Probabilistic robot planning under model uncertainty: an active learning approach

Robin JAULMES, Joelle PINEAU and Doina PRECUP
School of Computer Science
McGill University
Montreal, QC CANADA H3A 2A7

Editor: Greg Grudic and Jane Mulligan

Abstract

While recent POMDP techniques have been successfully applied to the problem of robot control under uncertainty, they typically assume a known (and stationary) model of the environment. In this paper, we study the problem of finding an optimal policy for controlling a robot in a partially observable domain, where the model is not perfectly known, and may change over time. We present an algorithm called MEDUSA which incrementally learns a POMDP model using oracle queries, while still optimizing a reward function. We demonstrate effectiveness of the approach for realistic robot planning scenarios, with minimal *a priori* knowledge of the model.

1. Introduction

Partially Observable Markov Decision Processes (POMDPs) provide a sound decision-theoretic framework for planning under uncertainty. Due to recent advances in algorithmic techniques, POMDPs have recently been applied in large domains where the state of the robot and other agents in the environment is only partially observable (12; 13; 14; 18). Techniques currently applicable in large-scale problems however consistently require a known and stable model of the robot's sensors, as well as of the dynamics of the robot itself, the other agents and the environment. This is a severe limitation for many practical applications, where writing down an exact model can be time-consuming and error-prone. Experience-based approaches have been proposed, which learn a model from direct experience with the environment (3; 10; 15; 17; 2; 16). However these typically require very large amounts of data, and assume a stationary model. Both of these assumptions may be hard to guarantee in many realistic robotic task scenarios.

In reality, there are many applications where it is relatively easy to provide a rough model, but much harder to provide an exact one. We would like to have planning techniques which offer a better compromise between model-based or experience-based, to effectively combine whatever information is available. Furthermore, because models may change over time (e.g. non-stationary environments), we would like to use experimentation to automatically track these changes, and adapt our model accordingly.

The goal of this work is to combine a partial model of the environment with direct experimentation, in order to produce solutions that are robust to model uncertainty and evolution, while scaling to large robotics domains. To do this, we assume that uncertainty is part of the model and design our agent to take it into account when making decisions.

The technique we propose in this paper is an algorithm called MEDUSA. It is based on *active learning* (4), which is a well-known problem formulation in machine learning for classification tasks with sparsely labelled data. In traditional active learning, the goal is to select which examples should be labelled by considering the expected information gain. More recently, others have shown that these ideas extend nicely to dynamical systems such as HMMs and MDPs (1);(5).

In MEDUSA, we assume that prior knowledge about the model, as well as the level of uncertainty in the model, can be represented by a Dirichlet distribution over possible models. The parameters of the Dirichlet are then updated whenever new experience is acquired, thus allowing a simple framework to combine *a priori* knowledge of the model, with direct experience with the environment.

MEDUSA assumes availability of an oracle, which upon request can provide the agent with exact information about the current state. For example while a robot experiments with a new environment, it can query the oracle to obtain exact state information whenever this is deemed necessary. The state information is used only to improve the model, not in the action selection process. This means that there can be delays between the query request and the query processing. It also means that the answer to the query can be noisy. Furthermore, in some cases, the (Dirichlet) parameters over model uncertainty can also be updated directly from experience, without resorting to the oracle. All of these factors make the assumption of an oracle more realistic for practical application.

The original MEDUSA algorithm was presented in (6). We discussed in (7) the conditions under which MEDUSA converges to the correct model. We now present a new and improved version of MEDUSA, featuring better techniques for combining query and non-query learning, and online adaptation for non-stationary environments. We perform the first large-scale empirical validation of MEDUSA, and present results stemming from the application of MEDUSA to a simulated robot planning scenario where an indoor mobile robot must with unknown sensor model navigate to find a human user with unknown motion behavior.

