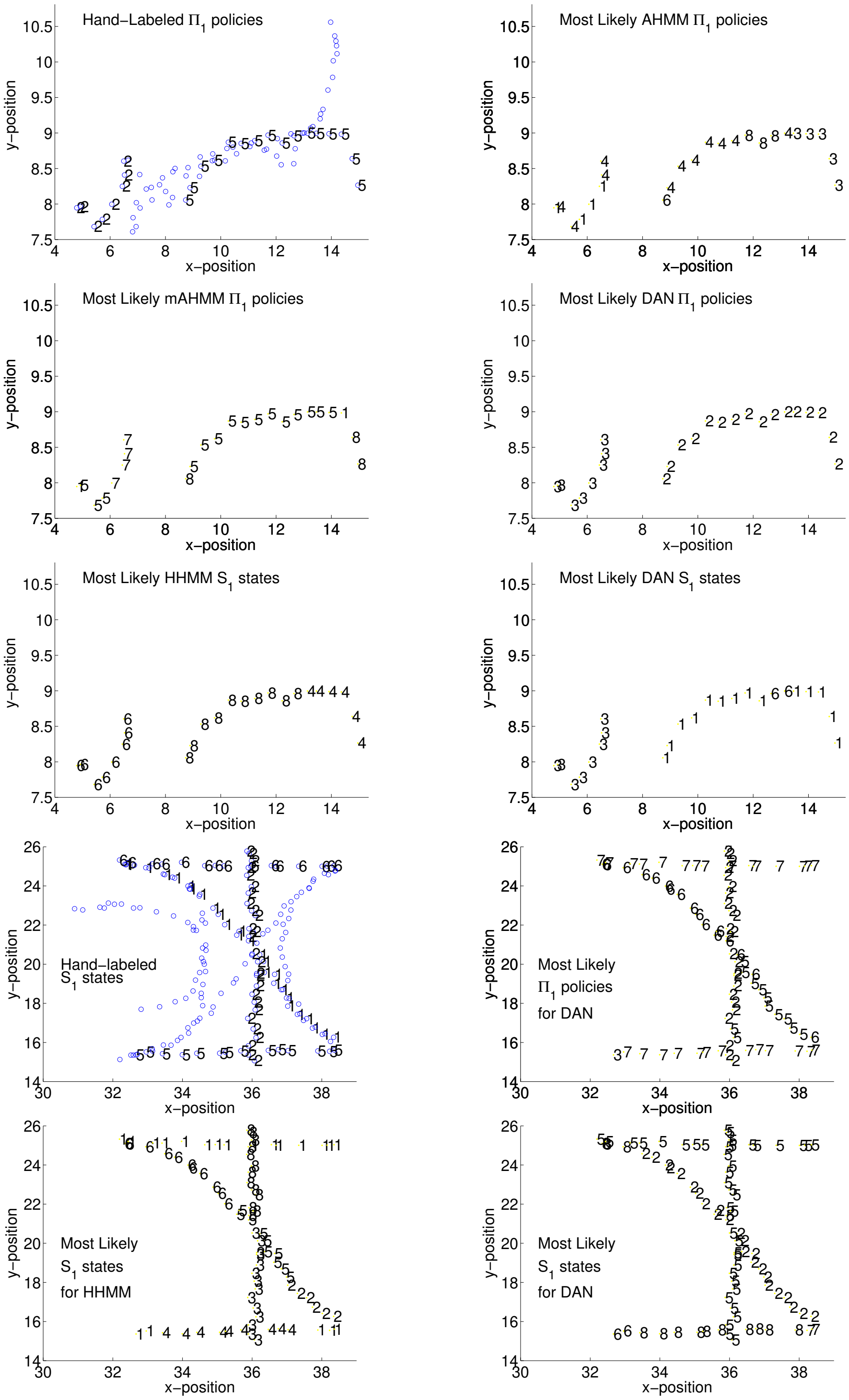


Figure 5: Most likely level 1 policies and states for test sequences 2 and 5 of the *Lab* data and for and test sequences 1, 2, 5, and 6 of the *Entry* data. The hand-labeled most likely policies are overlain over the full data.

