# Representing hierarchical POMDPs as DBNs for multi-scale robot localization

**Georgios Theocharous** THEOCHAR@AI.MIT.EDU
**Kevin Murphy** MURPHYK@AI.MIT.EDU
**Leslie Pack Kaelbling** LPK@AI.MIT.EDU
MIT Artificial Intelligence Laboratory, 200 Technology Square, MIT Building NE43, Cambridge, MA 02139 USA

## Abstract

We explore the advantages of representing hierarchical partially observable Markov decision processes (H-POMDPs) as dynamic Bayesian networks (DBNs). In particular, we focus on the special case of using H-POMDPs to represent multi-resolution spatial maps for indoor robot navigation. Our results show that a DBN representation of H-POMDPs can train significantly faster than the original learning algorithm for H-POMDPs or the equivalent flat POMDP, and requires much less data. In addition, the DBN formulation can easily be extended to parameter tying and factoring of variables, which further reduces the time and sample complexity. This enables us to apply H-POMDP methods to much larger problems than previously possible.

## 1. Introduction

Partially observed Markov decision processes (POMDPs) have become a popular model in AI in general, and in mobile robotics in particular (Cassandra et al., 1996; Koenig & Simmons, 1998). Unfortunately, standard methods for planning, inference, and learning with POMDPs all take time exponential in the number of (discrete) states, $S$, making them impractical for large problems.

A hierarchical extension to POMDPs was introduced by (Theocharous et al., 2001). Such H-POMDPs, which represent the state-space at multiple levels of abstraction, scale much better to larger environments. In particular, they simplify the planning and learning problems. Planning is simpler (requires less time) in H-POMDPs because abstract states (at the coarse-level of resolution) have lower entropy, i.e., are more deterministic (Theocharous & Mahadevan, 2002a). Learning is simpler (requires less data) in H-POMDPs because the number of free parameters is reduced, and the structure of the model provides a way of encoding prior knowledge. In this paper, we focus on the learning problem, and do not address planning, i.e., we assume a fixed (given) control policy.

To learn the parameters of an H-POMDP (using e.g., EM) requires an inference algorithm, since the true state of the world is not observed. The inference algorithm that was used in (Theocharous et al., 2001) was based on the one proposed in (Fine et al., 1998) for hierarchical HMMs, and takes $O(ST^3)$ time, where $T$ is the length of the sequence. This made it intractable to train the model on long data sequences, which is essential for learning large environments. (Long sequences are necessary to reduce perceptual aliasing.)

In this paper, we show how to do inference in H-POMDPs in $O(S^{1.5}T)$ time by representing the H-POMDP as a dynamic Bayesian network (DBN); this is an extension of (Murphy & Paskin, 2001). This allows us to learn much larger models much faster, without requiring manual segmentation of the training data. In addition, we exploit the DBN formalism to factor the robot state into location and orientation; this further decreases the sample and time complexity.

We explain and test our model in the domain of map learning and robot localization. However, the techniques are more general, and can be applied to any POMDP which exhibits hierarchical structure. As such, this technique is not directly comparable to more specialized map learning techniques such as SLAM (e.g., (Dissanayake et al., 2001)), which are harder to generalize to other problems, but which have the advantage of performing online structure learning.

The structure of the paper is as follows. In Section 2, we define H-POMDPs more precisely; in Sections 3 and 4, we discuss how to perform inference and learning in such models; in Section 5, we show some experiments which demonstrate the superiority of hierarchical POMDPs over flat POMDPs, and factored H-POMDPs over unfactored H-POMDPs, both in terms of accuracy (for a fixed amount of training data) and

speed of learning. Finally, in Section 6, we conclude and discuss future work.

## 2. Representation

### 2.1. Hierarchical HMMs

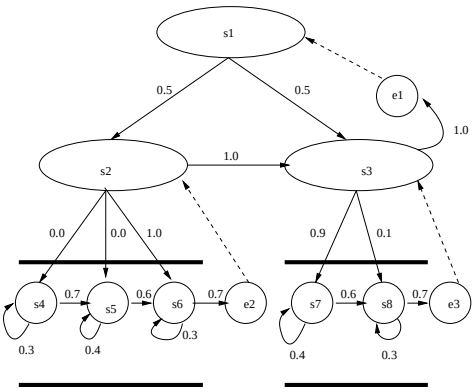


*Figure 1.* State transition diagram of a three-level HHMM. The ovals represent abstract states, the small circles represent concrete states. Dotted arcs represent return of control. See text for details. In this example, the sub-HMMs have a left-to-right structure, but this need not be the case in general. Absent arcs have zero probability. Some arcs which are shown also have zero probability; this is just to ensure that the corresponding flat model in Figure 2 is not too complex.

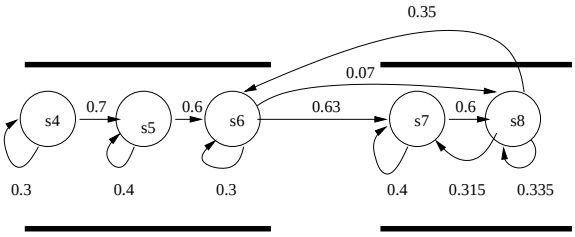


*Figure 2.* A flattened version of Figure 1.

