Partial Observability Under Noisy Sensors — From Model-Free to Model-Based

Ronen I. Brafman BRAFMAN@CS.BGU.AC.IL
Guy Shani SHANIGU@CS.BGU.AC.IL
Solomon E. Shimony SHIMONY@CS.BGU.AC.IL
Department of Computer Science, Ben-Gurion University, Beer-Sheva 84105, Israel

Abstract

Agents learning to act in a partially observable domain may need to overcome the problem of noisy output from the agent's sensors. Research in the area has focused on model-free methods — methods that learn a policy without learning a model of the world. When the agent's sensors provide deterministic output, model-free methods produce close to optimal results. However, when the noise in the sensors increases, these methods provide less accurate policies (Shani & Brafman, 2004). Another, less explored, option is the model-based approach — learning a POMDP model of the world, and obtaining an optimal policy from the learned model. In this paper we explore the advantages of model-based techniques over model-free methods, focusing on the ability to handle noisy sensors. We show how two important model-free algorithms: internal memory (Peshkin et al., 1999), and Utile Suffix Memory (McCallum,), can be used to learn a model of the environment.

1. Introduction

Consider an agent situated in a partially observable domain: It executes an action that may change the state of the world; this change is reflected, in turn, by the agent's sensors; the action may have some associated cost, and the new state may have some associated reward or penalty. Thus, the agent's interaction with this environment is characterized by a sequence of action-observation-reward steps, known as $instances$. In this paper we focus our attention on agents with imperfect and noisy sensors that learn to act in such environments without any prior information about the underlying set of world-states and the world's dynamics, except for information about their sensors' capabilities (namely, a predefined sensor model). This is a known variant of reinforcement learning (RL) in partially observable domains (Cassandra et al., 1994).

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

Learning in partially observable domains can take one of two forms; the agent can learn a policy directly (Peshkin et al., 1999; McCallum,), or it can learn a model of the environment, usually represented as a Partially Observable Markov Decision Process (POMDP) [1] , and solve it (Chrisman, 1992; Nikovski,). This approach has not been favored by researchers, as learning a model appears to be a difficult task, and computing an optimal solution may prove impractical for large models. However, in the past few years, much progress in the area of approximate (Poupart & Boutilier, 2004; Spaan & Vlassis,) solutions for POMDP models has been made. In view of these advances, we reconsider the model-based methods, focusing on their ability to handle sensor noise.

Identifying the "real" states of the world using noisy sensors is a difficult task. Even when the agent's sensors provide it with deterministic output, the agent may still suffer from the problem of *perceptual aliasing* (Chrisman, 1992), when different actions should be executed in two states where sensors provide the same output. For example, in Figure 1(a) the left and right corridors are perceptually aliased if sensors can only sense adjacent walls. Model-free methods - such as augmenting the observations with internal memory (Peshkin et al., 1999), or using variant-length finite history windows (McCallum,) - can be used to disambiguate the perceptually aliased states. When the agent's sensors provide deterministic output, learning to properly identify the underlying world states reduces the problem to a fully observable MDP, making it possible for methods such as Q-learning to compute an optimal policy.

When sensors provide slightly noisy output, model-free methods still produce close to optimal results, but as noise