2. Partially Observable Markov Decision Processes

We assume the standard POMDP formulation (8); namely, a POMDP consists of a discrete and finite set of states S, a similar set of actions A, and a similar set of observations Z. The POMDP model is defined by state-to-state transition probabilities $\{P^a_{s,s'}\} = \{p(s_{t+1} = s' | s_t = s, s_t = a)\}, \forall s \in S, \forall a \in A, \forall s' \in S$ and observation emission probabilities $\{O^a_{s,z}\} = \{p(z_t = z | s_t = s, a_{t-1} = a)\}, \forall z \in Z, \forall s \in S, \forall a \in A$. It assumes a known discount factor $\gamma \in [0;1]$ and a deterministic reward function $R : S \times A \times S \times Z \rightarrow \Re$, such that $R(s_t, a_t, s_{t+1}, z_{t+1})$ is the reward for the corresponding transition.

At each time step, the agent is in an unknown state $s_t \in S$. It executes an action $a_t \in A$, arrives in an unknown state $s_{t+1} \in S$ and gets an observation $z_{t+1} \in Z$. Agents using POMDP planning algorithms typically keep track of the belief state $b \in \Re^{|S|}$, which is a probability distribution over all states given the history experienced so far. A policy is a function that associates an action to each possible belief state. Solving a POMDP means finding the policy that maximizes the expected discounted return $E(\sum_{t=1}^{T} \gamma^t R(s_t, a_t, s_{t+1}, z_{t+1}))$.

While finding an exact solution to a POMDP is computationally intractable, many methods exist for finding approximate solutions. In this paper, we use a point-based algorithm (12), in order to compute POMDP solutions. However, the algorithms that we propose can be used with other approximation methods.

We assume the learner knows the reward function, since it is directly linked to the task that it wants to execute. We focus instead on learning $\{P^a_{s,s'}\}$ and $\{O^a_{s,z}\}$. These probability distributions are typically harder to specify correctly by hand. They may also be changing over time. For instance in robotics, the sensor noise and motion error are often unknown and may vary with the amount of light, the wetness of the floor, or other parameters that might not be fixed in advance. To learn the transition and observation models, we assume that our agent can occasionally make a query, the answer to which will correctly identify the current state. This is a strong assumption. In many tasks it is possible (but very costly) to have access to the full state information. It can require asking a human to label the state, or using a high-precision sensor to recover the information. However the assumption of an oracle is not entirely unrealistic, since in practice we do not need to know the query result immediately, or exactly. Nonetheless the agent should strive to make as few queries as possible, and carefully select when to make them.

3. The MEDUSA algorithm

In this section we describe the MEDUSA algorithm. Its main idea is to represent the model uncertainty with a Dirichlet distribution over possible models, and to update directly the parameters of this distribution as new experience is acquired. It is also built so that it can cope with non-stationary environments, where the parameters evolve with time. Furthermore, MEDUSA makes sparse (and efficient) use of queries, by learning model uncertainty parameters from interaction with the environment, using both *query* updates and *non-query* updates.

This approach scales nicely: we need one Dirichlet parameter for each uncertain POMDP parameter, but the size of the underlying POMDP representation remains unchanged, which means that the complexity of the planning problem does not increase. However this approach requires the agent to repeatedly sample POMDPs from the Dirichlet distribution and solve the sampled models, to best select the next query.

3.1 Dirichlet Distributions

Consider a N-dimensional multinomial distribution with parameters $(\theta_1, \ldots \theta_N)$. A Dirichlet distribution is a probabilistic distribution over these parameters. The Dirichlet itself is parameterized by hyper-parameters $(\alpha_1, \ldots \alpha_N)$. The likelihood of the multinomial parameters is defined by:

$$p(\theta_1 \ldots \theta_N | D) = \frac{\prod_{i=1}^N \theta_i^{\alpha_i - 1}}{Z(D)}, \text{ where } Z(D) = \frac{\prod_{i=1}^N \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^N \alpha_i)}$$

And the maximum likelihood multinomial parameters $\theta_1^* \ldots \theta_N^*$ can be easily computed: $\theta_i^* = \frac{\alpha_i}{\sum_{k=1}^N \alpha_k}, \forall i = 1, \ldots N$. The Dirichlet distribution is convenient because its hyper-parameters can be updated directly from data. For example, if instance $X = i$ is encountered, α_i should be increased by 1. Also, we can sample from a Dirichlet distribution conveniently using Gamma distributions.

