Utile Distinction Hidden Markov Models

Daan Wierstra INFO@WIERSTRA.COM
Marco Wiering MARCO@CS.UU.NL
Institute of Information and Computing Sciences, Utrecht University, 3508 TB Utrecht, The Netherlands

Abstract

This paper addresses the problem of constructing good action selection policies for agents acting in partially observable environments, a class of problems generally known as Partially Observable Markov Decision Processes. We present a novel approach that uses a modification of the well-known Baum-Welch algorithm for learning a Hidden Markov Model (HMM) to predict both percepts and utility in a non-deterministic world. This enables an agent to make decisions based on its previous history of actions, observations, and rewards. Our algorithm, called Utile Distinction Hidden Markov Models (UDHMM), handles the creation of memory well in that it tends to create perceptual and utility distinctions only when needed, while it can still discriminate states based on histories of arbitrary length. The experimental results in highly stochastic problem domains show very good performance.

1. INTRODUCTION

Consider the problem of an explorer robot navigating the surface of the planet Mars. The robot moves about and makes local observations. Since no human operators are available to steer the machine in realtime, the robot must decide for itself what to do given the circumstances, the 'circumstances' being its history of observations and previously taken actions. It cannot disambiguate its current situation and position from the current observation alone, because different situations might produce the same observations. This robot has to deal with the problem of *hidden state* (Lin & Mitchell, 1993), and with *perceptual aliasing* (Whitehead & Ballard, 1991). These problems are caused by the fact that multiple states look the same to the agent, which has limited sensor capabilities. The agent's task, nevertheless, is to find an optimal policy in such an uncertain and ambiguous environment. We generally refer to this sort of problem as a Partially Observable Markov Decision Process or POMDP.

Perceptual aliasing can be viewed as both a blessing and a curse. It is a blessing, because it gives the agent the opportunity to generalize across states — different states that nevertheless produce the same or similar observations are likely to require the same actions. It is a curse, because, for optimal action selection, the agent needs some kind of memory. That is a very serious problem indeed, because, in order to create memory, it has to keep track of its history of actions and observations to make a good estimate of its current world state. Unexpected events, malfunctioning motors and noisy observations make the problem even more difficult. How can this problem possibly be solved?

In this paper we present a novel algorithm, Utile Distinction Hidden Markov Models (UDHMMs). It is based on the construction of anticipatory behavior models using a generalization of Hidden Markov Models (HMMs) (Rabiner, 1989). The nodes of this HMM are used to model the internal state space (hidden states) and the transitions between them are used to represent the actions executed by the agent. Using this model, we can propagate *belief* about internal states during the execution of a trial. During a trial, we update the belief for every internal state at each time step. Building on this probabilistic framework, we modify the Baum-Welch parameter estimation procedure such that it enables the HMM to make *utility* distinctions, which amounts to a search for *relevant* discriminations between states. Superimposed on this model, we use a particular form of Reinforcement Learning (Kaelbling, Littman & Moore, 1996; Sutton & Barto, 1998) to be able to estimate the utility of actions for newly learnt hidden behavior states.

Previous work relevant to UDHMM has been carried out by Chrisman (1992) and McCallum (1993, 1995a,

Appearing in *Proceedings of the 21st International Conference on Machine Learning*, Banff, Canada, 2004. Copyright by the authors.

1995b). Chrisman first used a HMM to predict hidden world state. His Perceptual Distinctions Approach constructs a world model (HMM) that predicts observations based on actions, and can solve a number of POMDP problems. However, it fails to make distictions based on utility — it cannot discriminate between different parts of a world that look the same but are different in the assignment of rewards. He posed the Utile Distinction Conjecture claiming that state distinctions based on utility would not be possible.

McCallum (1993) refuted that conjecture by developing a similar HMM-based algorithm that splits states based on their utility, Utile Distinction Memory (UDM). However, apart from being slow, UDM suffers from a severe limitation: while considering a state split, it only considers the previous belief state, while ignoring longer history traces. This renders certain POMDP problems which involve memory of more than one time step insoluble by UDM. McCallum solved this problem by introducing USM (McCallum, 1995a) and U-tree (McCallum, 1995b). Those algorithms construct variable-depth decision trees in which historical information is stored along the branches of the tree. The branches are split if a statistical test shows that splitting aids the algorithm in predicting future discounted reward (return or utility).

