Article draft, please do not cite or distribute.
To appear as: Zubek, R. 2006. Introduction to Hidden Markov Models. In Rabin, S. (ed.), *AI Game Programming Wisdom 3.* Charles River Media, Hingham, MA.

Introduction to Hidden Markov Models

Robert Zubek—Electronic Arts / Maxis
robert.zubek@alumni.northwestern.edu

Stochastic processes are commonly used to estimate and track activity based on limited or noisy information. A number of interesting types of activity can be modeled using stochastic processes, including movement through physical space, the production of gestures or spoken language, and even participation in social interaction.

This article introduces hidden Markov models, an inexpensive and intuitive method for modeling stochastic processes. The following sections present motivation behind the technique, examples of how it can be used to track a player's movement and behavior based on scattered and uncertain observations, as well as details of a computationally efficient implementation.

Motivation

My dog is a transparent beast—I don't even need to be paying attention to know what he's up to. As I sit in front of the TV, I only need to hear his claws tapping on the hardwood floor, a quiet cling of metal, the sound of a plastic bag rustling, and I know immediately what he's been doing. He walked through the dining room to the kitchen, drank some water, noticed the cabinet door was ajar, and went over to sniff around the bag of dog food. And I know immediately that he's going to come back in a moment, sit down right before me, and start begging for a treat, because that's what he does after he notices that there's food around.

How I arrived at these conclusions is neither unusual nor noteworthy—we do this kind of reasoning all the time, without even noticing it. I recognized my dog's movement through the apartment because I know its layout, and I could match what I heard against what I know; first, the hardwood floor sound located him in the dining room, then the metal water bowl located him in the kitchen, making it easy to figure out his general path. Similarly, the snack-begging behavior is typical, and goes through fixed steps; his standard progression from noticing food to coming over and begging is so predictable that I only needed to recognize the first step of the protocol to know what happens next.

We can mimic some aspects of this kind of reasoning in an artificial agent. In particular, if the activity exhibits strong underlying structure, we can use this structure to recognize where we are in the activity, how it progressed, and how it might proceed.

Processes with a strong underlying structure invite easy simplification into finite state spaces. But for many natural processes, standard deterministic finite-state machines are not enough. Even if we know the state space, the current state of the process can rarely be known for certain, and will have to be estimated from whatever evidence is available. Moreover, we need to deal with the possibility that the evidence for this estimation will usually be noisy, incomplete, or incorrect.

Noisy processes can be successfully estimated using stochastic techniques. Hidden Markov models (HMMs), in particular, have been successfully used to track finite-state processes based on noisy evidence. For example, in speech recognition HMMs track the production of words as movement through a space of phonemes and sounds, based on very noisy evidence but very concrete transition rules; in robot navigation HMMs model the movement of a robot through physical space based on occasional and potentially erroneous sensor readings, and so on. The same approach can also be used to track processes in games, such as the movement of the player through physical or abstract state spaces.

Generally speaking, a Markov model can be visualized as a finite state machine with probabilistic edges. In a *hidden* Markov model, the state of the process is not directly observable, so we can never be sure of the current state of the process. Instead, the HMM includes a probability function that matches states with some kind of observable evidence—this allows us to estimate the current state based on a history of evidence observations.

The following sections explain the details of this approach. First, Markov models and hidden Markov models will be presented as theoretical entities, and it will be shown how the state of a model can be estimated from the model definition and a history of observations. Second, the implementation of an HMM will be described, including an optimization into a sequence of simple matrix operations. Finally, its behavior will be demonstrated, using the example of estimating a player's movement through physical space, followed by a description of how HMMs can be used to track movement in conversational space.

Hidden Markov Models

Before discussing hidden models, let's recall the definition of fully-observable *n*-gram Markov models. The following will only be a quick refresher, since *n*-grams have already been discussed in previous AI Game Programming Wisdom articles [Laramée02, Hutchens02].

N-gram Markov Models

A Markov model consists of a set of states and a transition function; some authors also include an initial probability distribution. The transition function represents the probability of the process being in some new state given a history of previous states. The size of the history window determines the model's order: in a first-order model (or bigram), the current state only depends on the immediately previous state, in a second-order model (or trigram), it depends on the last two states, and so on.

First-order models can be conveniently described as probabilistic finite state automata, $M = \langle S, p \rangle$ such that:

- S is the set of states in the process, and
- p is the transition probability function, where $p(s_t \mid s_{t-1})$ signifies the probability of transition from state s_{t-1} to state s_t.

The probabilities associated with all out-edges add up to one, or $\forall s_{t-1} \sum_{s_t \in S} p(s_t \mid s_{t-1}) = 1$.