In the context of POMDPs, model parameters are typically specified according to multinomial distributions. Therefore, we use a Dirichlet distribution to represent the uncertainty over these. We use Dirichlet distributions for each state-action pair where either the transition probabilities or the observation probabilities are uncertain. Note that instead of using increments of 1 we use a learning rate, λ, which measures the importance we want to give to one query.

3.2 MEDUSA

The name MEDUSA comes from *Markovian Exploration with Decision based on the Use of Sampled models Algorithm*. The algorithm is as follows. First, we assume that initial Dirichlet parameters are given, representing both *a priori* knowledge of the model, and uncertainty over model parameters. Next, our agent samples a number of POMDP models according to this Dirichlet distribution. The agent then computes an (approximately) optimal policy for each of these models. At each time step, one of the models is chosen at random (with probability equal to the weight of each model under the current Dirichlet distribution) and the corresponding optimal action is applied. This allows us to obtain reasonable execution performance throughout the active learning process. This also allows the agent to focus the active learning in regions of the state space most often visited by good policies.

Each time an action is made and an observation is received, the agent can decide to query the oracle for the true identity of the hidden state [1] If we do a query, the Dirichlet distributions are updated according to the outcome of that query. Note that with MEDUSA, we need not specify a separate Dirichlet parameter for each unknown POMDP parameter. It is often the case that a small number of hyper-parameters suffice to characterize the model uncertainty. For example, noise in the sensors may be highly correlated over all states and therefore we could use a single set of hyper-parameters for all states. In this setup, the corresponding hyper-parameter would be updated whenever action a is taken and observation z is received, regardless of the state.[2] This results in a very expressive framework to represent model uncertainty. As we showed in our previous work, we can vary the number of hyper-parameters to trade-off the number of queries versus model accuracy and performance (6). In (7), we study the theoretical properties of MEDUSA, and show that under certain conditions (in the limit of infinite exploration, and infinite number of sampled models), the algorithm is guaranteed to converge to the correct model.

3.3 Non-Query Learning

MEDUSA is premised on the rather strong assumption that an oracle is available to reveal full state information at any time. In reality, it is not necessary to make a query at every step, rather it is possible to update the model based solely on evidence, using *non-query learning*. This requires (1) deciding when to query and (2) if we don't query (or if the query isn't answered), then using the information we have from the action-observation sequence and knowledge extracted from the previous queries to update the Dirichlet parameters.

To do an efficient non-query learning we introduce the concept of an alternate belief β. For each model $i = 1, \ldots, n$, we keep track of such an alternate belief, (denoted β_i) in addition to the standard belief (denoted b_i). The alternate belief is updated in the same way as the standard one, with each action and observation. The only difference is that when a query is done, β_i is updated to reflect the fully known state information, whereas b_i tracks belief uncertainty according to model i's parameters (ignoring the state information). The alternate belief allows us to keep track of the information available from the latest query, when applying non-query learning.

1. In practice there can be a time delay between the query and the answer, since we don't redraw models after every step.
2. As a matter of fact, we have a function that maps any possible transition or observation parameters to either a hyper parameter or to a *certain* parameter.

The decision of when to make a query or not can be based on the use of different indicators[3]:

- `Variance`$= \sum_i (Q(m_i, \Pi(h, m_i)) - \hat{Q})^2$

  measures the variance over values computed by each model for the optimal action. Experiments show that this indicator captures efficiently how much learning remains to be done.