Aberdeen (2003), like McCallum for UDM, also utilizes a HMM as a model, although in quite a different way. His method models the rewards at every HMM node. This algorithm does not lead to very good results, though. This is because of Baum-Welch's difficulty with relating events that take place several time steps from one another, and because no future information is included in the modeling procedure.

We present an approach that can make utility distinctions based on history traces of arbitrary length, while at the same time constructing a behavior model that can be used by any POMDP reinforcement learning method. The agent using UDHMM is able to cope with noisy observations, actions, and rewards, and uses Baum-Welch to decide which features of the environment are relevant to its task. Furthermore, the algorithm is simple and performs well in the number of steps required for the agent to learn its task.

In section 2 we outline our utile distinctions approach and briefly describe the algorithm. In section 3 we elaborate on the details of UDHMM. Section 4 presents experimental results on two problems, where we show a very good performance of the UDHMM approach, especially in a highly stochastic domain. In section 5 we discuss the results and leave room for some speculation.

2. UTILE DISTINCTION AND HIDDEN MARKOV MODELS

The behavior model used by UDHMM is a generalization of the Input-Output Hidden Markov Model (IOHMM) (Bengio & Frasconi, 1995), an extension of the standard HMM, in which the agent's actions serve as input signals. The IOHMM in our case extends the standard HMM by allowing for transition probabilities between states conditioned by the actions. In this model, we represent actions as transition probabilities between states and we model observations for every node in the IOHMM.

So, using this IOHMM framework, how do we introduce the necessary utile distinction? It is important to allow for history traces of length more than one, in order to provide enough flexibility to cope with a complex environment. Simple environments may contain landmark observations that indicate utility to the agent. UDM is able to distinguish those. But if a landmark consists of a sequence of events instead of one single observation, UDM would have considerable difficulty with the task, or might even fail to solve it.

But, not only do we want to be able to discriminate between longer history traces, we want to avoid unnecessary memory bookkeeping that slows down the performance, and might produce overfitting. We want model memory to be created where needed, going back as long as needed.

For an elegant solution, we turn to the Baum-Welch procedure itself. We define *return* to be an approximation to expected utility, the *future discounted reward*. By including the return at every time step in the observation vector of every time step, we let Baum-Welch search for a model that predicts *both* observations and utility, in relation to one another. This encourages the modeling of distinctions predictive of utility.

The general idea behind UDHMM is as follows. We split up learning by constantly repeating two phases after one another, one for online control learning (reinforcement learning during trials), one for offline model learning (model learning between trials). When the agent is online, that is, acting in the world and learning its control policy, the utilities are not known yet. Only after the completion of a trial we can compute what the returns were at every time step. So, during online performance, we simply *ignore* the utility factor. During online execution, the agent uses the forward-pass of Baum-Welch to update its belief over internal states. It observes the environment, chooses an action based on Q-values calculated by a version of SARSA(λ)-learning (Sutton & Barto, 1998), then updates its Q-values according to the perceived results and updated belief.

As soon as a trial is finished, offline learning — model learning between trials — begins. It consists of updating the behavior model, the HMM, this time *including* the now known returns. The returns are handled as being part of the observation, and used in the reestimation of the model parameters. Every internal state now not only models transition statistics for all actions and probability densities for observation vectors, but also probability densities for the perceived returns. Utility becomes part of the model.

UDHMM exploits the interaction between the model-learning and model-solving stages. The idea is that a better model leads to a better policy, and a better policy leads to a better model. Of course, for model-learning, UDHMM should only consider recent histories, since those are produced by the most recent and presumably best policies.

By using Baum-Welch, UDHMM manages to create a behavior model that uses memory that can span multiple time steps. By including the utility in the observation, Baum-Welch is enabled to predict utility based on history, creating memory where needed. In order to be able to do this, the system needs spare states to be used for the creation of memory. By splitting states whose utility distributions are not Gaussian according to a statistical test, the algorithm creates its own state space to craft its utility predictions.