Hierarchical HMMs (Fine et al., 1998) are like HMMs except the states of the stochastic automaton can emit single observations or strings of observations. (For simplicity of exposition, we shall assume all observations are discrete symbols, but HHMMs can easily be generalized to handle continuous observations.) Those that emit single symbols are called "production states", and those that emit strings are termed "abstract states". The strings emitted by abstract states are themselves governed by sub-HHMMs, which can be called recursively. When the sub-HHMM is finished, control is returned to wherever it was called from; the calling context is stored using a depth-limited stack.[1] The result is a set of nested sequences. An HHMM is thus a generalization of a segment (semi-Markov) model (Ostendorf et al., 1996) to multiple levels, where the duration within each segment is controlled by an HMM.

We illustrate the generative process with the example in Figure 1. We start in the root node, $s_1$, and then make a "vertical transition" to either $s_2$ or $s_3$; suppose we choose $s_2$. From there, we make another vertical transition, until we end up at one of the concrete nodes at the leaves, say $s_6$. From state $s_6$, we emit the first symbol. We then make a "horizontal transition", say to the exit state $e_2$ (which cannot emit any symbols); this causes control to be returned to the calling state, $s_2$. From $s_2$, we make a horizontal transition to $s_3$, and then a vertical transition to, say, $s_7$, and emit another symbol. And so on.

Any HHMM can be converted to an HMM by creating a state for every possible legal stack configuration $X_t^1, \ldots, X_t^D$. (If the HHMM transition diagram is a tree, as above, there will be one HMM state for every HHMM production state; If the HHMM transition diagram is a DAG (i.e., it has shared substructure), this structure must be duplicated in the HMM, generally resulting in a larger state-space.) Then, for every pair of states $(s_1, s_2)$, the flat transition probabilitiy is the sum of the probabilities of all paths that transition between $s_1$ and $s_2$ in the HHMM. For example, consider converting Figure 1 to Figure 2. The probability of the self-transition for $s_8$ in the flat model is $0.3 + 0.7 \times 0.5 \times 0.1 = 0.335$, corresponding to the paths $s_8 \rightarrow s_8$ and

$$s_8 \rightarrow e_3 \rightarrow s_3 \rightarrow e_1 \rightarrow s_1 \rightarrow s_3 \rightarrow s_8$$

(Note that the $s_8 \rightarrow e_3 \rightarrow s_3 \rightarrow s_8$ path is illegal, since $s_3$ does not have a self-loop; the only legal transition from $s_3$ is to $e_1$ and then up to the root.) The resulting flat model is clearly more highly interconnected (because the extra paths induced by passing through abstract nodes must be represented explicitly as new edges), and hence is harder to learn.

### 2.2. Hierarchical POMDPs

A hierarchical POMDP (Theocharous et al., 2001) extends an HHMM by conditioning all state transitions (and optionally, observations) on the action that is performed. In this paper, we restrict our attention to a two-level hierarchical model designed for robot navigation. The states are as follows: $X_t^2$ (an abstract node) represents which corridor the robot is in, and $X_t^1$ (a concrete node) represents which grid cell within the corridor, plus the orientation. See Figure 3 for an example.

The actions (controls), $U_t$, represent "go forward by 1m", "turn left by $90\mathrm{deg}$", or "turn right by $90\mathrm{deg}$". In this paper, we ignore the problem of how to choose actions, and simply model them as exogeneous inputs. Finally, $Y_t$ represents the observations (sensor measurements). In our simulations, we assume this represents the presence or absence of a wall (ob-

[1] The fact that the stack has a fixed depth, $D$, means that an HHMM is a finite-state model; it is therefore less expressive, but more efficient, than stochastic context-free grammars and recursive transition networks.

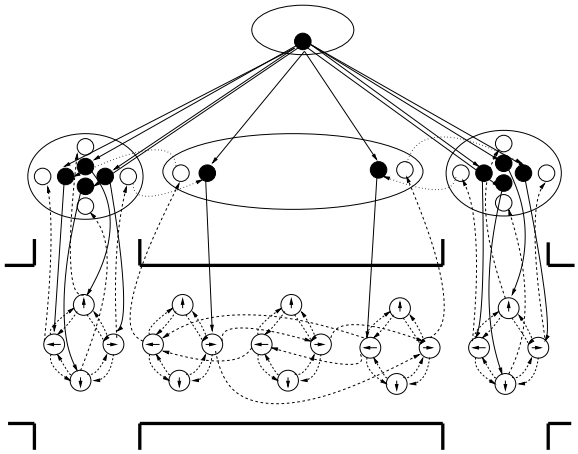

*Figure 3.* State transition diagram of the H-POMDP used to model corridor environments. Throughout the paper we refer to this model as the "hierarchical model". Large ovals represent abstract states; the small solid circles within them represent entry states, and the small hollow circles represent exit states. The small circles with arrows represent concrete state and orientation. Arcs represent non-zero transition probabilities as follows: Dashed arrows from concrete states represent concrete horizontal transitions, dotted arrows from exit states represent abstract horizontal transitions, and solid arrows from entry states represent vertical transitions.

stacle) on the front, back, left and right sides of the robot; thus $Y_t$ can be represented by four bits. In our actual robot experiments (see Section 5.4), we use a neural network to convert laser range finder readings into this form.