- `InfoGain`$= \sum_{k=1}^{n} [w_k \sum_{i,j \in S^2} [B_t(i,j)(\frac{1}{\sum_{k' \in S} \alpha^t_{A,i,k'}} + \frac{1}{\sum_{k' \in Z} \alpha^z_{A,j,k'}})]$

  measures the quantity of information that a query could bring[4]: It is equal to zero whenever a query won't provide new information (e.g. whenever the model is already sufficiently certain). Generally very useful to reject queries in these situations.

- `AltStateEntropy`$= \sum_{s \in S} -[\sum_{i=1}^{N} \beta_i(s)] \log(\sum_{i=1}^{N} \beta_i(s))$

  measures the entropy in the mean alternate belief. It is a good indicator of how much knowledge has been lost since the last query. Whenever the state is still well identified, non-query learning is appropriate, however when the alternate belief has high entropy, a query is called for.

- `NumberOfQueries` counts the number of queries already answered.

We combine these heuristics to decide when to perform *query* versus *non-query* learning, using a simple decision-rule. We do a query iff:

(AltStateEntropy $> \varepsilon_1$) AND (InfoGain $> \varepsilon_2$) AND ((Variance $> \varepsilon_3$) OR (NumberOfQueries $<$ Nmin))

The first condition ensures that no query should be done if enough information about the state is possessed because of previous queries. The second condition ensures that a query will not be made if it does not bring direct information about the model, which is the case if we are in a subpart of the model we know very well. The third condition ensures that we will stop doing queries when our model uncertainty does not have any influence on the expected return. Note for this term, considering the number of queries already done in necessary in some cases (especially in a learning setup with a completely uninformed prior) since at the beginning all the models can be equally bad, which can give a very low value for the Variance heuristic.

To do the non-query update of the transition alpha-parameters, we use the *alternate transition belief* $B_t(s,s')$ which is computed according to Equation 1: it is the distribution over the transitions that could have occurred in the last time step. The *non-query learning* then updates the corresponding state transition alpha-parameters according to Equation 2 [5]. On the other hand the observation alpha-parameters are updated proportionally to the new alternate mean belief state $\tilde{\beta}'$ defined by Equation 3 according to Equation 4.

$$\forall s, s' \; B_t(s,s') = \sum_{i=1}^{n} w_i \frac{[O_i]^z_{s',A} [P_i]^{s'}_{s,A} \beta_i(s)}{\sum_{\sigma \in S} [O_i]^z_{\sigma,A} [P_i]^{\sigma}_{s,A}} \quad (1)$$

$$\forall i \in [1 \ldots n] \forall (s,s') \; \alpha_t(s,A,s') \leftarrow \alpha_t(s,A,s') + \lambda B_t(s,s') \quad (2)$$

$$\forall s \; \tilde{\beta}'(s) = \sum_{i=1}^{n} w_i \beta'_i(s) \quad (3)$$

$$\forall i \in [1 \ldots n] \forall s' \in S \; \alpha_z(s',A,Z) \leftarrow \alpha_z(s',A,Z) + \lambda \beta'_i(s') w_i \quad (4)$$

3. Other heuristics were considered, including `PolicyEntropy` (entropy of the policy over models - biased indicated) and `BeliefVariance` (variance on the belief state - typically more noisy than Variance).
4. B_t is defined in Equation 1. Furthermore, we consider that $\frac{1}{\sum_{k' \in S} \alpha^t_{A,i,k'}} = 0$ ($\frac{1}{\sum_{k' \in Z} \alpha^z_{A,j,k'}} = 0$) when the parameters corresponding to transitions (observations) in state i, action A, are *certain*.
5. There is an alternative to this procedure. We can also sample a query result from the alternate transition belief distribution and thus update only one parameter. However, experimental results shows that this alternative is as efficient as the belief-weighted method.