3. ALGORITHM DETAILS

The Utile Distinction Hidden Markov Model consists of a finite number of states $\mathcal{S} = \{s_1, s_2, \ldots, s_N\}$, a finite number of actions $\mathcal{A} = \{a_1, a_2, \ldots, a_K\}$ and a set of possible observations $\mathcal{O} = \mathbb{R}^{d+1}$. Note that with observations of dimension d, we add one element to the vector — the return — to obtain a vector of length $d+1$. In case of discrete observations, as is the case in our experiments below, the observation vector will be a tuple $\langle n, r\rangle$ where n is a natural number indicating the observation and r a real number indicating the return. For every state s we keep transition probabilities $T(s_j, a_k, s_i)$ to other states for every action. For every state, there is an observation mean vector and a covariance matrix that together describe a Gaussian probability density function $p(o_i|s_j)$. There are also the mean utility $\mu_R(s)$ and the utility variance $\sigma_R(s)$, kept at every node s. The agent's belief at every timestep about hidden state is denoted by the belief vector $\vec{b}_t = \langle b_t(s_1), b_t(s_2), \ldots, b_t(s_N)\rangle$. See Figure 1 for a Bayesian representation of the model.

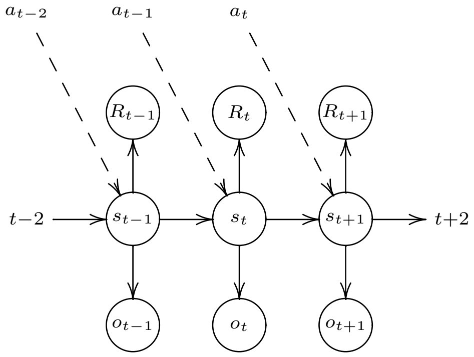

Figure 1. A dynamic Bayes net diagram representing graphically the relations between the different parts of a UDHMM. The influence of actions a_t is represented by dashed arrows.

We compute the belief of the agent at every time step by using the forward-pass of Baum-Welch:

$$b_t(s_i) = \eta \cdot p(o_t|s_i) \sum_j T(s_j, a_{t-1}, s_i) b_{t-1}(s_j)$$

where η is a normalization constant to ensure the belief values correctly add up to 1.

In this paper we use a highly approximate reinforcement learning algorithm for adjusting the agent's policy: BARBA(λ)-learning. Since we face non-deterministic environments in which state certainty is unavailable and the only information we have is a belief vector over internal states, we need to design a method to deal with this uncertainty. At every node s, for every action a, we store Q-values $q(s,a)$. Then, we define the Q-value of a particular action a for a particular belief vector $\vec{b}_t$ to be a result of a weighting process of all states:

$$Q(\vec{b}_t, a) = \sum_i b_t(s_i) q(s_i, a)$$

Using the belief vector, we let the agent update its Q-values by BARBA(λ)-learning, an approach similar to that of Loch and Singh (1998) but generalized to the linear case like in Littman, Cassandra and Kaelbling (1995). It uses eligibility traces $e_t(s,a)$ to update state-action values. On experiencing experience tuple $\langle \vec{b}_t, a_t, r_t, \vec{b}_{t+1}, a_{t+1}\rangle$ the following updates are made:

$$\forall (s, a \neq a_t): \quad e_t(s,a) := \gamma\lambda e_{t-1}(s,a)$$

$$\forall (s): \quad e_t(s,a_t) := \gamma\lambda e_{t-1}(s,a_t) + b_t(s)$$

$$\delta_t = r_t + \gamma Q(\vec{b}_{t+1}, a_{t+1}) - Q(\vec{b}_t, a_t)$$

$$\forall(s,a): \quad \triangle q(s,a) := \alpha e_t(s,a)\delta_t$$

where α is the learning rate and γ the discount factor.