Figure 1a illustrates a first-order Markov model of a simple agent's internal state.

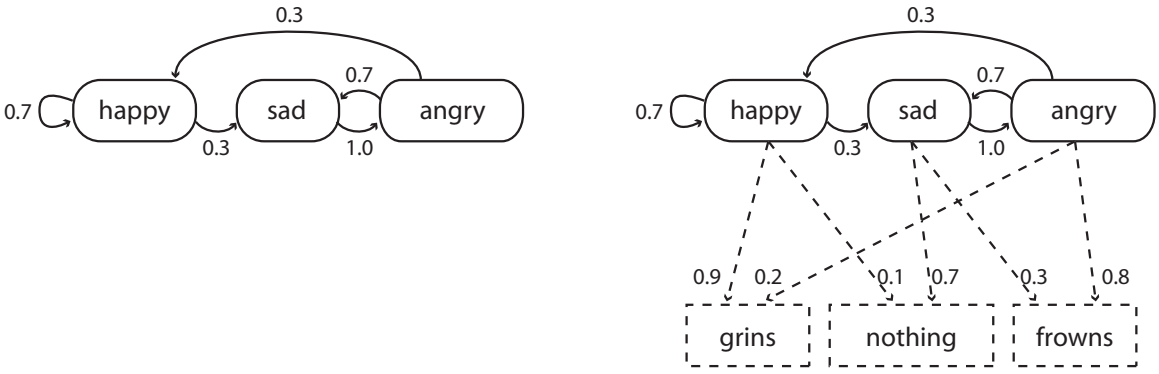


Figure 1. Representations of a Markov model for a simple agent behavior: a) bigram model on the left; b) model with action production probabilities for each state on the right.

Hidden Markov Models

When the state of a process cannot be inspected directly, it must be *estimated* from some sequence of observations. For example, the emotional state of another agent cannot be inspected without peeking into its head, but the emotional state is responsible for the agent's actions—so we should be able to estimate the agent's inner state by observing what it is doing.

Hidden Markov models are used to represent processes that are not fully observable. They augment the *n*-gram model with a set of actions that can be observed, and a probabilistic mapping between actions and states.

A first-order HMM is a tuple $M = \langle S, A, p, q \rangle$ where:

- S is the set of states in the process,
- A is the set of actions that can be observed,
- p is the transition probability function, where $p(s_t \mid s_{t-1})$ signifies the probability of transition from state s_{t-1} to state s_t, and
- q is the action observation probability function, where $q(a_t \mid s_t)$ denotes the probability of observing action a_t at time t given state s_t.

Figure 1b shows an HMM for a simple agent. Dashed boxes and arrows represent actions and action observation probabilities, respectively.

HMM Belief Estimation

To estimate the current state of a process, we calculate the *probability* of being at some state s_t after observing a sequence of actions $a_1,...,a_t$, which we can write as $b(s_t) = p(s_t \mid a_1,...,a_t)$. Doing this for every possible state s_t gives us a probability distribution over the entire state space.

This estimation is performed iteratively, using the well-known *forward algorithm* [Jelinek97]. Let's assume that we know where the process starts out, that is, we know the value of the initial belief distribution $b(s_0)$. Given some action observation a_t, and the belief distribution from the previous iteration, we compute the new belief distribution over all states $s_t \in S$ as follows:

$$b(s_t) = q(a_t \mid s_t) \sum_{s_{t-1} \in S} p(s_t \mid s_{t-1}) b(s_{t-1}) \eta$$

In other words, the probability of being in some state s_t is a product of:

1. the probability *q* of observing action a_t in the state s_t,
2. the probabilities of having been at other states in the last iteration, times the probability of having transitioned over to s_t, all added up, and
3. the normalization constant η, which ensures that $\sum_{s_t \in S} b(s_t) = 1$.

For example, Figure 2 shows the belief computation for the *sad* state, after the *frowns* action was observed. Given that the previous belief values are as shown, the new belief probability for the *sad* state is:

$$\begin{aligned} b(sad) &= q(frowns \mid sad)\left[p(sad \mid happy) b(happy) + p(sad \mid angry) b(angry)\right] \eta \\ &= 0.3 * \left[0.3 * 0.1 + 0.7 * 0.7\right] \eta \end{aligned}$$

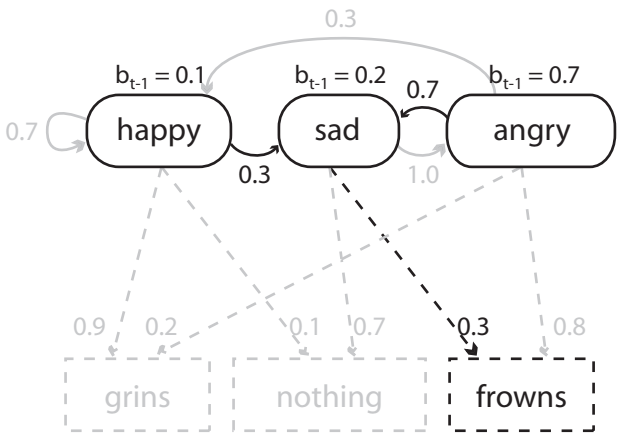


Figure 2. The computation of the probability of the state *sad* from Figure 1b, given some previous belief values, and the observation of the *frowns* action. Elements not used in the computation are grayed out.