1. Let $\lambda \in (0,1)$ be the learning rate. Initialize the necessary Dirichlet distributions. For any uncertain transition probability, $T^a_{s,\cdot}$, define $Dir \sim \{\alpha_1, \ldots \alpha_{|S|}\}$. For any uncertain observation parameter $O^a_{s,\cdot}$, define $Dir \sim \{\alpha_1, \ldots \alpha_{|Z|}\}$.
2. Sample n POMDPs $P_1, \ldots P_n$ from these distributions. (We typically use $n = 20$).
3. Compute the likelihood of each model: $\{p_{01}, \ldots p_{0n}\}$.
4. Solve each model $P_i \rightarrow \pi_i, i = 1, \ldots n$. We use a finite point-based approximation(12).
5. Initialize the history $h = \{\}$.
6. Initialize the belief for each model $b_1 = \ldots = b_n = b_0$ and the *alternate* belief $\beta_1 = \ldots = \beta_n = b_0$.
7. Repeat:
   (a) Compute the optimal actions for each model: $a_1 = \pi_1(b_1), \ldots a_n = \pi_n(b_n)$.
   (b) Randomly pick and apply an action to execute, according to the model weights: $a_i = \pi_i(b_i)$ is chosen with probability w_i where $\forall i \; w_i = \frac{p_i}{p_{0i}}$. p_i is the *current* likelihood model i has according to the Dirichlet distribution.
   (c) Receive the observation z.
   (d) Update the history $h = \{h, a, z\}$
   (e) Update the belief state for each model: $b'_i = b_i^{a,z}, i = 1..n$. Also update the alternate belief.
   (f) Determine if any learning should be done by using the InformationGain heuristic.
      - When it is the case, use the Variance heuristic and the NumberOfQueries heuristic to determine if a learning of good quality is required.
        – When it is the case, determine if a query has to be done by using the AlternateStateEntropy heuristic.
          ∗ If we have to make a query, make it, receive the query outcome, and update the Dirichlet parameters according to it. $\alpha(s,a,s') \leftarrow \alpha(s,a,s') + \lambda$ $\alpha(s',a,z) \leftarrow \alpha(s',a,z) + \lambda$
          ∗ Otherwise update the alpha-parameters using non-query learning according to equations 2 and 4.
        – Otherwise, use non-query learning according to equations 2 and 4 with a lower learning rate (we use $\lambda' = 0.01\lambda$).
   (g) When the Dirichlet structure has been modified, re-compute the model likelihoods p_i.
   (h) If we are in the non-stationary case, decay all the parameters concerned by the update by the parameter ν.
   (i) At regular intervals, draw another model P'_i according to the current Dirichlet distribution and add it to the set of models. Solve it: $P'_i \rightarrow \pi'_i$ update its belief $b'_i = b^h_0$, where b^h_0 is the belief resulting when staring in b_0 and seeing history h and its alternate belief, which takes into account what the latest query result was. The quality of the policy of the models sampled (the horizon considered and the number of belief points considered) is increased each time a model is sampled.
   (j) If the number $N_{\max}$ of models is reached, remove from the set of models the model P_i with the lowest likelihood.

Table 1: The MEDUSA algorithm

3.4 Handling non-stationarity

There are many interesting robotics domains where the model parameters may change over time. For example, slow decay in the wheel alignments may change the robot motion model, or a person interacting with the robot may change preferences over time. We would like our learning algorithm

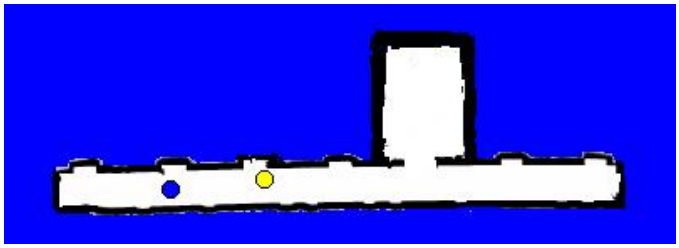


Figure 1: Map of the environment used for the robot simulation experiment.

to be able to handle such non-stationarity in the model parameters. This is not typically possible in the standard POMDP formulation.