So far we have described the online control-learning part of the learning algorithm. Now we consider the offline part, anticipatory model learning. We use Baum-Welch to update the model parameters such as the transition probabilities $T(s_j, a_k, s_i)$, initial occupation probabilities $\pi(s)$, and the observation probability density function $p(o|s)$ parameters. But we ought to include the utility as well. We define an approximation to expected utility, future discounted reward or *return*, to be:

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^n r_{t+n} + \ldots$$

These are the values we are going to model. For if the *distributions* of returns from one state are significantly different depending on what history preceded it, the state is not a consistent, Markovian state in that it may require a different policy depending on its history. In order to make the state Markovian again concerning utility, UDHMM uses Baum-Welch to relate events with multiple time steps in between. Assuming independence between o_t and R_t, we include the probability density function (pdf) of the return in the observation probability densities for Baum-Welch parameter re-estimation:

$$p(o_t, R_t|s) := p(o_t|s) \cdot \mathcal{N}(R_t, \mu_R(s), \sigma_R(s))^\theta$$

where $\mathcal{N}(x, \mu, \sigma)$ is a function for variable x in a Gaussian bell curve with mean μ and variance σ. θ is a parameter indicating the importance and impact of utility modeling relative to observation modeling. In principle, σ_R is divided by θ in the Gaussian, which lessens or strengthens the sensitivity of the nodes to 'observing' utilities. In our experiments, we found that when θ was too high, the perception modeling degraded too much, while when too low, utility prediction failed. By replacing the observation pdf with the combined normalized observation-utility pdf, we enable the algorithm to use Baum-Welch to make utility distinctions that span multiple time steps. This is in sharp contrast with McCallum's UDM, which only looks at the node active at the previous time step in order to decide whether it should split a state.

As noted before, including the utility in the observation is only done during model learning. During trial execution (model solving), returns are not available yet, since they depend on future events. Therefore, online belief updates are done *ignoring* the utility information.

It should be noted that, as the agent becomes more adept at its task, the utility statistics change for hidden states. As its policy improves, its behavior model should improve, and a better behavior model leads to better policies. However, early experiences tend to be less interesting, and even misleading since utility statistics are still very bad (because of bad policies). This means that Baum-Welch should preferably be performed on the latest trials, since trials from its early history might distort utility modeling. Therefore we only keep a limited history of trials.

In order to be able to predict utility, the algorithm needs enough nodes in the HMM in order to create memory. This means that some kind of heuristic for state-splitting would be appropriate. We choose to split states that do not have a Gaussian return distribution. This means return statistics must be gathered in order to be able to do this test.

Determining whether and how a state should be split involves a statistical test to falsify beyond reasonable doubt that a state's utility distribution is not *normal* (i.e., Gaussian). If a split is performed, the EM algorithm is used to find the best fit to the utility distribution, and state splitting occurs according to the found mixture components. Transition probabilities are distributed evenly among the resulting split states.

We split states after the Chi-Square test on a discretized return distribution for a node shows that the return distribution is not a Gaussian. When this happens, the UDHMM invokes the EM algorithm on the node, where mixtures of Gaussians with various (2, 3 and 4) numbers of components are used to determine the best fit for the distribution. We define 'best fit' to be the Gaussian mixture with the smallest number of components that still offers an acceptable fit. After this is determined, the state is split, where a new state is created for each component. Transition probabilities are distributed equally over the newly created states. After each state-splitting, Baum-Welch is re-invoked to improve the overall model.

This state-splitting leads to enough *excess states* to enable the algorithm to discriminate good policies. It also does lead to some splits that do not enhance performance. However, it is necessary to have excess states in order to be able to discover more complex re-

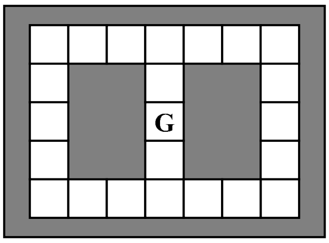

Figure 2. The Hallway navigation maze. The agent randomly starts each trial at one of the four corners. It observes its immediate surroundings and is capable of the actions goNorth, goEast, goSouth, and goWest. It must reach the center (labeled with 'G') to gain a reward.

lationships between events and utilities (compare excess states with fringe nodes in McCallum's U-tree). Also note that this, in our experience, rarely leads to more than twice the number of nodes that is strictly necessary for performing a task.