[1]See Section 2.1 for an overview of MDPs and POMDPs

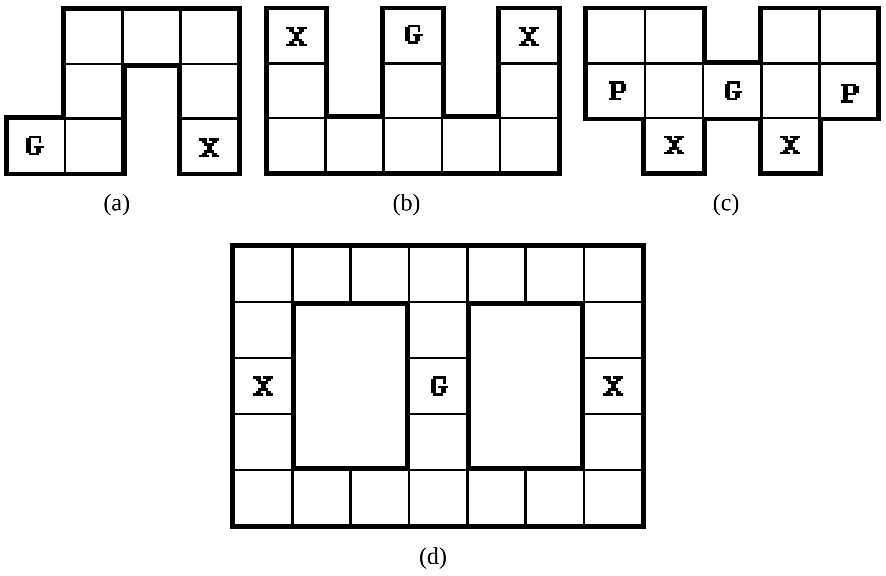


Figure 1. Four maze domains. The agent receives a reward of 9 upon reaching the goal state (marked with 'G'). Immediately afterwards (in the same transition) the agent is transferred to one of the states marked 'X'. Arrival at a state marked 'P' results in a penalty (negative reward) of 9.

in the sensors increases, their performance rapidly decreases (Shani & Brafman, 2004). This is because disambiguating the perceptually aliased states under noisy sensors does not result in an MDP, but rather in a POMDP. POMDP models are harder to solve, but their solution handles noisy observations optimally.

Using model-free methods to create a POMDP has been previously suggested by Nikovski (Nikovski,) in his PhD dissertation. Nikovski compared his method, based on McCallum's earlier NSM algorithm (McCallum,) to POMDP learning methods based on the Baum-Welch algorithm and to other state merging techniques. Nikovski did not show his models to be superior to any model-free techniques and did not experiment with sensor noise.

In this paper, we show how both McCallum's Utile Suffix Memory (USM) algorithm, that learns a variant-length finite history window, and the internal memory approach suggested by Peshkin *et al.* can be used to initialize a POMDP model of the world. We continue to solve the resulting models and compare the average reward collected by model-free and model-based approaches. We then show that the resulting models provide superior results to the policy learned by the model-free methods. This indicates that the use of an explicit model is advantageous for diverse methods of initializing the model.

We note that using an explicit model of the environment can have many other advantages such as taking the "value of information" into consideration, and helping to find the areas of the world that require learning, and may hold potential rewards, but we leave those topics to future research.

One contribution of this paper is in suggesting that solution of a POMDP generated from memory models can be used to improve agent performance. Another contribution is the development of the specific techniques that apply this idea to the memory bits and the USM schemes. Empirical results for the above memory schemes show the advantages using the derived POMDP in these cases.

This paper is structured as follows: we begin (Section 2) with an overview over MDPs, POMDPs, the memory bits and USM schemes. We then explain how the learned policy of model-free methods can be used to construct a POMDP in Section 3. We provide an experimental evaluation of our work in Section 4, followed by a short discussion in Section 5 and conclude in Section 6.

2. Background

2.1. MDPs and POMDPs

A Markov Decision Process (MDP) (Howard, 1960) is a model for sequential stochastic decision problems. An MDP is a four-tuple: $\langle S, A, R, tr \rangle$, where S is the set of the states of the world, A is a set of actions an agent can use, R is a reward function, and tr is the stochastic state-transition function. A solution to a MDP is a policy $\pi : S \rightarrow A$ that defines which action should be executed in each state.

Various exact and approximate algorithms exist for computing an optimal policy, and the best known are policy-iteration (Howard, 1960) and value-iteration (Bellman, 1962). Solving MDPs is known to be a polynomial problem in the number of states, and therefore exponential in the number of state variables.

A well known extension to the MDP model is the Partially Observable Markov Decision Process (POMDP) model (Cassandra et al., 1994). A POMDP is a six-tuple $\langle S, A, R, tr, \Omega, O \rangle$, where S, A, R, tr define an MDP, Ω is a set of possible observations and $O(a, s, o)$ is the probability of executing action a, reaching state s and observing o. In a POMDP the agent is unable to identify the current state and is therefore forced to estimate the current state given the set of current observations (e.g. output of the robot sensors). In most applications a POMDP is more natural and complete formalization than an MDP, but using POMDPs increase the difficulty of computing an optimal solution.

Solving a POMDP is an extremely difficult computational problem, and various attempts have been made to compute approximate solution problems that work reasonably well in practice. Among these methods is the use of state-space reductions (Poupart & Boutilier, 2004), point based value iteration (Spaan & Vlassis,) and many more.