Repeating this calculation for every state in the state space results in a belief distribution over the space, showing how likely the process is to be in each state.

Implementation

HMM estimation lends itself to very efficient implementation as a series of matrix operations. The representation of model parameters as matrices is illustrated in Figure 3.

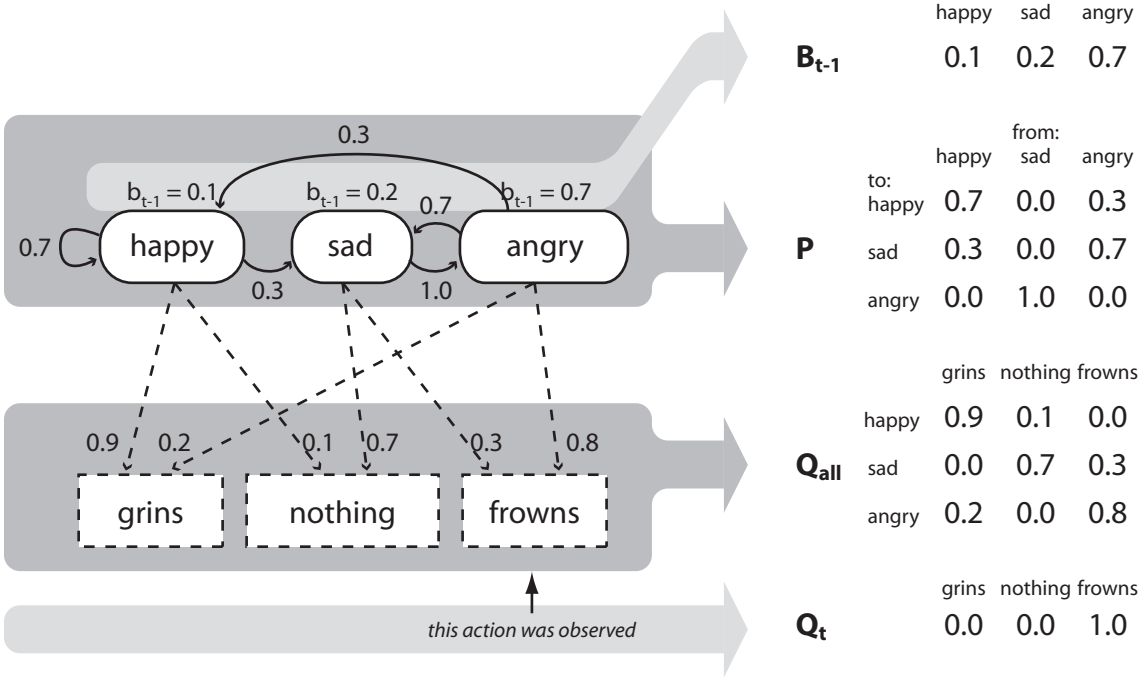


Figure 3. HMM parameters from Figure 1b, stored as vectors and matrices.

In particular, suppose we have an HMM with *n* states and *m* actions. The last known belief distribution b_{t-1} is defined over the *n* states, so we can store it in an *n*-element vector $\mathbf{B_{t-1}}$; similarly with b_t, the new distribution we are trying to estimate, which will be stored as the vector $\mathbf{B_t}$. The transition probability from a source state to a destination state can also be represented straightforwardly as a matrix **P** of size $n \times n$. For the model in Figure 3, these would be as follows:

$$\mathbf{B_{t-1}} = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.7 \end{bmatrix} \quad \textit{(previous belief distribution)}$$

$$\mathbf{P} = \begin{bmatrix} 0.7 & 0 & 0.3 \\ 0.3 & 0 & 0.7 \\ 0 & 1 & 0 \end{bmatrix} \quad \textit{(state transition probabilities)}$$