4. EXPERIMENTAL RESULTS

Utile Distinction Hidden Markov Models have been tested in several environments, of which two are described below. The first environment is a deterministic maze, which has previously been described and successfully solved by McCallum (1995a). The second is a large and extremely stochastic navigation environment with 89 states, for which we show very good results with UDHMM.

4.1. HALLWAY NAVIGATION

The first problem we tested was McCallum's hallway navigation task (McCallum 1995a). In this task (see Figure 2), the agent starts randomly in one of the four corners of the maze. It can only detect whether there is a wall immediately north, east, south, and/or west of itself, so there are $2^4 = 16$ possible observations. It can perform four different actions: goNorth, goEast, goSouth, and goWest, which move the agent in the indicated direction. Its objective is to reach the goal, labeled with 'G' in the figure — it gets a reward of 5.0 there. When attempting to move into a barrier, it gets a punishment of -1.0. All other actions result in a reward of -0.1. We use a temporal discount factor $\gamma = 0.9$.

For a number of settings, we ran 21 experiments, which all consisted of many online trials and offline between-trial Baum-Welch steps. During each offline phase, we ran Baum-Welch only for the last 12 trials in order to save computation time and to prevent the model from settling down in a local suboptimum (world dynamics change over time because the agent's policy changes as well). Of those 21 runs we plotted the one with median performance, where performance is defined as the number of trials it takes for the algorithm to consistently have an average number of steps to the goal under 7.0.

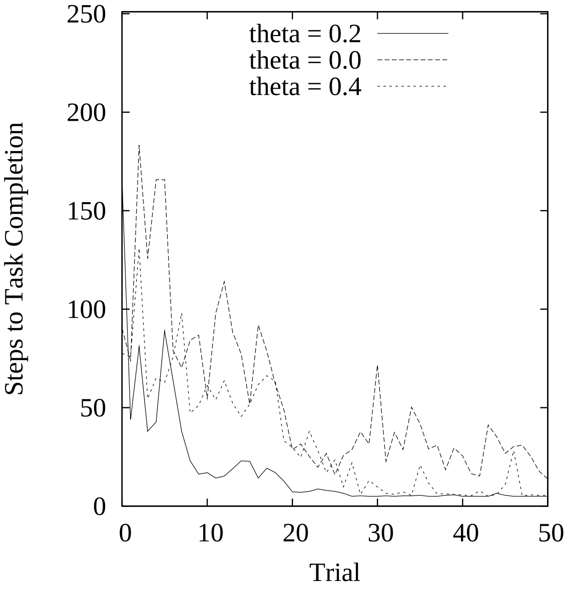

Figure 3. Performance of UDHMM on the Hallway Maze for different values of θ. We plot the number of steps it takes to reach the goal.

In Figure 3 we see the results for the UDHMM algorithm with different values of θ. When $\theta = 0.0$ we observed that in most runs, the algorithm does not reach the success criterion within 100 trials. This is because with $\theta = 0.0$, return is effectively ignored and cannot help with modeling utile distinction. Relying only on observations, the algorithm does not find a suitable model close to the underlying MDP. So clearly, utile distinction is needed to ensure effective behavior modeling.

When we tried $\theta = 0.4$, the algorithm did find a model close to the underlying MDP and good performance in all runs, but later than with $\theta = 0.2$. This is probably due to the fact that, certainly in the first few trials, a too large emphasis on return modeling causes too much modeling of bad policy noise in the return values.

The plot in Figure 4 shows the results of UDHMM with added noise. Noise consisted of a 0.1 probability of performing a random action, a 0.1 probability of observing a random observation and at every step a random number from the range -0.1 to +0.1 added to the reward.