The idea of learning a POMDP model of the environment was examined by early researchers (Chrisman, 1992; McCallum,) who used a variant of the Baum-Welch algorithm for learning hidden Markov models, refining the state space when it was observed to be inadequate. These methods were slow to converge and could not outperform the rapid convergence and reasonable results generated by model-free methods. Nikovski (Nikovski,) used McCallum's earlier model-free method, Nearest Sequence Memory (NSM) (McCallum,), to identify the states of the world and learn the transition, reward, and observation functions. He showed that the learned models produced superior results to the models obtained by using the Baum-Welch algorithm. His models, however, were tested on domains with little noise, and are much less adequate when sensors are noisy. This is to be expected, as NSM handles noisy observations poorly, while USM can still produce reasonable results, though in no way optimal.

Nikovski also did not attempt to use any modern technique for solving his models and obtaining a policy. Instead, he experimented with approximate methods based on the solution to the underlying MDP model.

2.2. Memory bits

Early research in model-free techniques has shown that MDP based techniques such as Q-learning, SARSA and eligibility traces (Sutton & Barto, 1998) fail to converge in the presence of perceptual aliasing.

Peshkin *et al.* has suggested to augment the agent state space with bits of internal memory (though he referred to it as external), and actions that change the value of a single memory bit. The agent can therefore choose to either execute an action that influences the environment, or flip one of its internal memory bits. Q values are learned for all such actions using any RL technique, such as SARSA(λ) — SARSA with eligibility traces. Agents with internal memory can learn to remember events that happened arbitrarily far in the past in order to disambiguate the perceptual aliasing.

2.3. Utile Suffix Memory

Instance-based state identification (McCallum,) resolves perceptual aliasing with variable length short term memory. An instance is a tuple $T_t = \langle T_{t-1}, a_{t-1}, o_t, r_t \rangle$ — the individual observed raw experience. Algorithms of this family keep all the observed raw data (sequences of instances), and use it to identify matching subsequences. The algorithm assumes that if the suffix of two sequences is similar both were likely generated in the same world state.

Utile Suffix Memory creates a tree structure, based on the well known suffix trees for string operations. This tree maintains the raw experiences and identifies matching suffixes. The root of the tree is an unlabeled node, holding all available instances. Each immediate child of the root is labeled with one of the observations encountered during the test. A node holds all the instances $T_t = \langle T_{t-1}, a_{t-1}, o_t, r_t \rangle$ whose final observation o_t matches the observation in the node's label. At the next level, instances are split based on the last action of the instance a_t. Then, we split again based on (the next to last) observation o_{t-1}, etc. All nodes act as buckets, grouping together instances that have matching history suffixes of a certain length. Leaves take the role of states, holding Q-values and updating them. The deeper a leaf is in the tree, the more history the instances in this leaf share.

The tree is built on-line during the test run. To add a new instance to the tree, we examine its precept, and follow the path to the child node labeled by that precept. We then look at the action before this precept and move to the node labeled by that action, then branch on the precept prior to that action and so forth, until a leaf is reached.

Identifying the proper depth for a certain leaf is a major issue, and we shall present a number of adjustments to McCallum's methods (some already suggested in his PhD thesis). Leaves should be split if their descendants show a statistical difference in expected future discounted reward associated with the same action. We split a node if knowing where the agent came from helps predict future discounted rewards. Thus, the tree must keep what McCallum calls fringes, i.e., subtrees below the "official" leaves.

For better performance, McCallum did not compare the

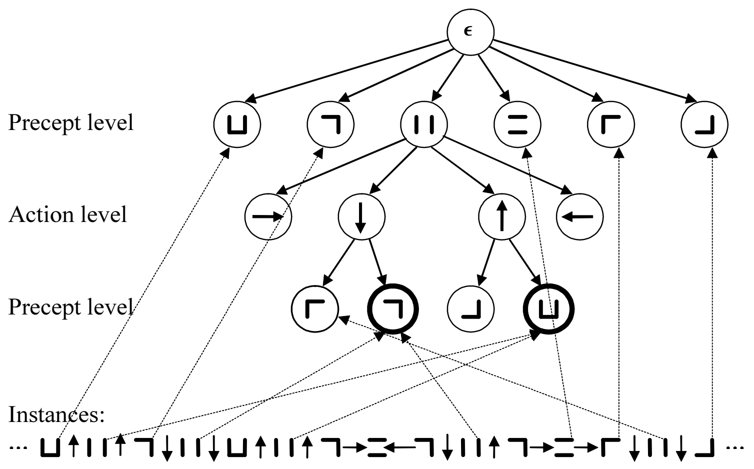


Figure 2. A possible USM suffix tree generated by the maze in Figure 1(a). Below is a sequence of instances demonstrating how some instances are clustered into the tree leaves. The two bolded leaves correspond to the same state — the right perceptually aliased corridor. During most executions under deterministic sensor output the above tree structure was generated.