The action observation function *q* produces a probability of observing a particular action given some particular state. We will store it in an *n*-element vector **Q**, though it's also convenient to think of it as a composite: $\mathbf{Q} = \mathbf{Q_{all}}\ \mathbf{Q_t}$, where $\mathbf{Q_{all}}$ is a matrix of size $n \times m$, containing all the possible state and action probabilities, and $\mathbf{Q_t}$ is an *m*-element observation vector that "selects" data from the matrix based on which action was observed. Using Figure 3 as our example, these become:

$$\mathbf{Q_{all}} = \begin{bmatrix} 0.9 & 0.1 & 0 \\ 0 & 0.7 & 0.3 \\ 0.2 & 0 & 0.8 \end{bmatrix} \quad \textit{(all action observation probabilities)}$$

$$\mathbf{Q_t} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad \textit{(the action selector, signifying the} \text{ frowns } \textit{action was observed)}$$

Belief distribution computation can now be performed as a series of matrix operations. Using * to denote component-wise vector multiplication, the un-normalized belief distribution computation can be compacted into the following expression:

$$\mathbf{B'_t} = (\mathbf{P}\,\mathbf{B_{t-1}}) * (\mathbf{Q_{all}}\,\mathbf{Q_t})$$

We normalize this distribution by dividing every belief value by the sum of all belief values, thus ensuring that the components of the **B** vector add up to one.

$$\eta = \frac{1}{\sum_{s \in S} B_s}$$

$$\mathbf{B_t} = \eta\ \mathbf{B'_t}$$

Using matrix notation, the belief computation in the above example becomes:

$$\mathbf{B'_t} = (\mathbf{P}\,\mathbf{B_{t-1}}) * (\mathbf{Q_{all}}\,\mathbf{Q_t}) = \begin{bmatrix} 0.28 \\ 0.52 \\ 0.2 \end{bmatrix} * \begin{bmatrix} 0 \\ 0.3 \\ 0.8 \end{bmatrix} = \begin{bmatrix} 0 \\ 0.156 \\ 0.16 \end{bmatrix}$$

$$\eta = \frac{1}{0.156 + 0.16} = 3.165$$

$$\mathbf{B}_t = \eta\, \mathbf{B'}_t = \begin{bmatrix} 0 \\ 0.494 \\ 0.506 \end{bmatrix}$$

The new distribution $\mathbf{B}_t$ is the latest estimation of the state of the process. In the next iteration, $\mathbf{B}_t$ will be copied into $\mathbf{B}_{t-1}$, and the belief distribution computation will begin anew.

Performance Considerations

The observant reader will notice that the state estimation cost is $O(|S|^2)$. This becomes prohibitive for very large state spaces, and requires additional optimizations. If an approximate solution is acceptable, one can estimate large models very efficiently using particle filters [Berertonn04]. These work by sampling only a part of the state space, based on the distribution of the belief probability.

Alternatively, we can optimize the forward algorithm computation. In many practical cases the transition matrix will be rather sparse, so standard optimizations of sparse matrix multiplication [Press92] can be applied. In addition, if action observation is always Boolean, the computation of $\mathbf{Q}$ can be reduced to a simple column extraction from the $\mathbf{Q}_{all}$ matrix.

Examples

Hidden Markov models provide an elegant representation for the particular class of processes that exhibit finite state structure, whose state cannot be observed directly, but which produce some possibly noisy evidence that can be used to estimate the state. The following examples illustrate the use of HMMs in two particular scenarios: estimating a player's movement through physical space based on limited sensory evidence, and estimating a player's movement through a dialog based on what they say.

Tracking Movement in Physical Space

HMMs can be used to mimic human tracking of movement through physical space, based on only a modicum of evidence. In the example given in the introduction, I didn't need to look around to know when my dog was going to his water bowl in the kitchen. I only needed to hear a few familiar noises, and I could tell exactly where he was, and which path he took to get there. To mimic this process with a computer, we can represent movement as a stochastic process: model the layout of the physical space as a set of states and explicating a set of actions that will serve as evidence for the states.

For example, suppose we have a first-person shooter game with an agent hiding inside a level and waiting to jump out at the player. The agent cannot *see* the player, but it can roughly hear what they're doing. Given some knowledge of the layout of the level, and

how the action evidence matches up against it, the agent can efficiently estimate the player's position.

Let's assume the level has the physical layout shown in Figure 4. It's a simple area made up of 27 regions of four types: grass, metal grates, water, and door portals.