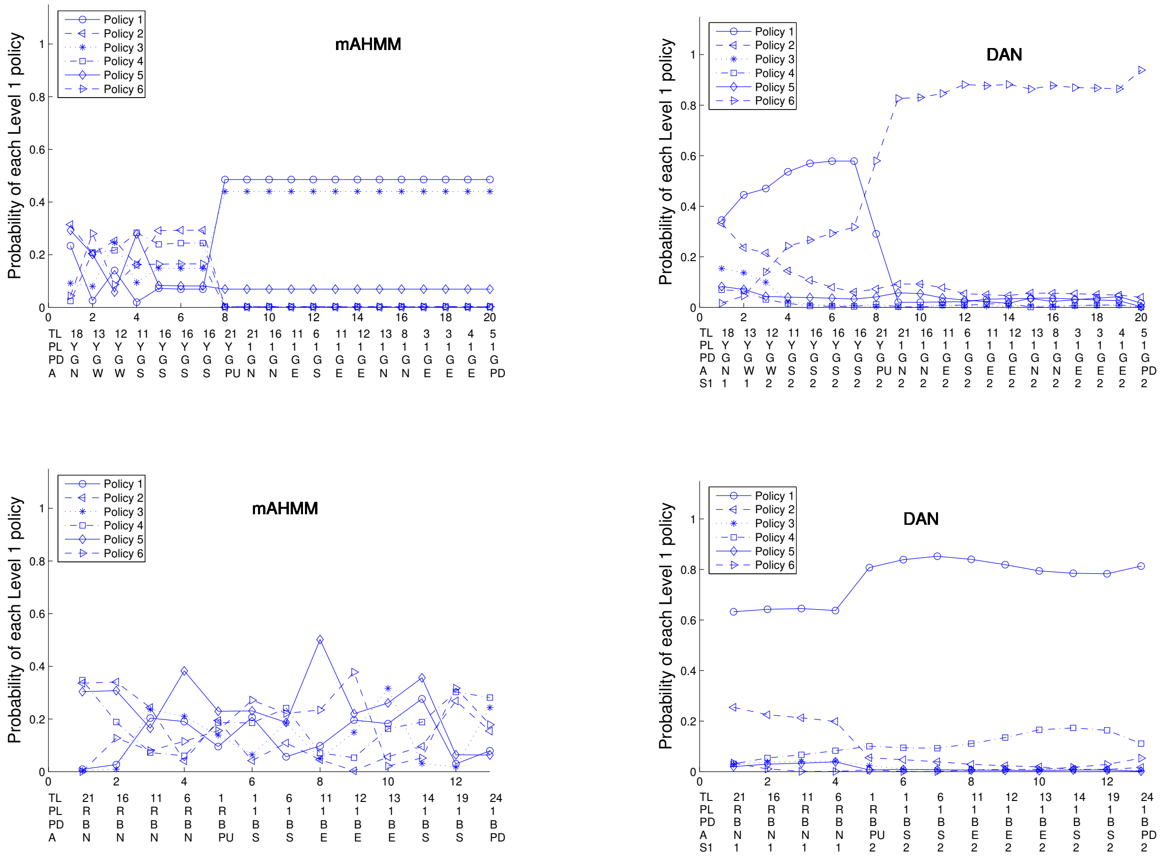


Figure 6: Level 1 policies and states for two sample sequences from the *Taxi* data. The TL, PL, and PD state values and P_0 actions are also shown.

Figure 5 shows the most likely policies for sequences 2 and 5 of the *Lab* data for the AHMM, mAHMM, and DAN, and the most likely states for the HHMM and DAN, as well as the most likely states and policies for sequences 1, 2, 5, and 6 of the *Entry* data for the HHMM and DAN. What is apparent from Figure 5 is that the DAN is identifying unique or semi-unique policies and states for different sequences. Notice particularly in the most likely state graphs for the *Entry* data that the HHMM does not maintain the same most likely state across the juncture at (x, y)-position $(36, 21)$, unlike the DAN.

Unlike the *Lab* and *Entry* datasets, the *Taxi* domain focuses less on spatial data. Figure 6 shows the probability of each level 1 policy and state for two sample Taxi sequences for the mAHMM and DAN. What we see from Figure 6 is that now the mAHMM has difficulty identifying a single most likely policy with high probability, unlike before, while the DAN model is still able to identify a unique most likely policy with high probability. As shown in Figure 6, the mAHMM performs particularly poorly on the second sequence, never identifying a single policy as most likely for more than a couple of timesteps. Note that while the mAHMM has difficulty identifying a single policy as most likely, this is not a consequence of the policies themselves being poorly learned. Rather, because the mAHMM has learned every policy over the entire state space (due to the structure of the model), all policies are equally good, so any can be used. In essence only a single global policy has been learned. The problem with this is that it cannot be reused, unlike the more local policies learned by the DAN. In particular we see from the DAN graphs in Figure 6 that Policy 1 is used for part or all of both sequences.

5 Conclusions and Future Work

In summary, we proposed a new type of hierarchical graphical model, dynamic abstraction networks, which combines state and temporal abstraction. Using this model we showed that abstract policies learned using state abstraction appear to be better than abstract policies learned using only a flat state representation; we also found that the availability of policy information seems to affect the types of state abstractions that were learned. Further experiments as well as more diverse data would be useful to clarify how abstract policies affect the learning of abstract states and vice versa, particularly experiments using multiple layers of abstraction (unlike the single layer explored in this paper). Other interesting experiments would include iterating between learning state abstractions and learning policy abstractions, and determining the effect of learning policy abstractions from only actions.

References

[1] A. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Special Issue on Reinforcement Learning, Discrete Event Systems Journal*, 13:41–77, 2003.

[2] C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-specific independence in Bayesian networks. In *UAI-96*, pages 115–123, 1996.

[3] H. Bui, S. Venkatesh, and G. West. Policy recognition in the abstract hidden Markov model. *Journal of Artificial Intelligence Research*, 17:451–499, 2002.

[4] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, 39:1–38, 1977.

[5] T. G. Dietterich. Hierarchical reinforcment learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[6] S. Fine, Y. Singer, and N. Tishby. The hierarchical hidden markov model: Analysis and applications. *Machine Learning*, 32(1):41–62, 1998.

[7] J. Johns and S. Mahadevan. A variational learning algorithm for the abstract hidden Markov model. In *AAAI'05*, 2005.

[8] L. Liao, D. Fox, and H. Kautz. Learning and inferring transportation routines. In *AAAI'04*, 2004.

[9] K. Murphy. The Bayes net toolbox for Matlab. *Computing Science and Statistics*, 33, 2001.

[10] K. Murphy and M. Paskin. Linear time inference in hierarchical HMMs. In *NIPS'01*, 2001.

[11] S. Osentoski, V. Manfredi, and S. Mahadevan. Learning hierarchical models of activity. In *IROS'04*, 2004.

[12] S. Saria and S. Mahadevan. Probabilistic plan recognition in multiagent systems. In *Intl. Conf. on AI and Planning Systems*, 2004.

[13] G. Theocharous. *Hierarchical learning and planning in partially observable Markov decision processes.* PhD thesis, Michigan State University, 2002.