nodes in the fringes to their siblings, only to their ancestor ”official” leaf. He also did not compare values from all actions executed from the fringe, only the action that has the highest Q-value in the leaf (the policy action of that leaf). To compare the populations of expected discounted future rewards from the two nodes (the fringe and the ”official” leaf), he used the Kolmogorov-Smirnov (KS) test — a non-parametric statistical test used to find whether two populations were generated by the same distribution. If the test reported that a difference was found between the expected discounted future rewards after executing the policy action, the leaf was split, the fringe node would become the new leaf, and the tree would be expanded to create deeper fringes. Figure 2.3 presents an example of a possible USM tree, without fringe nodes.

Instead of comparing the fringe node to its ancestor ”official” leaf, we found it computationally possible to compare the siblings of the fringe, avoiding the problem that the same instance appears in both distributions. McCallum compared only the expected discounted future rewards from executing the policy action, where we compare all the values following all actions executed after any of the instances in the fringe. McCallum used the KS test, where we choose to use the more robust randomization test (Yeh, 2000) that works well with small sets of instances. McCallum also considered only fringe nodes of certain depth, given as a parameter to the algorithm, where we choose to create fringe nodes as deep as possible, until the number of instances in the node diminish below some threshold (we use a value of 9 in our experiments).

We define the expected discounted reward of instance T_i:

$$Q(T_i) = r_i + \gamma U(L(T_{i+1})) \tag{1}$$

where $L(T_i)$ is the leaf associated with instance T_i and $U(s) = max_a(Q(s,a))$.

After inserting new instances into the tree, we update Q-values in the leaves using:

$$R(s,a) = \frac{\sum_{T_i \in T(s,a)} r_i}{|T(s,a)|} \tag{2}$$

$$Pr(s'|s,a) = \frac{|\forall T_i \in T(s,a), L(T_{i+1}) = s'|}{|T(s,a)|} \tag{3}$$

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} Pr(s'|s,a)U(s') \tag{4}$$

We use s and s' to denote the leaves of the tree, as in an optimal tree configuration for a problem the leaves of the tree define the sates of the underlying MDP. The above equations therefore correspond to a single step of the value iteration algorithm used in MDPs.

Now that the Q-values have been updated, the agent chooses the next action to perform based on the Q-values in the leaf corresponding to the current instance T_t:

$$a_{t+1} = argmax_a Q(L(T_t), a) \tag{5}$$

McCallum uses the fringes of the tree for a smart exploration strategy. In our implementation we use a simple ϵ-greedy technique for exploration.

3. Constructing Models from Model-Free Methods

Any model-free technique designed to learn deterministic policies in a partially observable domain with perceptual aliasing, must employ some type of internal memory. The structure of the internal memory can be used to initialize a POMDP model. The method for converting the internal memory into depends on the way the model-free method constructs its memory state and transitions. This section explains how the computed policy of the memory bits and USM approaches can be used to initialize a POMDP model.

Model-free methods have been extensively studied, and there are many other approaches to resolving perceptual aliasing we have not reviewed, including the use of finite-state automata (FSA) (Meuleau et al., 1999), which can be viewed as a special case of the memory-bits approach, but can learn faster and more accurately and the use of neural networks for internal memory (Lin & Mitchell, 1992; Hochreiter & Schmidhuber, 1997). It is likely that those approaches can also be used to initialize a POMDP model, similarly to USM and memory bits. We note that most researchers test their algorithms on environments with very little noise, and do not analyze the effect of noisy sensors.

3.1. Creating a POMDP from Utile Suffix Memory

After the USM algorithm has generated a tree structure, one can use this tree structure to create a POMDP. The state space is defined as the set of leaves computed by USM. We note that this state representation is not necessarily compact, as it is quite possible that if a perceptually aliased state can be reached from two different locations, it may have two different leaves that represent it. For example, consider the two leaves in thick line-style in Figure 2.3. We tried several techniques to merge leaves that correspond to the same underlying world state, such as merging leaves that have similar expected rewards, and merging leaves that have similar transition probabilities, but both methods either did not merge any leaves, or resulted in wrong leaves getting merged and thus in suboptimal performance.