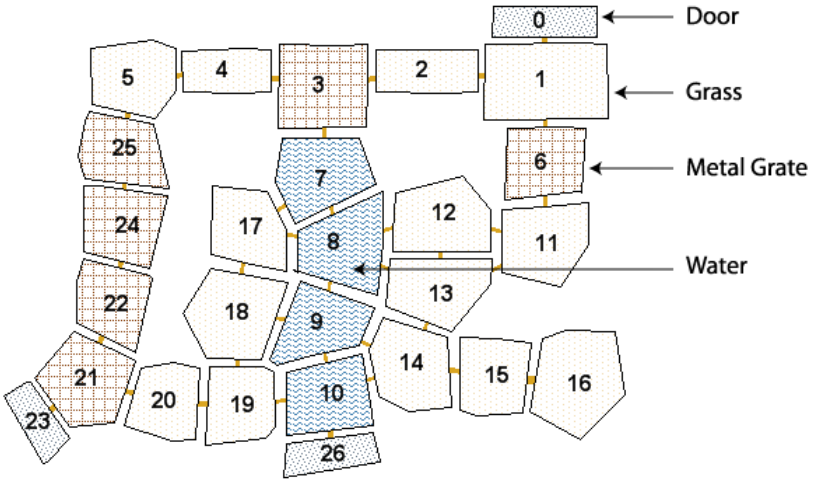


Figure 4. Sample physical space. Includes 27 regions of four different types: grass, water, metal, and door.

Walking on the different terrain types produces different kinds of noises—for example, walking through water tends to produce splashing sounds, while walking on grass usually produces no sound at all. We can allow for considerable ambiguity in the mapping between terrain types and sounds—the corresponding observation probabilities are shown in Table 1.

Table 1. Action observation probabilities for the terrain types.

Action	Grass	Metal	Water	Door
Nothing was heard	0.9	0.1	0	0
Footsteps on metal	0	0.9	0	0
Water splashing	0	0	0.9	0
Door opening	0	0	0	0.9
Unclear noise	0.1	0	0.1	0.1

Please notice that the action observation probabilities are not unique to particular types of terrain—the unclear sounds can serve as evidence for a number of terrain types, as can silence. Similarly, there are numerous regions of each type in the level (e.g., six different metal grate areas), so some actions serve as evidence for all of them at the same time.

Transition probabilities have been defined so that, for a state with n neighbors, the probability of a transition to each neighbor is $0.9/n$, and the probability of the state remaining unchanged (that is, of the player not moving at all) is 0.1.

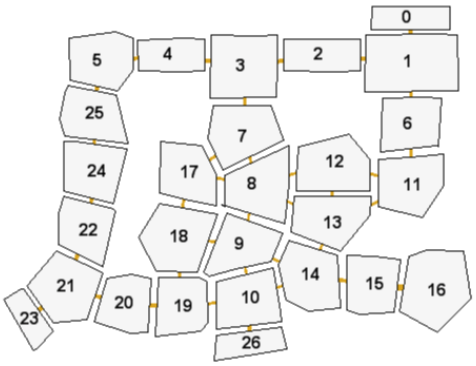
A sample localization task is presented in Figure 5. In this example, the system was presented with a sequence of only three action observations:

1. Some unclear noise
2. Sound of footsteps on metal
3. Sound of water splashing

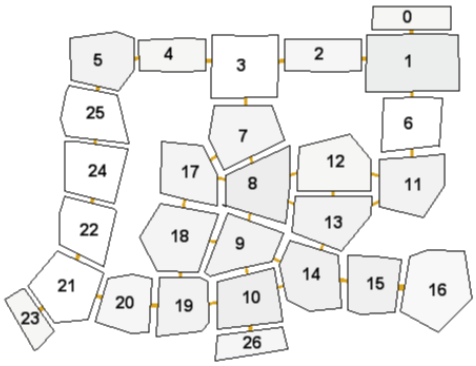
These few observations are enough for the system to figure out the player's probable location. The initial probability distribution is completely uniform—there is no bias towards any particular starting point. The first observation of some unclear noise constrains the player's possible location to a number of regions, such as those containing grass, water, or a door. The second observation suggests that the player moved onto a metal grate, so the metal regions next to the previous best guesses become the most probable.

Finally, the third observation suggested movement into a water-filled region, and the one water-filled region that lies next to the two locations that were previously most probable becomes the new best candidate. In fact, after the third observation, region number 7 has by far the highest probability associated with it, and is therefore confidently identified as the player's most likely final location.

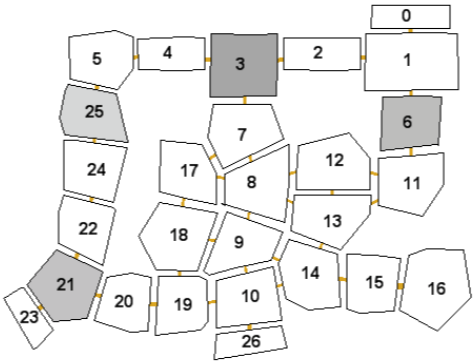
Another example of localization, using the same model, is presented in Figure 6. Here, the series of observations is longer, and we can see the belief distribution lose and gain precision. It is interesting to notice that, even if the distribution is fairly broad, sometimes all it takes is one good observation to narrow it down to just one or two states.