In the original HHMM model, when we exit from a sub-model, we "pop" the old concrete state off the call stack; the new concrete state is chosen according to what is left on the stack, namely just the current abstract state number. However, in the robot navigation domain, the concrete state we enter depends on which exit state we used; i.e., which end of the new corridor we enter depends on which end of the previous corridor we exited from. Hence we must condition the new concrete state not only on the new abstract state, but also on the previous concrete state. However, this is equivalent to a flat model, since it allows full interconnectivity between concrete states. Hence we introduce the notion of "bottleneck" states, which in our domain are the ends of corridors; information can only flow from one sub-HMM to another via these bottlenecks. In particular, we allow multiple exit states, which summarize the previous concrete state; the new concrete state is allowed to depend on the previous exit state, but not the previous concrete state. We will make this more precise below.

Every H-POMDP can be converted to a flat POMDP in a manner similar to flattening an HHMM (see Figure 4 for an example). However, the flattening operation loses most of the structure, making learning much harder, as we show below.

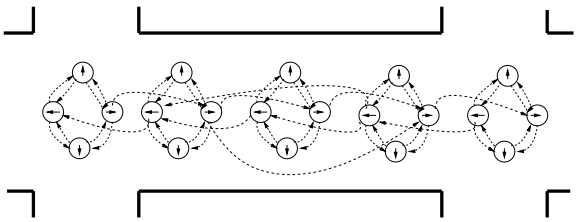

*Figure 4.* State transition diagram for a flattened version of Figure 3. Throughout this paper we refer to this model as the "flat" model. Transition matrices for each action can be computed from the hierarchical model as follows: for each pair of concrete states $s_1, s_2$, we need to sum up the probabilities of all the paths that transition from $s_1$ to $s_2$ under some action $a$.

### 2.3. Representing H-POMDPs as DBNs

A DBN model for HHMMs was introduced by (Murphy & Paskin, 2001). We can extend this to the H-POMDP case by making three changes: we add action nodes, $U_t$, we add orientation nodes, $\Theta_t$, and we allow exit nodes, $E_t$, to be multi-valued (not just binary). See Figure 5.

The action nodes represent the movement made by the robot. The orientation nodes are present because we now factor $X_t^1$ into concrete location, $L_t^1$, and orientation, $\Theta_t$, instead of having to duplicate each (concrete) location four times as is done in Figures 3 and 4. We will denote the abstract location, $X_t^2$, by $L_t^2$. The exit node $E_t$ can take on five possible values, representing no-exit, north-exit, east-exit, south-exit, and west-exit. If $E_t$ = no-exit, then we make a horizontal transition at the concrete level, but the abstract state is required to remain the same. We will explain the model in more detail below.

#### 2.3.1. NOTATION

Lower case letters denote values of random variables. To shorten notation, we will sometimes denote the probability of events like $P(\Theta_t = \theta, L_t^2 = a, L_{t-1}^1 = i, L_t^1 = j)$ by $P(\theta, a, i, j)$, i.e., we will use $\theta_t$ to denote an assignment to $\Theta_t$, $a$ for $L_t^2$, $j$ for $L_t^1$ and $i$ for $L_{t-1}^1$.

#### 2.3.2. TRANSITION MODEL

We now define the conditional probability distributions (CPDs) of each type of node in the DBN. All distributions, except for the observations, are conditioned on the input $U_{t-1}$; this is not shown explicitly, to simplify notation. For the abstract nodes,

$$P(L_t^2 = j | L_{t-1}^2 = i, E_{t-1} = e) = \begin{cases} \delta(i,j) & \text{if } e = \text{no-exit} \\ H^2(i,e,j) & \text{otherwise} \end{cases}$$

where $H^2(i,e,j)$ is the abstract horizontal transition matrix through exit of type $e$ and $\delta$ is the Kronecker delta function. (This corresponds to the connections between the big ovals in

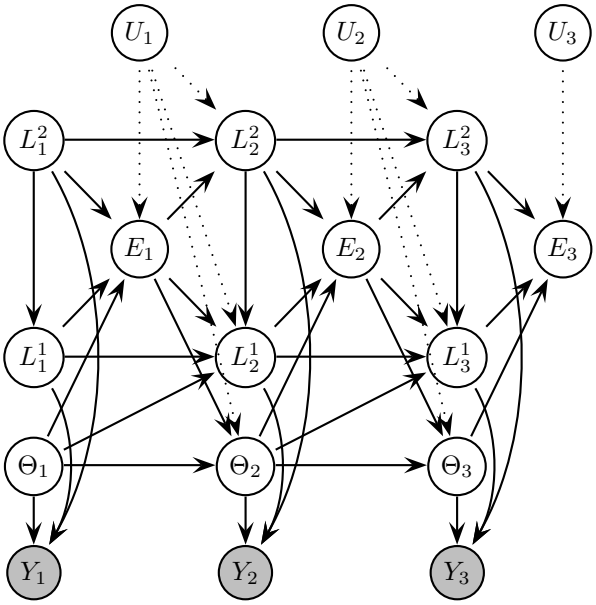


*Figure 5.* A 2-level factored H-POMDP represented as a DBN. Throughout the paper we refer to this model as the "factored hierarchical DBN". The arcs from the action node, $U_t$, are shown dotted merely to reduce clutter. The $L_t^2$ nodes denote the abstract state, the $L_t^1$ nodes denote the concrete location, the $\Theta_t$ nodes denote the orientation, the $E_t$ nodes denote the state of the exit variable, and $Y_t$ denotes the state of the observation variables.