Obtaining the POMDP parameters from the USM tree structure is straightforward. The actions (A) and observations (Ω) are known to the agent prior to learning the model. As stated above, the leaves of the tree (after convergence of the USM algorithm) are used as the states of S. The transition function ($tr(s,a,s')$) is defined by Equation 3 and the reward function ($R(s,a)$) by Equation 2.

Learning the observation function is harder, as in USM a state always corresponds to a single observation, and all instances mapped to the state will always observe the same sensor output. It is therefore unclear how to learn $pr(o|a,s)$ — the probability of observing o after reaching state s with action a. We are able to learn a different probability function — the probability of observing o after executing action a from state s, but most of the domains modeled by POMDPs depend on the target state, not on the source state, making the latter observation function insufficient.

We therefore adopt the approach taken by Shani et al. (Shani & Brafman, 2004), where an observation model is assumed, and define the observation function based on the observation model. It is reasonable to assume that the agent has some sensor model defining $pr(o|s)$ — the probability that the agent will observe o in world state s. For example, such information can easily be obtained in real world robot applications by placing a robot in front of a wall and measuring how likely it is to sense the wall. Thus, it is possible to define the correct observation function given the state features (which are uniquely determined by the state: the walls are the features in our experiment), but does not entail actual knowledge of the entire state space.

3.2. Creating a POMDP from Memory Bits

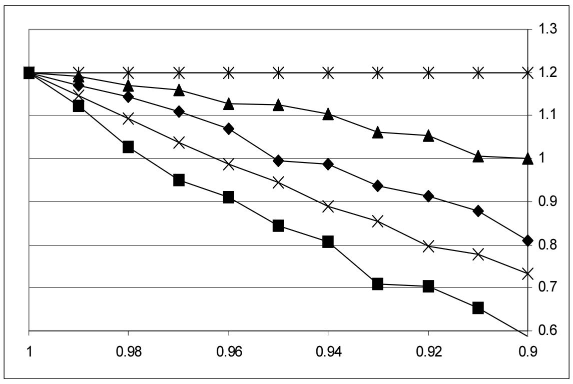
Once the memory bits algorithm has run for a while, one can use the learned Q-table and the observed instances to initialize the POMDP state space. Let us define $s = \langle o, m \rangle$ the agent state composed of the sensor observation o and the agent internal memory state m. An observation o originates in a perceptually aliased state if there exists two agent states $s =< o, m >$, $s' =< o, m' >$, such that $m \neq m'$ and $max_a Q(s,a) \neq max_a Q(s',a)$, and none of these actions is an action that flips a memory bit. In other words, state are perceptually aliased if the agent learned that it needs to act differently observing the same sensor output, but different internal memory states.

We can now merge every $s =< o, m >$ and $s' =< o, m' >$ that are not perceptually aliased. Using the observed instances, the transition $tr(s,a,s')$ and reward $R(s,a)$ functions can be computed, much the same way as we did for the USM based model. Again, we assume that the observation function is pre-defined.

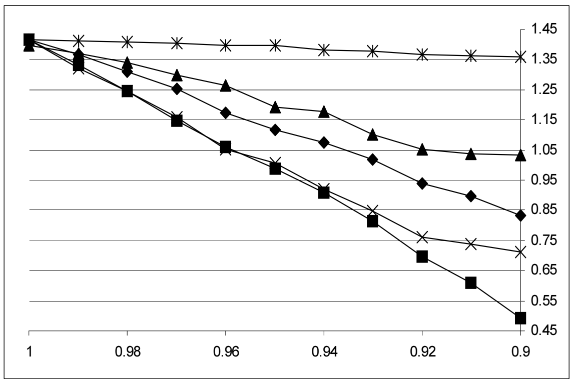
The instances we use to learn the model must be generated by the memory bits algorithm when it is close to convergence, or after it has converged, in order to properly learn the probabilities. When collecting these instances, the agent must still explore, but it must not use exploration in states where the optimal action modifies the memory bit, since this will cause it to observe transitions that are impossible when following the learned policy. We note that the learned model is therefore imperfect and does not have all transition and reward data, yet it still outperforms the original memory bits algorithm.