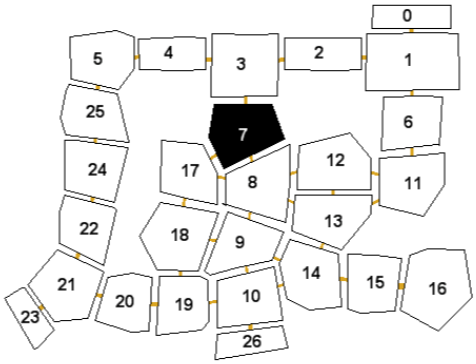
0. Initial probability distribution



0. Initial probability distribution
1. Unclear noise



0. Initial probability distribution
1. Unclear noise
2. Footsteps on metal

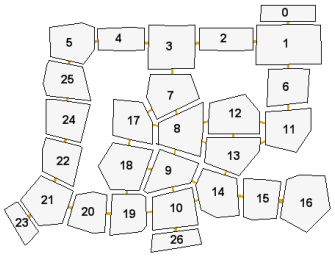


0. Initial probability distribution
1. Unclear noise
2. Footsteps on metal
3. Water splashing

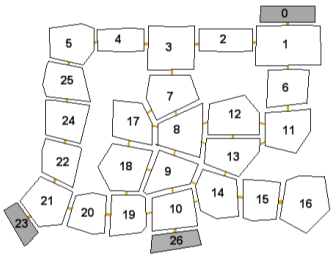
Figure 5. Sample localization progression given a sequence of observations. The darkest regions have the highest belief probabilities.

Sequence of
action observations:

0. Initial State

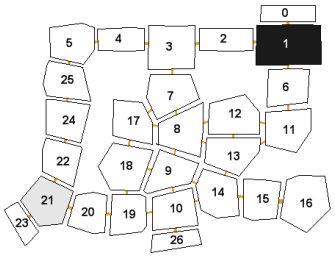


0. Initial State
1. Door opening

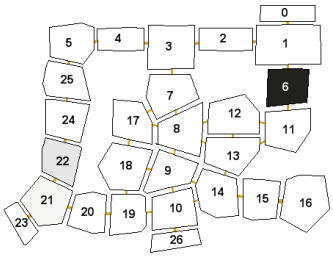


---

0. Initial State
1. Door opening
2. Nothing heard

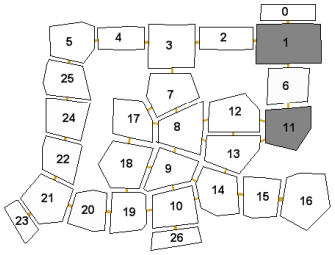


0. Initial State
1. Door opening
2. Nothing heard
3. Footsteps on metal

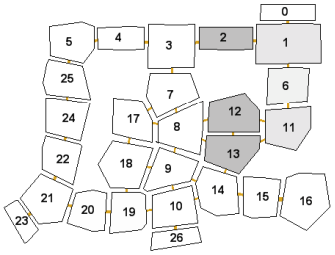


---

0. Initial State
1. Door opening
2. Nothing heard
3. Footsteps on metal
4. Nothing heard

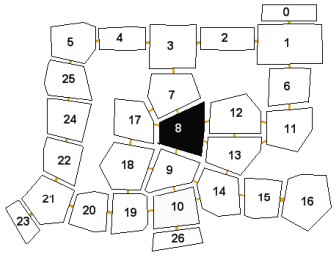


0. Initial State
1. Door opening
2. Nothing heard
3. Footsteps on metal
4. Nothing heard
5. Nothing heard

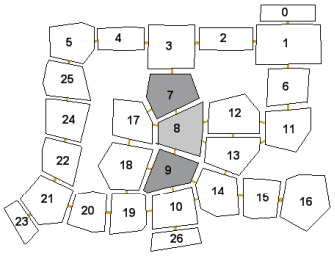


---

0. Initial State
1. Door opening
2. Nothing heard
3. Footsteps on metal
4. Nothing heard
5. Nothing heard
6. Water splashing

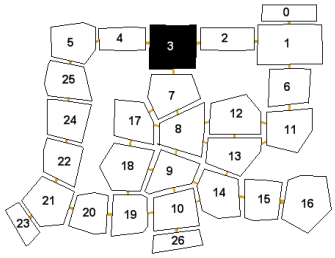


0. Initial State
1. Door opening
2. Nothing heard
3. Footsteps on metal
4. Nothing heard
5. Nothing heard
6. Water splashing
7. Water splashing