Figure 3.) For the concrete nodes,

$$P(L_t^1 = j | L_{t-1}^1 = i, E_{t-1} = e, \theta_{t-1}, L_t^2 = a) = \begin{cases} H^1(i, \theta_{t-1}, a, j) & \text{if } e = \text{no-exit} \\ V(e, a, j) & \text{otherwise} \end{cases}$$

where $H^1(i, \theta_{t-1}, a, j)$ is the concrete horizontal transition matrix (the horizontal connections below each oval in Figure 3), and $V(e, a, j)$ is the concrete vertical entry vector (the downward pointing arcs in Figure 3). For the exit nodes,

$$P(E_t = e | j, \theta_t, a) = X(j, a, \theta_t, e)$$

where $X(j, a, \theta_t, e)$ is the probability of concrete state $j$ entering exit state $e$ given that it is in abstract state $a$ and has orientation $\theta_t$ (the upward pointing arcs in Figure 3).

#### 2.3.3. Observation model

Although the observations are shown as a single node in Figure 5, in fact we make the naive Bayes assumption that $P(Y_t|X_t) = \prod_{i=1}^{4} P(Y_t^i|X_t)$, where $Y_t^i$ are the four different observations, and $X_t = (L_t^1, L_t^2, \Theta_t)$ is the entire state.

Our observation model is the probability of seeing a wall or opening on each of the four sides of the robot. However, we must first map the global coordinate frame of the map to the robot's local coordinate frame by taking into account its orientation. We can do this as follows. Let $B(a, j, \theta, y) \stackrel{\text{def}}{=} P(Y_t^\theta = y | L_t^2 = a, L_t^1 = j)$ for all $t$, where $\theta \in \{N, E, S, W\}$, $B(a, j, N, y)$ is the probability of observing $y$ to the north of cell $(a, j)$, etc. Then:

$$\begin{aligned} P(Y_t^F = y | L_t^1 = j, \theta_t, a) &= B(a, j, \theta_t, y) \\ P(Y_t^B = y | L_t^1 = j, \theta_t, a) &= B(a, j, R_{180}\theta_t, y) \\ P(Y_t^L = y | L_t^1 = j, \theta_t, a) &= B(a, j, R_{-90}\theta_t, y) \\ P(Y_t^R = y | L_t^1 = j, \theta_t, a) &= B(a, j, R_{90}\theta_t, y) \end{aligned}$$

where $R_{180}$ represents a 180 degree rotation matrix, mapping north to south, etc. For example, $P(Y_t^B = y | L_t^1 = j, \theta_t = E, L_t^2 = a) = B(a, j, W, y)$, i.e., if the robot is facing east, what the robot sees in its back sensor is determined by what is on the west edge of the cell. This form of parameter tying allows us to learn about the appearance of a cell in all possible directions at once, even if we only approach it in a single direction.

## 3. Inference

There are at least two kinds of inference: online filtering and offline smoothing. By online filtering we mean recursively computing the belief state $P(X_t|y_{1:t}, u_{1:t})$; this is then passed to the controller (policy), in order to choose the next action. By offline smoothing, we mean computing $P(X_t|y_{1:T}, u_{1:T})$; this is necessary for parameter estimation (see Section 4).

The offline inference algorithm used by (Fine et al., 1998; Theocharous et al., 2001) takes $O(K^D T^3)$ time, where $K$ is the number of states at each level of the hierarchy, $D$ is the depth of the hierarchy, and $T$ is the length of the sequence. This makes it intractable to train models from long, unsegmented sequences of data. In addition, no online inference algorithm was presented: the resulting model had to be flattened before being used in a controller. For the DBN representation, on the other hand, we can apply any standard Bayes net inference algorithm, such as junction tree, to perform filtering or smoothing. Exact offline algorithms take $\sim DK^{1.5D}T$ time; the factor of 1.5 arises because the largest clique in the junction tree contains all the state nodes in slice $t-1$, and half of the state nodes in slice $t$ (Murphy & Paskin, 2001). (This is a general result for the DBN representation of H-POMDPs, and is not specific to this application.)

Although it may seem that $O(DK^{1.5D}T)$ is not much improvement on $O(K^{2D}T)$, the time required to do inference in the flat model, note that inference in the flat model cannot be used to compute the expected sufficient statistics needed to learn the hierarchical model (see Section 4). So although inference in flat models is fast, they are not practical, because they require exponentially more data to learn. The DBN representation of H-POMDPs, by contrast, has low sample and time complexity. Furthermore, we can exploit the structure of the DBN even more if we are willing to use approximate

inference, such as assumed density filtering (Boyen & Koller, 1998) or loopy belief propagation (Murphy & Weiss, 2001). Such approximations, together with restricted forms for the CPDs (so that the number of parameters for the bottom level nodes is less than $K^D$), can reduce the time complexity to $O(DKT)$.