4. Experimental Results

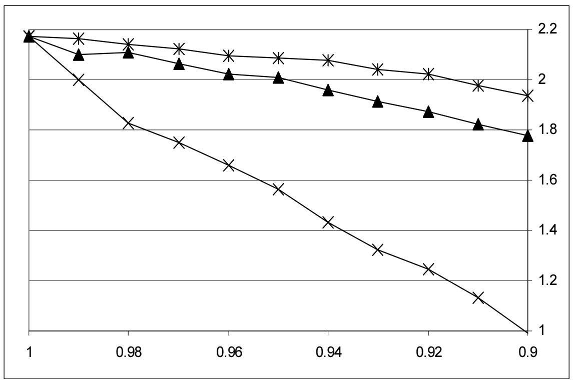
In our experiments we ran both USM and SARSA(λ) with one additional memory bit on the mazes in Figure 1. Once the average reward collected by the algorithms passed a certain threshold, their current state (USM's tree structure, and SARSA's Q-table) was kept, exploration was stopped



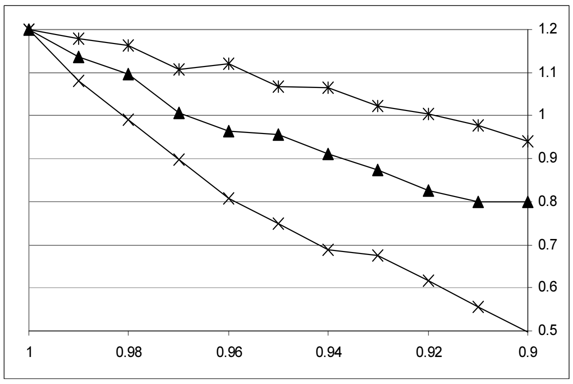
(a) Results for the maze in Figure 1(a)



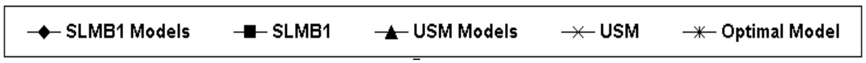
(b) Results for the maze in Figure 1(b)



(c) Results for the maze in Figure 1(c)



(d) Results for the maze in Figure 1(d)



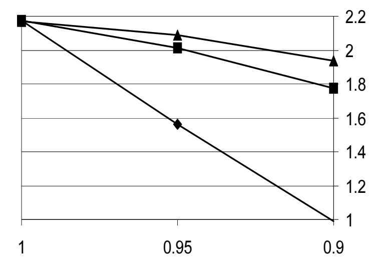
(e) Legend for the above graphs

Figure 3. Results for the mazes in Figure 1. In all the above graphs, the X axis contains the diminishing sensor accuracy, and the Y axis marks average reward per agent action. The above results are averaged over 10 different executions for each observation accuracy and method. All variances were below 0.015 and in most cases below 0.005.

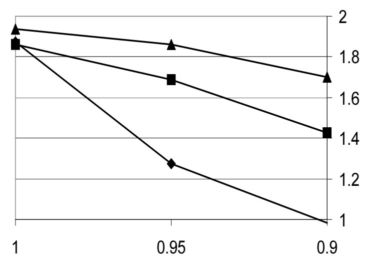
(as the POMDP policy does not explore). Then, the runs were continued for 5000 iterations to calculate the average reward gained by the converged algorithm. The agent current state was then used to learn a POMDP model as explained above. The model was then solved by the Perseus algorithm (Spaan & Vlassis,), and the resulting model was executed for another 5000 iterations. To show the optimal possible policy, we manually defined a POMDP model for each of the mazes above, solved it using Perseus and ran the resulting policy for 5000 iterations.

The agent in our experiments has four sensors allowing it to sense an immediate wall above, below, to the left, and to the right of its current location. Sensors have a boolean output with probability α of being accurate. The probability of all sensors providing the correct output is therefore α^4. We assume that the agent knows in advance the probability of sensing a wall if a wall exists, and compute the observation function from this information. In the maze there is a single location that grants the agent a reward of 9, and in the maze in Figure 1(c) there are two locations where the agent receives a negative reward (punishment) of 9. Upon receiving a reward or punishment, the agent is transformed to any of the states marked by X. If the agent bumps into a wall it pays a cost (a negative reward) of 1. For every move the agent pays a cost of 0.1.