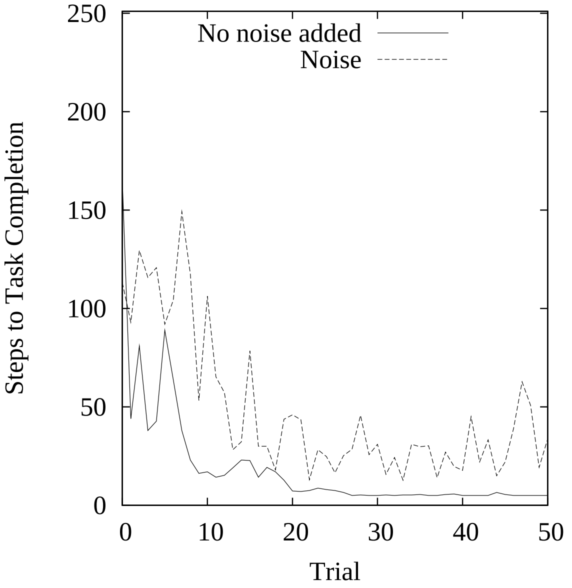

Figure 4. Results for UDHMM with $\theta = 0.2$. Results are shown for UDHMM in a Hallway Maze without noise, and for UDHMM with noise added to the observations, actions and rewards.

When compared to the results of McCallum (1995a), we must concede that his USM achieves success in about 4 times as few trials. We do note, however, that our algorithm is ideally suited to noisy stochastic environments, which USM and U-tree are not. The next problem will illustrate this.

4.2. THE 89-STATE MAZE

This problem is taken directly from Littman, Cassandra and Kaelbling (1995), where seeded Linear Q-learning is applied on the task. It involves a maze in which the agent, next to position, also has an orientation, five actions Forward, TurnLeft, TurnRight, TurnAbout, and doNothing. Like the Hallway problem, here the agent also does not perceive its immediate location or orientation. It only senses one of the $2^4 = 16$ possible observations. It starts each trial at a random location with random orientation, and its goal is to reach the goal labeled 'G', where it gets the task's only reward 1.0. (see also Figure 5). The actions and observations are extremely noisy. For example, the chances of getting a correct observation are only about 70%. Also, the actions are only successful with a probability of about 70%.

Seeded Linear Q-learning's (Littman, Cassandra and Kaelbling, 1995) good performance can be explained by the availability of the world model to the algorithm. Loch and Singh (1998) showed very good results (see Table 1) with a simple model-less and memory-less SARSA(λ) approach, where Q-values are stored not for states but directly for observations.

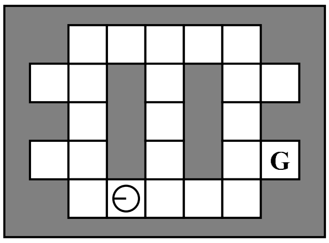

Figure 5. The 89-state maze. In this maze, the agent has a position, an orientation, and can execute five different actions: Forward, TurnLeft, TurnRight, TurnAbout and doNothing. The agent starts every trial in a random position. Its goal is to move to the square labeled 'G'. The actions and observations are extremely noisy, for example, there is only about 70% chance that the agent will get an observation right. For the precise world model we refer to Littman, Cassandra and Kaelbling (1995).

We took Loch and Singh's parameters (exploration method, $\lambda = 0.9$, $\alpha = 0.01$, $\gamma = 0.9$) and generalized the algorithm to our UDHMM version with BARBA(λ) by replacing direct observations with UDHMM's hidden states. We set $\theta = 0.2$. Every trial was allowed up to 251 steps. We ran 21 runs. The results for the median run (median in the sense of median percentage success in all test trials) are shown in Figure 6 and 7.

Of course, this algorithm performs worse than recent algorithms that do have a world model. A good comparison must therefore be sought among model-free algorithms. Within that domain, the results are similar to RL-LSTM (Bakker, 2004), a recurrent neural network approach that uses gating to handle long-term

Table 1. Overview of the performance of several algorithms on the 89-state maze. Shown is the percentage that reached the goal within 251 steps, and the median number of steps to task completion. 'Human' refers to the experiments Littman (Littman, Cassandra and Kaelbling, 1995) did himself.

ALGORITHM	GOAL%	MEDIAN STEPS
RANDOM WALK	26	>251
HUMAN	100	29
LINEAR Q (SEEDED)	84	33
SARSA(λ)	77	73
RL-LSTM	94	61
UDHMM	92	62

time dependencies. RL-LSTM yields very good results, although it is slow to converge. Our algorithm converges much faster, because it both learns and uses a behavior model.