The various inference algorithms help us to understand the basic difference between hierarchical and flat models. In a hierarchical model, a transition to an abstract state at time $t$ has zero probability unless the abstract state is able to produce a complete sub-sequence (you cannot enter a corridor unless you are sure you will exit it again); this is what enables the algorithm to work at a higher level of abstraction (since the algorithm does not need to "think" about which concrete state the system may be in). The inference algorithm in (Fine et al., 1998; Theocharous et al., 2001) achieves this by considering all possible subsequences of observations under the different abstract states, which takes $O(T^3)$ time.[2]

In the DBN representation we can achieve the same result by asserting that at the last time slice, all sub-HMMs must be finished. Since we do not know which exit state they will use, just that $E_T \neq$ no-exit, we assign "soft" or "virtual" evidence to the $E_T$ node: the local evidence becomes $(0, 0.25, 0.25, 0.25, 0.25)$, which encodes the fact that we can exit by any direction with equal likelihood, but the no-exit condition is impossible. If we do not add this virtual evidence, the hierarchical DBN behaves the same as the flat model (before the first iteration of EM; thereafter, the hierarchical model will outperform the flat model, since it can learn better, as we discuss below.)

### 3.1. Space requirements

In addition to running time, space is an important issue when training on long sequences: when learning, we need to compute the smoothed belief states for all time-slices; this requires storing all the forward filtered estimates for $t = 1 : T$ until the backwards pass. For the flat model, this takes $O(ST)$ space, where $S = O(K^D)$ is the number of states (one per leaf in the calling graph); for the $O(T^3)$ algorithm, the space requirements are $O(ST^3)$; but for the DBN algorithm, the space requirements are $O(S^{1.5}T)$, which can be prohibitive for large $T$. Fortunately, there is a simple divide-and-conquer algorithm that can perform inference in any DBN in $O(S^{1.5} \log T)$ space, if one waits $O(S^{1.5}T \log T)$ time (Binder et al., 1997; Zweig & Padmanabhan, 2000). Furthemore, approximate inference can reduce $S$ from $O(K^D)$ to $O(KD)$.

[2]The original inference algorithm for HHMMs is very similar to the inside-outside algorithm for SCFGs (see e.g., (Jurafsky & Martin, 2000)), which computes $P(N^i \rightarrow N^j N^k | O_{t:t+k})$, where $P(N^i \rightarrow N^j N^k)$ is the probability that non-terminal $i$ expands into non-terminal $j$ followed by non-terminal $k$. If there are $N$ non-terminals in the grammar and the training sequence is of length $T$, then the Inside-Outside algorithm requires $O(N^3 T^3)$ time.

## 4. Learning

We can compute maximum likelihood parameter estimates using the EM algorithm (or gradient ascent) in the same was as for any Bayes net. In particular, in the E step, we compute the expected sufficient statistics, $\sum_t P(V_t, \text{Pa}(V_t) | y_{1:T}, u_{1:T})$, for each node $V_t$ with parents $\text{Pa}(V_t)$; in the M step, we use the equations in the sections below. (Parameter learning in flat POMDPs models was studied in the context of map learning in e.g., (Thrun et al., 1998).)

The topology of the environment is encoded in the structural zeros of the transition matrices. (Note that if an element is set to 0, EM will leave it as 0.) In this paper, we pre-specify a noisy version of the topology, e.g., we specify that the go-forward action moves to the adjacent state with high probability. However, the robot still has to learn the observation probabilities, i.e., the appearance of each grid cell; without these, the robot will suffer from severe perceptual aliasing, and will not be able to estimate its state.

### 4.1. Estimating the transition model

To shorten notation, let $O = (y_{1:T}, u_{1:T})$ represent all the observed data, and let $n$ be the value of $E_t$ representing no-exit. The parameters can be estimated by normalizing matrices of expected counts, just as in a regular HMM. For example, for the abstract horizontal transition matrix, the matrix of expected counts is

$$H^2(t, i, e, j) \stackrel{\text{def}}{=} P(L^2_{t-1} = i, L^2_t = j, E_{t-1} = e | O)$$

where $e \neq n$; the corresponding maximum likelihood estimate (MLE) is given by

$$\hat{H}^2(i, e, j) = \frac{\sum_{t=2}^{T} H^2(t, i, e, j)}{\sum_{j'} \sum_{t=2}^{T} H^2(t, i, e, j')}.$$

The equations for the other parameters are very similar, so we will just define the count matrices; to compute the corrsponding MLEs, simply sum up over $t$ (the first index) and then normalize with respect to the last dimension (to ensure the result is a stochastic matrix).