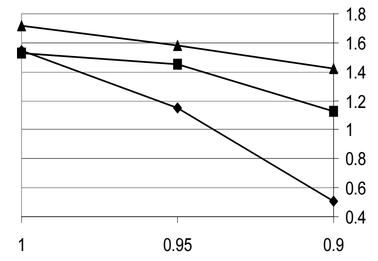
Figure 4 shows the average collected reward for each method when α (the sensor accuracy) varies from 1.0 (deterministic sensor output) to 0.9 (probability 0.65 for detecting all features correctly), averaged over 9 executions for every method. SLMB1 stands for adding a single memory bit to the Sarsa(λ) algorithm. SLMB1 Model and USM Model are the learned models after executing the memory bits and the USM algorithms, respectively, creating a POMDP from the algorithm output, solving it and executing the resulting policy. Optimal Model is the manually defined POMDP model. In the two latter mazes, the memory



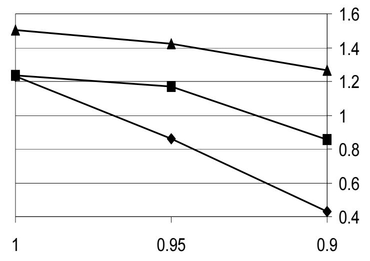
(a) 0 action noise (deterministic actions)



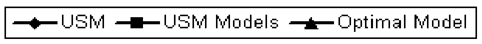
(b) 0.1 action noise



(c) 0.2 action noise



(d) 0.3 action noise



(e) Legend for the above graphs

Figure 4. Results for the maze in Figure 1(c) using various levels of action noise (i.e., (1 – action-success-probability)). In all the above graphs, the X axis is the diminishing sensor accuracy, and the Y axis is the average reward per agent action. The above results are averaged over 10 different executions for each observation accuracy and method. All variances were below 0.015 and in most cases below 0.005.

bits algorithm needed two memory bits and failed to converge as sensor noise increased. We report results only for USM and the manually defined POMDP on these domains.

As seen from the results, the model-based methods greatly improve the original model-based techniques and the advantage becomes more important as noise in the sensors increases. The memory bits based model does not perform as well as the USM based model, probably due to the inaccurate model parameters that were learned because of the inability to explore all states and actions. The USM model in our experiments is also suboptimal, mainly because several leaves correspond to the same world state.

In most cases actions in MDPs and POMDPs do not have deterministic effects. It is quite possible that an action attempted by an agent can fail (in our experiments, a failed action leaves the agent in the same state). We therefore also experimented with various action success probabilities. Figure 4 shows the results of decreasing action success probability. While the model computed from the USM tree becomes farther from the optimal model, it still outperforms USM by approximately the same amount. The USM based models performance degrades partially due to the increased number of leaves (and hence, states) in the presence of noisy actions, as failed transitions cannot be expressed in a single leaf and instead deeper branches are created for such histories (see Table 2).

The memory bits model in our tests produced smaller state spaces then USM (see Table 1). This is because the memory bits algorithm defines a constant upper bound on the number of states defined by all the possible combinations of the internal memory states and the observations.

5. Discussion

As we have seen above, the performance of model-free techniques degrades when sensor noise increases. This is due to the application of methods designed for learning in a fully observable MDP on a POMDP. These methods assume that the agent knows at each point in time its exact location. When the agent sensors are accurate, resolving the perceptual aliasing indeed results in an MDP and the model-free methods produce reasonable results. In the presence of low sensor noise, the model-free algorithms still produce reasonable approximations, but as sensor noise increases the agent often does not properly estimates its current state resulting in both the execution of