UDHMM's model is inherently probabilistic in nature and handles noise well. That partly explains its good performance on the 89-state maze. McCallum's U-tree should also be able to learn this task. But because of its discrete decision tree model structure, it often needs a new branch for a new noisy event, leading to a very large tree model. This makes UDHMM more appropriate for the task. Our algorithm is among the best model-free algorithms available.

As a final note, we want to report our findings on experimental results *without* the state-splitting heuristic. In these cases, instead of using states that model Gaussian return distributions, we used states that use *mixtures* of Gaussians to approximate their return distributions arbitrarily closely. This should be an improvement, since the assumption that return distributions are always Gaussian is not realistic. Because of the mixtures, we could not use the state-splitting test anymore, though. This we compensated by starting out with large models with many states. The results, surprisingly, were almost identical to the results achieved with state-splitting and single Gaussians. However, many more Baum steps were required to arrive at those same results.

5. DISCUSSION

The computational complexity of UDHMM is quadratic in the number of states, per Baum step $\mathcal{O}(N^2T)$. This is not very bad but considering that many hundreds of Baum steps are needed even to solve small POMDPs, it is a rather slow algorithm. In this sense it compares unfavorably with, for example, U-tree, which is instance-based.

UDHMM's performance measured in the number of steps taken in the world in order to approach a good policy is very good for certain classes of problems, though it varies considerably due to possible suboptimal local maxima. Utility modeling is done elegantly using Baum-Welch, and a state-splitting heuristic ensures the availability of enough memory nodes. Moreover, UDHMM, unlike U-tree, allows for a continuous multi-dimensional observation space in a very natural way and develops a probabilistic behavior model during its task performance. Also, UDHMM includes a method for *selective attention* in multi-dimensional observation spaces: parts of the observation vector that are helpful to utility prediction tend to be modeled more precisely than parts of the vector that are not. Now UDHMM heavily biases the modeling of observations together with the modeling of utility, thereby leading to a kind of selective perception. In our findings, the behavior models found by UDHMM were very close or equivalent to the underlying MDP, and certainly much closer than model-learning with $\theta = 0$. However, the behavior models we found were sometimes larger than the underlying MDP. This suggests we might benefit from a merge operator in order to keep the model small.

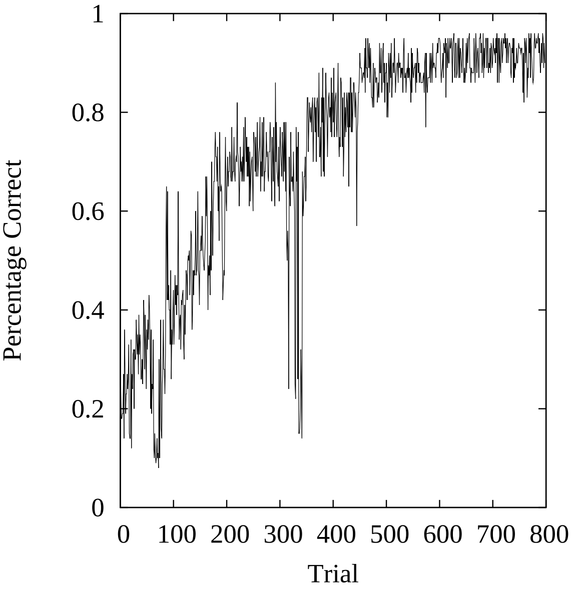

Figure 6. Results for the 89-state maze. Here we plot the percentage of tests that reach the goal within 251 steps.

In this paper, we used a highly approximate POMDP-control learning algorithm. However, recent advances in solving POMDPs with models suggests the possibility of using other model-solving methods. The UDHMM can also be used in combination with other reinforcement learning methods such as other Q-learning variants or policy-gradient methods (Aberdeen, 2003). This makes it a flexible tool for multiple POMDP approaches (including exact algorithms). We think more research on the interaction between model-learning and model-solving stages could be useful.