For the concrete horizontal transition matrix, we condition on the event $E_{t-1} = n$, meaning we are still in the same sub-HMM:

$$H^1(t, i, \theta_{t-1}, a, j) \stackrel{\text{def}}{=} P(L^1_{t-1} = i, E_{t-1} = n, \Theta_{t-1} = \theta_{t-1}, L^2_t = a, L^1_t = j | O)$$

For the vertical transition vector, we condition on $E_{t-1} \neq n$, meaning that we just exited from a sub-HMM at the previous

step:

$$V(t,e,a,j) \stackrel{\text{def}}{=} P(E_{t-1}=e, \theta_{t-1}, L_t^2=a, L_t^1=j|O),\ e \neq n$$

Finally, for the exit probabilities:

$$X(t,j,\theta_t,a,e) \stackrel{\text{def}}{=} P(L_t^1=j, \theta_t, L_t^2=a, E_t=e|O)$$

### 4.2. Estimating the observation model

The robot can learn about the appearance of the world simultaneously in all four directions: if it knows which way it is facing, it can map its local observations into world-centered coordinates. For example, the probability of observing $y$ when facing North in cell $j$ can be estimated by counting the (expected) number of times $y$ is observed in front when in $j$ and facing north, plus the number of times $y$ is observed to the left when in $j$ and facing east, etc. We also sum up these counts over all time to compute the following expected sufficient statistic:

$$\begin{aligned} B(t,a,j,N,y) = & \\ & P(L_t^2=a, L_t^1=j, \Theta_t=N, Y_t^F=y|O) \\ & + P(L_t^2=a, L_t^1=j, \Theta_t=E, Y_t^L=y|O) \\ & + P(L_t^2=a, L_t^1=j, \Theta_t=S, Y_t^B=y|O) \\ & + P(L_t^2=a, L_t^1=j, \Theta_t=W, Y_t^R=y|O) \end{aligned}$$

Normalizing this yields the maximum likelihood estimate:

$$\hat{B}(a,j,N,y) = \frac{\sum_{t=1}^{T} B(t,a,j,N,y)}{\sum_{y'} \sum_{t=1}^{T} B(t,a,j,N,y')}$$

The appearance of the other directions are estimated similarly. (Note that we are conflating the true appearance of the world with the sensor noise model; separating these can yield still lower sample complexity, at the cost of a harder learning problem.)

## 5. Experimental results

To investigate the advantages of learning H-POMDPs represented as DBNs, we performed experiments using a simulation environment shown in Figure 6. First we created artificial data using a good hand-coded model (which we call the "original" model ) for which all transitions to the correct state were set to $0.9$ and the rest of the probability mass, $0.1$, was divided between self and overshoot-by-one transitions. For the sensor models we initialized the probabilities of perceiving the correct observation to be $0.99$ and all probabilities of perceiving an incorrect observation to be $0.01$. The control policy was a random walk.

We then used the artificial data to train four different models: a flat POMDP, a hierarchical POMDP, a hierarchical POMDP with factored orientation, and a hierarchical POMDP with factored orientation and for which we also do parameter tying on the observation model (as in Section 4.2). We used "uniform" initial models for training, where all non-zero parameters in the "original" model were changed to uniform (e.g., $\langle 0.1, 0.9 \rangle$ was changed to $\langle 0.5, 0.5 \rangle$). Also, we used a uniform Dirichlet prior on all non-zero probabilities, to prevent us from incorrectly setting probabilities to zero due to small training sets.

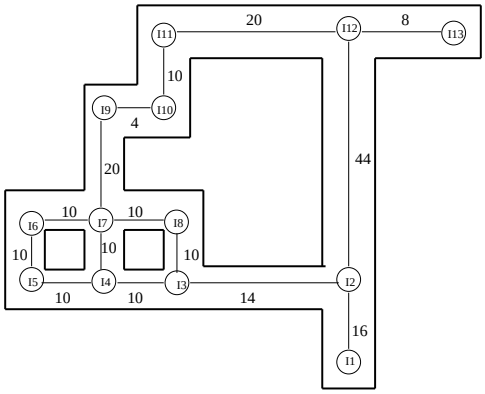


*Figure 6.* The corridor environment used in our simulation experiments. The circles represent abstract states (junctions). The numbers next to the lines represent distance in meters. Each concrete state represents a $2m \times 2m$ square. The model compiles to 595 H-POMDP states.

For testing, we created 500 sequences of length 30 by sampling the "original" model using a policy that performs a random walk. We evaluated the different models using three different measures: log-likelihood relative to the original model, localization accuracy, and computation time.

### 5.1. Log-likelihood

We meaured the "distance" between a learned model, with parameters $\lambda_L$, and the generating (original) model, with parameters $\lambda_G$, using the relative log-likelihood (Juang & Rabiner, 1985): $D(\lambda_L, \lambda_G) = (\log P(y_{1:T}|u_{1:T}, \lambda_L) - \log P(y_{1:T}|u_{1:T}, \lambda_G))/T$.

If $D(\lambda_L, \lambda_G) > 0$, then $P(y_{1:T}|u_{1:T}, \lambda_L) > P(y_{1:T}|u_{1:T}, \lambda_G)$, meaning that the learned model has higher likelihood than the generating model, which usually indicates overfitting.

Our results are shown in Figure 7. It is clear that the hierarchical models require much less data than the flat model. Furthermore, factoring and parameter tying help even more.