In MEDUSA, however, non-stationarity can easily be accommodated by simply allowing recent experience to be weighed more heavily than older experience. To do this, each time we update one of the hyper-parameter, we multiply all the hyper-parameters corresponding to the associated multinomial distribution by $\nu \in]0;1[$, the *model discount factor*. This can be thought of as a decay weight over model certainty. It does not change the most likely value of any of the updated parameters (Section refsec:dirichlet), but does diminish confidence over the parameters. Note that the *equilibrium confidence* is defined as: $C_{max} = \lambda_{\frac{1}{1-\nu}}$, which is attained after an infinite number of samples, and is an indicator of our confidence in past experience. This is high when we believe the model is stable. This concludes our description of the MEDUSA algorithm. Table 1 provides a detailed description of MEDUSA, including implementation details. We now proceed with a description of experimental validation.

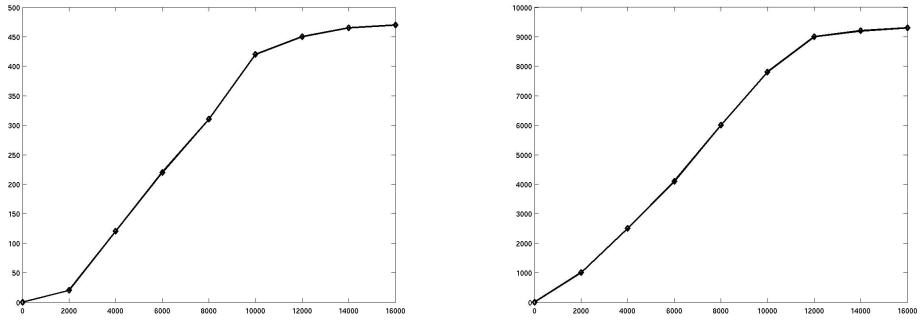
4. Experimental results

In previous work, we validated the performance of MEDUSA on standard problems from the POMDP literature (6). We now turn our attention to the application of MEDUSA in a realistic robot simulation domain. In order to do this, we combined MEDUSA with the Carmen robot simulator (11). The Carmen toolbox has been widely used in the robotics community for the control of indoor mobile robots. Its simulator is known to be highly reliable and policies with good simulation performance can typically be ported without modification to the corresponding robot platform. When linking MEDUSA to Carmen, in effect, we require that all data and model priors used by MEDUSA be generated by Carmen.

Our experiments thus far have focused on a scenario where the robot must navigate in an environment, with the goal of finding a person who is hiding in this environment. This problem has been studied before under various names (Hide, Tag, Find-the-patient). Previous work considered at a fully modeled version of this problem, where the person's location is unknown, but the person's motion model is precisely modeled, as are the robot's sensor and motion models. We now consider the case where in addition to not knowing the person's position, we are also uncertain about the person's motion model and the robot's sensor model. In total, MEDUSA is trying to learn 52 distinct parameters. We consider the environment shown in Figure 1. Planning and learning are done over a discretized version; the associated POMDP has 362 states, 24 observations and 5 actions.

As we can see from Figure 2, MEDUSA converges within roughly 12,000 time steps, after having received answers to approximately 9,000 queries. This is orders of magnitude faster than

experience-based approaches, which can require millions of steps to learn problems with less than a dozen states. We note however that experience-based approaches do not require an oracle. The high number of queries required by MEDUSA for this problem is in large part a consequence of the fact that the initial model prior is completely uninformed. Using a more informed prior would lead to faster learning, but would require more knowledge engineering (we are currently conducting experiments to demonstrate this empirically). In the end, it's debatable whether it's preferable to provide more precise priors, or require more data labelling. To reduce the number of queries, we could also build a simpler model with fewer alpha-parameters. The main purpose of these preliminary results is to show that MEDUSA can in fact learn models for problems with hundreds of states and that the approach is applicable to realistic robotic domains. MEDUSA's flexible approach to knowledge-engineering (i.e. combination of model priors, labelled data a.k.a. queries, and non-labelled data a.k.a. non-query learning) make it particular attractive for real-world domains where parts of the problem can be specified differently.