---

0. Initial State
1. Door opening
2. Nothing heard
3. Footsteps on metal
4. Nothing heard
5. Nothing heard
6. Water splashing
7. Water splashing
8. Footsteps on metal



0. Initial State
1. Door opening
2. Nothing heard
3. Footsteps on metal
4. Nothing heard
5. Nothing heard
6. Water splashing
7. Water splashing
8. Footsteps on metal
9. Water splashing

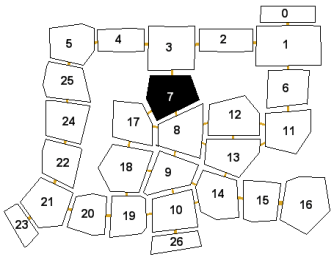


Figure 6. Example of localization based on a series of nine actions.

Non-uniform Probability Functions

The model used in the previous example was based on uniform transition probabilities—each state had the same probability of transitioning to its neighbors, as well as a probability of 0.1 of transitioning back to itself. In practice, however, players tend to have non-uniform preferences for where to move—in first-person shooters, for example, they tend to move away from exposed or transitory areas such as hallways, and stay longer near cover and camping spots. These behaviors can be reflected in appropriate modifications to the transition probabilities.

The example in Figure 7 has been modified to include a camping spot in region number 16. The player is assumed to be unlikely to leave that region, and the transition distribution has been modified to reflect this, by defining $p(s_{16} \mid s_{16}) = 0.9$ and $p(s_{15} \mid s_{16}) = 0.1$. Consequently, over time region number 16 "sucks up" the probability distribution.

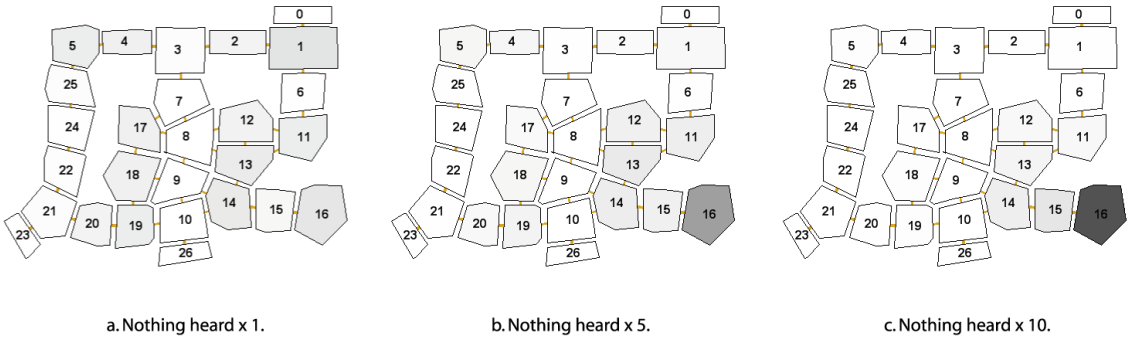


Figure 7. Estimation for a model with non-uniform transition probabilities, and a series of "nothing heard" observations. Belief converges on the region with the high self-transition probability.

Ultimately, the transition probabilities would be best acquired automatically by observing player behavior. The complex topic of how to automatically acquire transition probabilities and other model parameters will not be addressed here; the relevant details can be found in the standard literature on Markov models [Charniak93]. Fortunately, for simple models such as those described here, manual estimation of probabilities is often adequate.

Estimation of Movement through a Social Interaction

Perhaps surprisingly, even something as poorly-defined as social interaction can be modeled using stochastic processes. In this example, we consider the interaction between a player and a computer taking place over natural language utterances.

In many cases of highly structured or familiar interactions, we can impose a specific ordering on how the interactions are supposed to proceed. Familiar situations—buying

things, asking clerks for information, asking for directions, and so on—do not require novel strategies from the participants, and people routinely fall into predictable patterns of behavior when engaging in them. This enables us to develop a finite-state representation of how the interaction is supposed to evolve over time.

Unfortunately, the 'current state' of an interaction cannot be observed directly. However, if the interaction exhibits a stable structure, we can represent it with a finite-state model. This model's state can then be estimated by observing what participants say, and using it as evidence for where they must be in the dialog.

Social interaction, just like many other complex phenomena, can be simplified via hierarchical decomposition. We can decompose the overall interaction into smaller sub-interactions that are responsible for particular small-scale elements—greeting, selling items, answering questions, fulfilling requests, etc. Each of these can be conveniently tracked with a separate HMM. Figure 8 presents an example of a simple HMM responsible for answering requests for assistance.