### 5.2. Localization

To assess the ability of the models to estimate the robot's position after training (a more relevant criterion than likeli-

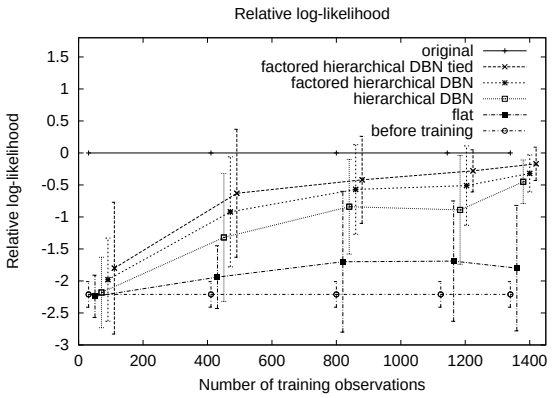


*Figure 7.* Relative log-likelihood for the four models averaged over 500 test sequences. The curves are as follows, from bottom to top: flat, hierarchical, factored hierarchical, factored hierarchical with tying, The bottom flat line is the model before training, the top flat line is the generating model.

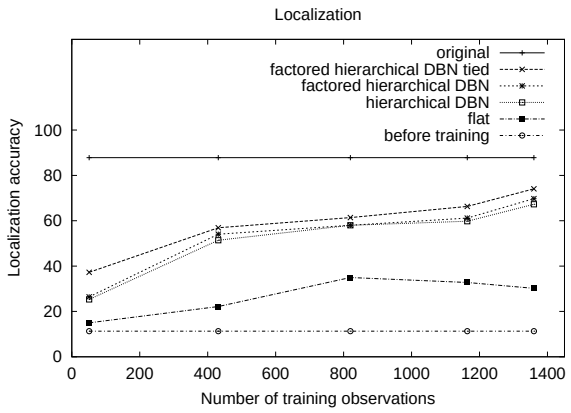


*Figure 8.* Percentage of time the robot estimates its location correctly (see text for precise definition), as a function of the amount of training data. The curves are as follows, from bottom to top: flat, hierarchical, factored hierarchical, factored hierarchical with tying. The bottom flat line is the model before training, the top flat line is the generating model.

hood), we computed the total probability mass assigned to the true state sequence (which is known, since we generated the data from a simulator): $\sum_{t=1}^{T} b_t(s)$, where $b_t(s) = P(X_t = s|y_{1:t}, u_{1:t})$ is the belief state at time $t$. If the belief state was a delta function, and put all its mass on the correct state, then $\sum_{t=1}^{T} b_t(s) = T$; this is the score that an algorithm with access to an oracle would achieve. The best score that is achievable without access to an oracle is obtained by computing the belief state using the original generating model; this score is 87%. The scores of the other algorithms are shown in Figure 8. (This is the total probability mass assigned to the true states in all the test sequences.)

The rank ordering of the algorithms is the same as before: the flat model performs the worst, then hierarchical, then factored hierarchical, and the factored hierarchical with tying is the best.

### 5.3. Speed

To investigate the scalability of our algorithm, we also plotted the time per training epoch with respect to the length of the training sequence (see Figure 9). The results are shown in Figure 9. The $O(T^3)$ behavior of the original H-POMDP algorithm is clear; the flat and DBN algorithms are all $O(T)$, although the constant factors differ. The flat algorithm was implemented in C, and therefore runs faster than the hierarchical DBN (although not by much!); the factored hierarchical DBN is the fastest, even though it is also implemented in Matlab, since the state space is much smaller.

### 5.4. Real robot results

To verify the applicability these ideas to the real world, we conduced an experiment with a B21R mobile robot. We

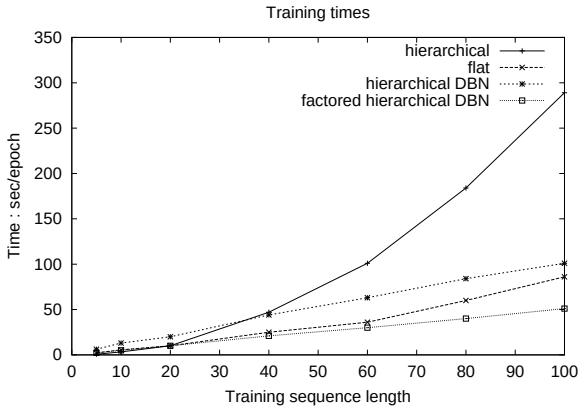


*Figure 9.* Running time (in elapsed user seconds) vs sequence length for the different algorithms. The curves from bottom to top are: factored hierarchical DBN (Matlab), flat (C), hierarchical DBN (Matlab), original hierarchical H-POMDP (C).

created a topological map by hand of the 7th floor of the AI lab; this has about 600 production states (representing $1m \times 1m$ grid cells), and 38 abstract states (representing corridors, junctions and open space). We manually joysticked the robot around this environment, and collected data using a laser-range finder. The data was classified using a neural network to indicate the presence of a wall or opening on the front, left, right and back sides of the robot. We then created several initial models, based on the true topology but with unknown (uniform) observations, and tried learning their parameters from several training sequences totalling about 600 observations (corresponding to about 3 laps around the lab).

The results are qualitatively similar to the simulation results.