Much research has been directed at many different extensions to the basic HMM framework. Future work on UDHMM could include factorizing the internal state space by using Factorial or Coupled Hidden Markov Models. We could think of hierarchical approaches, using Hierarchical Hidden Markov Models, that could increase detection of relevant long-term dependencies and could facilitate the re-use of (parts of) already

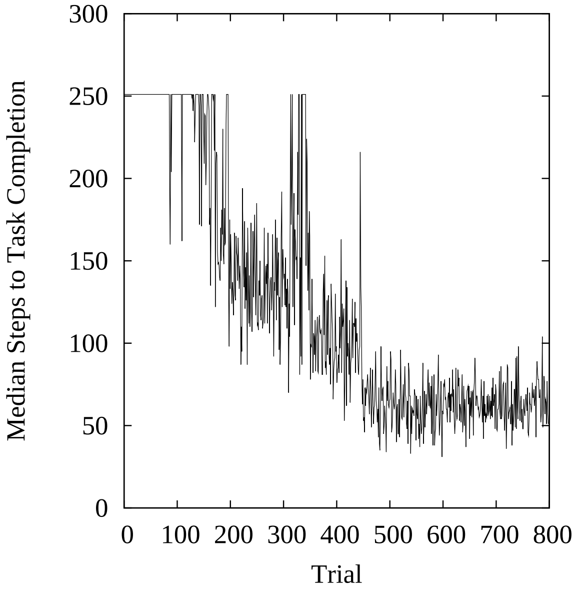

Figure 7. Results for the 89-state maze. Here we plot the median number of steps taken to task completion, i.e. reaching the goal. Note that the first trials have all 251 median steps associated with them, since if a trial takes longer than 251 steps, we break off and start a new trial.

learnt policies. Several other EM methods seem applicable. Another possible improvement could be the modeling of *difference* in utility instead of utility per se. Considering all this, we conclude this is a very promising direction of research.

Acknowledgements

We would like to thank the anonymous reviewers for their helpful comments and suggestions.

References

Aberdeen, D. (2003). *Policy-Gradient Algorithms for Partially Observable Markov Decision Processes.* Doctoral dissertation, Research School of Information Science and Engineering, Australian National University.

Bakker, B. (2004). *The State of Mind.* Doctoral dissertation, Unit of Cognitive Psychology, Leiden University.

Bengio, Y., & Frasconi, P. (1995). An input/output HMM architecture. In G. Tesauro & D. Touretzky & T. Leen (Ed.), *Advances in Neural Information Processing Systems 7* (pp. 427–434). Cambridge, MA: MIT Press.

Chrisman, L. (1992). Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. *Proceedings of the Tenth International Conference on Artificial Intelligence* (pp. 183–188). San Jose, California: AAAI Press.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: a survey. *Journal of Artificial Intelligence Research, 4*, 237–285.

Lin, L., & Mitchell, T. (1993). Reinforcement learning with hidden states. *From animals to animats 2: Proceedings of the second international conference on simulation of adaptive behavior* (pp. 271–280). Cambridge, MA: MIT Press.

Littman, M. L., Cassandra, A. R., & Kaelbling, L. P. (1995). Learning policies for partially observable environments: Scaling up. *Proceedings of the Twelfth International Conference on Machine Learning* (pp. 362–370). San Francisco: Morgan Kaufmann.

Loch, J., & Singh, S. (1998). Using eligibility traces to find the best memoryless policy in partially observable Markov decision processes. *The Proceedings of the Fifteenth International Machine Learning Conference* (pp. 141–150). San Francisco: Morgan Kaufmann.

McCallum, R. A. (1993). Overcoming incomplete perception with utile distinction memory. *The Proceedings of the Tenth International Conference on Machine Learning* (pp. 190–196). San Francisco: Morgan Kaufmann.

McCallum, R. A. (1995a). Instance-Based Utile Distinctions for Reinforcement Learning with Hidden State. *The Proceedings of the Twelfth International Conference on Machine Learning* (pp. 387–395).

McCallum, R. A. (1995b). *Reinforcement Learning with Selective Attention and Hidden State.* Doctoral dissertation, Department of Computer Science, University of Rochester.

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE, 77(2)* (pp. 257–286).

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction.* Cambridge, MA: MIT Press.

Whitehead, S. D., & Ballard, D. H. (1991). Learning to perceive and act by trial and error. *Machine Learning, 7(1)*, 45–83.