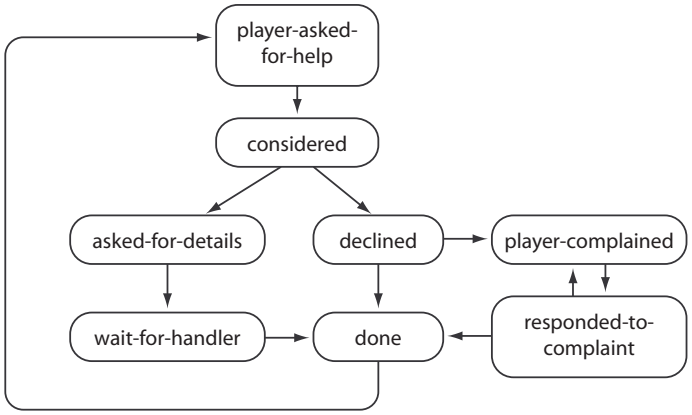


Figure 8. Sample model for handling players' requests for assistance.

The evidence for these HMMs comes from observations of what participants say. However, it would be unfeasible to look for particular sentence forms as evidence—after all, there are many ways of saying the same thing. Instead, it's beneficial to use a layer of linguistic abstraction, such as speech acts, to first translate particular sentences into more general units, and then use those as evidence of position in the dialog.

Additionally, we can increase the robustness of the system by using several different HMMs to track the same interaction element on different levels of generality. For example, a request for assistance can be tracked by three different models at the same time: a very specific *request-for-help* handler, a more general *generic-request* handler, and the broadest *generic-turn-taking* handler—as long as the player follows the expected help request protocol, the most specific handler will track it successfully, but the less specific ones will also track it redundantly, and take over if the most specific one fails.

Running a number of separate redundant models in parallel, but arranged in a control hierarchy, greatly increases the system's robustness.

Tracking stereotyped interactions based on noisy and ambiguous evidence is a matter of estimating state in an interaction state space, and proceeds as in the previous example. Further details on how to track stereotyped interactions using hierarchical parallel models can be found in recent publications [Zubek05a, Zubek05b].

In Closing

Probabilistic methods such as Bayesian networks have proven to be useful in reasoning under uncertainty [Tozour02], and HMMs in particular have been popularly used in a number of recognition and estimation tasks. HMMs are particularly successful at tracking finite-state processes that have well-known structure, whose state is not directly observable, but which produce evidence that can be used to estimate their state. They are also computationally inexpensive, and cope well with noisy and ambiguous evidence.

This article introduced hidden Markov models, and the problem of calculating belief probability distributions using the forward algorithm. It showed how the relevant calculations can be represented as a series of simple matrix operations, and implemented inexpensively in software.

Acknowledgements

Special thanks to Aaron Khoo, John Manslow, and Paul Tozour for their helpful comments on previous versions of this paper.

References

[Bererton04] Bererton, C., "State Estimation for Game AI Using Particle Filters," *AAAI Workshop on Challenges in Game AI,* AAAI Tech Report WS-04-04. AAAI Press, 2004.

[Charniak93] Charniak, E., S*tatistical Language Learning,* MIT Press, 1993.

[Jelinek97] Jelinek, F., *Statistical Methods for Speech Recognition,* MIT Press, 1997.

[Laramée02] Laramée, F. D., "A Rule-Based Architecture Using the Dempster-Shafer Theory," *AI Game Programming Wisdom,* pp. 358-366, Charles River Media, 2002.

[Hutchens02] Hutchens, J., Barnes, J., "Practical Natural Language Learning," *AI Game Programming Wisdom,* pp. 602-614, Charles River Media, 2002.

[Press92] Press, William H., Teukolsky, Saul A., Vetterling, William T., Flannery, Brian P., *Numerical Recipes in C, 2nd ed.,* available online at http://www.library.cornell.edu/nr/, Cambridge University Press, 1992.

[Tozour02] Tozour, P. "Introduction to Bayesian Networks and Reasoning Under Uncertainty," *AI Game Programming Wisdom,* pp. 345-357, Charles River Media, 2002.

[Zubek05a] Zubek, R., Horswill, I. D., "Hierarchical Parallel Markov Models of Interaction", Artificial Intelligence and Interactive Digital Entertainment Conference, also available online at http://www.cs.northwestern.edu/~rob/publications/hierarchical-parallel-markov.pdf, AAAI Press, 2005.

[Zubek05b] Zubek, R. *Hierarchical Parallel Markov Models for Interactive Social Agents.* Ph.D. Dissertation, NU CS Tech Report NWU-CS-05-10, available online at http://www.cs.northwestern.edu/publications/techreports/2005_TR/NWU-CS-05-10.pdf, Computer Science Department, Northwestern University. 2005.