However, a lot depends on the quality of the initial model: if we start from a good initial model, there is little left to learn, so all methods perform similarly. The tied model always does significantly better, however, since it is able to learn the appearance of all 4 directions in a single pass through a corridor.

## 6. Conclusions and future work

We have shown how to represent H-POMDPs as DBNs, and demonstrated that this allows us to learn large models with less data and much less time than previously needed. We can further reduce the time by using approximate inference, and can further reduce the sample complexity by doing yet more parameter tying (e.g., factoring the observation model into the appearance of the world and the sensor noise, and factoring the motion model into the location of obstacles and the motor reliability).

The main open problem is to learn the structure of the environment. In this paper, we assumed the topology was known. In (Theocharous & Mahadevan, 2002b), they show that it is possible to learn the structure at the abstract level just by doing parameter estimation in the H-POMDP: the initial abstract transition matrix was almost fully interconnected (ergodic), but with the constraint that corridor states could only connect to junction states, and vice versa. The resulting parameterss, when thresholded, recovered the correct topology. (By contrast, attempts to learn flat HMM topology using EM have generally been considered unsuccessful, with the notable exception of (Brand, 1999), who uses a minimum entropy prior to encourage structural zeros.)

Rather than attempting to learn the structure at the concrete level, we can recognize that all sub-HMMs that model corridors have the same left-right structure; the only uncertainty is the length and the appearance (assuming a tied motion model). Similarly, sub-HMMs that model junctions consist of a single state, and only differ in their appearance. Learning this kind of structure shares some similarity with approaches to unsupervised learning of hierarchical structure in language (see e.g., (de Marcken, 1996)), which we hope to explore in the future.

## References


Binder, J., Murphy, K., & Russell, S. (1997). Space-efficient inference in dynamic probabilistic networks. *Intl. Joint Conf. on AI*.

Boyen, X., & Koller, D. (1998). Tractable inference for complex stochastic processes. *Proc. of the Conf. on Uncertainty in AI*.

Brand, M. (1999). Structure learning in conditional probability models via an entropic prior and parameter extinction. *Neural Computation, 11*, 1155–1182.

Cassandra, A., Kaelbling, L. P., & Kurien, J. (1996). Acting under uncertainty: Discrete Bayesian models for mobile-robot navigation. *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*.

de Marcken, C. (1996). *Unsupervised language acquisition*. Doctoral dissertation, MIT AI lab.

Dissanayake, M. G., Newman, P., Clark, S., & Durrant-Whyte, H. F. (2001). A Solution to the Simultaneous Localization and Map Building (SLAM) Problem. *IEEE Trans. Robotics and Automation, 17*, 229–241.

Fine, S., Singer, Y., & Tishby, N. (1998). The hierarchical Hidden Markov Model: Analysis and applications. *Machine Learning, 32*, 41.

Hu, M., Ingram, C., M.Sirski, Pal, C., Swamy, S., & Patten, C. (2000). *A Hierarchical HMM Implementation for Vertebrate Gene Splice Site Prediction* (Technical Report). Dept. Computer Science, Univ. Waterloo.

Juang, B. H., & Rabiner, L. R. (1985). A probabilistic distance measure for hidden Markov models. *AT&T Technical Journal, 64*, 391–408.

Jurafsky, D., & Martin, J. H. (2000). *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*. Prentice-Hall.

Koenig, S., & Simmons, R. (1998). A robot navigation architecture based on POMDP models. In D. Kortenkamp, R. Bonasso and R. Murphy (Eds.), *Artificial intelligence and mobile robots: case studies of successful robot systems*. MIT Press.

Murphy, K., & Paskin, M. (2001). Linear time inference in hierarchical HMMs. *Advances in Neural Info. Proc. Systems*.

Murphy, K., & Weiss, Y. (2001). The Factored Frontier Algorithm for Approximate Inference in DBNs. *Proc. of the Conf. on Uncertainty in AI*.

Ostendorf, M., Digalakis, V., & Kimball, O. (1996). From HMMs to segment models: a unified view of stochastic modeling for speech recognition. *IEEE Trans. on Speech and Audio Processing, 4*, 360–378.

Skounakis, M., Craven, M., & Ray, S. (2003). Hierarchical Hidden Markov Models for Information Extraction. *Intl. Joint Conf. on AI*.

Theocharous, G., & Mahadevan, S. (2002a). Approximate Planning with Hierarchical Partially Observable Markov Decision Processes for Robot Navigation. *IEEE Intl. Conf. on Robotics and Automation*.

Theocharous, G., & Mahadevan, S. (2002b). Learning the hierarchical structure of spatial environments using multiresolution statistical models. *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*.

Theocharous, G., Rohanimanesh, K., & Mahadevan, S. (2001). Learning Hierarchical Partially Observed Markov Decision Process Models for Robot Navigation. *IEEE Intl. Conf. on Robotics and Automation*.

Thrun, S., Burgard, W., & Fox, D. (1998). A probabilistic approach to concurrent mapping and localization for mobile robots. *Machine Learning, 31*, 29–53.

Zweig, G., & Padmanabhan, M. (2000). Exact alpha-beta computation in logarithmic space with application to map word graph construction. *Proc. Intl. Conf. Spoken Lang.*.