α	Maze in Figure 1(d)	Maze in Figure 1(c)	Maze in Figure 1(a)		Maze in Figure 1(b)	
	USM	USM	USM	Memory bits	USM	Memory bits
1.00	43.44(6)	11.2(5.61)	17.2(0.42)	10(0)	18(0)	23.6(7.1)
0.99	59.77(17.07)	28.7(5.61)	19.6(1.34)	27.7(1.82)	20.6(1.64)	44.7(23.77)
0.98	70.22(17.82)	44.3(10.8)	19.9(0.99)	29.3(2.49)	20.1(1.28)	57.8(19.99)
0.97	76.66(28.2)	51.6(15.84)	19(1.15)	29.8(2.44)	19.1(1.37)	53.6(23.13)
0.96	80.11(17.67)	61.4(19.76)	19.4(1.57)	34.9(6.15)	19.5(1.08)	48.9(24.37)
0.95	114.66(50.36)	70.1(11.8)	19.1(0.99)	36.9(4.14)	19.2(0.91)	47.5(19.79)
0.94	153.44(38.73)	79.6(12.75)	18.7(1.25)	40.5(11.07)	19.3(0.82)	46.2(16.2)
0.93	178.88(28.32)	87.6(18.3)	18.9(0.73)	47.4(18.04)	18.3(0.94)	39.8(3.62)
0.92	232.9(43.64)	113(25.16)	18.8(0.78)	49.7(16.05)	18.7(0.82)	43.5(4.45)
0.91	271.4(27.74)	103.6(27.33)	18.9(1.37)	52.1(19.43)	19.4(0.74)	54.3(12.06)
0.90	307(32)	107.4(20.98)	18.9(1.1)	82.5(15.36)	18.9(0.95)	80.2(16.26)

Table 1. Resulting model sizes for both the USM and Memory bits algorithms. Each result is averaged over 10 different executions. Standard deviation is reported in brackets.

α	Action noise 1.0	Action success 0.9	Action success 0.8	Action success 0.7
1.00	11.2(5.61)	25.5(12.41)	30.5(10.77)	27.9(11.43)
0.95	70.1(11.8)	89.5(24.98)	99.1(13.08)	96.3(16.87)
0.90	107.4(20.98)	114.42(20.15)	147.88(25.69)	131.3(21.81)

Table 2. Resulting model sizes of USM for various levels of noise on the maze in figure 1(c). Each result is averaged over 10 different executions. Standard deviation is reported in brackets.

wrong actions, and with an unreliable value function.

POMDPs can be accurately solved using the belief state MDP, but the current model-free methods have no concept of a belief state or the belief state MDP. Defining a POMDP, solving it, and maintaining a belief state will therefore show superior results to using any model-free method.

6. Conclusions and Future Work

In this paper we explored the advantages of model-based methods over model-free methods for acting in partially observable domains with noisy sensors. As learning the model is difficult, we have shown how an agent can execute a model-free method until it converges and then use the learned data to initialize a POMDP model that outperforms the original model-free method.

Future research should focus on other advantages of model-based techniques, such as their ability to consider the ”value of information”. Splitting the execution into a learning and exploiting stage is also undesirable. It would be better to show how the model-free and model-based can be combined to produce an incrementally built model that is generated based on the model-free method. This approach seems especially attractive when using the USM algorithm.

References

Bellman, R. E. (1962). *Dynamic programming*. Princeton University Press.

Cassandra, A. R., Kaelbling, L. P., & Littman, M. L. (1994). Acting optimally in partially observable stochastic domains. *AAAI'94* (pp. 1023–1028).

Chrisman, L. (1992). Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. *AAAI'02* (pp. 183–188).

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*, 1735–1780.

Howard, R. A. (1960). *Dynamic programming and markov processes*. MIT Press.

Lin, L.-J., & Mitchell, T. M. (1992). *Memory approaches to reinforcement learning in non-markovian domains* (Technical Report CMU-CS-92-138).

McCallum, A. K. *Reinforcement learning with selective perception and hidden state*. Doctoral dissertation.

Meuleau, N., Peshkin, L., Kim, K., & Kaelbling, L. P. (1999). Learning finite-state controllers for partially observable environments. *UAI'99* (pp. 427–436).

Nikovski, D. *State-aggregation algorithms for learning probabilistic models for robot control*. Doctoral dissertation.

Peshkin, L., Meuleau, N., & Kaelbling, L. P. (1999). Learning policies with external memory. *ICML'99* (pp. 307–314).

Poupart, P., & Boutilier, C. (2004). Vdcbpi: an approximate scalable algorithm for large pomdps. *NIPS 17*. MIT Press.

Shani, G., & Brafman, R. (2004). Resolving perceptual aliasing in the presence of noisy sensors. *NIPS 17*.

Spaan, M. T. J., & Vlassis, N. *Perseus: randomized point-based value iteration for pomdps* (Technical Report IAS-UVA-04-02).

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. MIT Press.

Yeh, A. (2000). More accurate tests for the statistical significance of result differences. *COLING 18* (pp. 947–953).