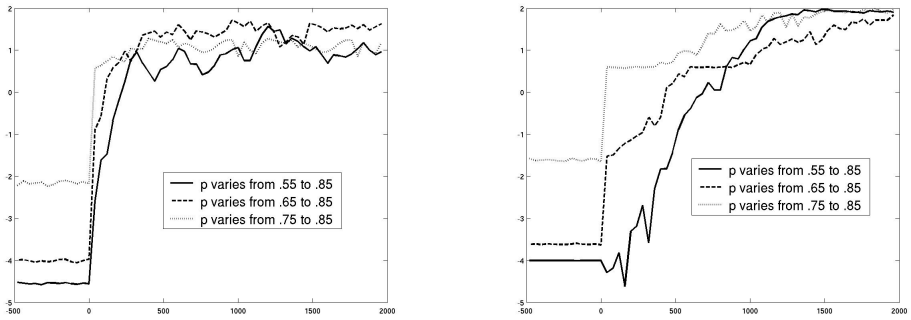
(a) (b)

Figure 2: Results for the robot simulation domain. (a) Evolution of the discounted reward with the number of time steps. (b) Evolution of the number of queries with the number of time steps.

Next, we consider the question of whether MEDUSA is robust to sudden changes in model parameters. This arises in non-stationary environments, where parameters can either (1) slowly drift over time (e.g. slow decay in the wheel alignments), or (2) change abruptly (e.g. person has an injury, which suddenly slows down his/her walking speed). Throughout these cases, MEDUSA should adapt to changes and learn the new correct model with high confidence. If the change in parameters is small, then non-query learning is sufficient, however if there are large changes, it is necessary to resort to queries.

We have not yet investigated MEDUSA's ability to handle non-stationary environments in the robot scenario, however we have performed preliminary tests on the standard Tiger domain (8), where we assume that the probability of correctly detecting the target (or tiger) suddenly changes. Figure 3 summarizes the results. As expected, the speed at which MEDUSA learns the new model depends on the confidence (sum of Dirichlet parameters). When the confidence is low ($<$100), the agent quickly adapts to the new parameters, even when there is a large shift in the model. However when confidence is high ($>$1000), the agent takes many more steps to learn new parameter values.

Finally, we investigate MEDUSA's robustness to errors in the query responses. MEDUSA's query learning assumes that an oracle can provide exact state identification on demand. However it



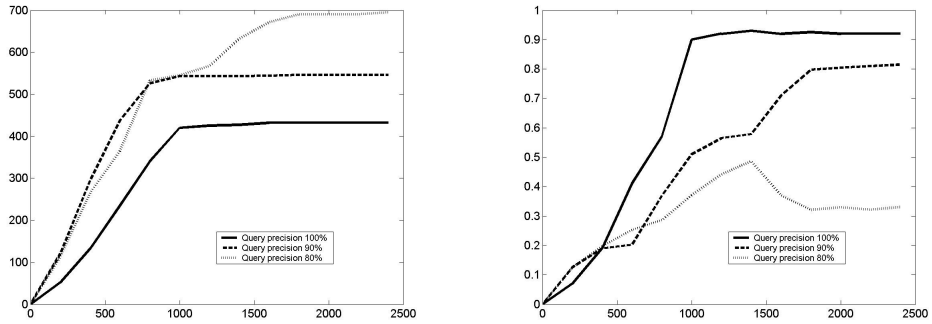
(a) (b)

Figure 3: Plot of the evolution of the discounted reward as a function of the number of time steps. There is a sudden change in the parameter p at time 0. (a) The equilibrium confidence before the change is low (100). (b) The equilibrium confidence before the change is high (1000).

is more realistic to assume some amount of noise in the state identification provided by the oracle. This allows us to broaden the class of information sources we can use an oracle. For example in robotics, one could use a high-precision (but expensive or obtrusive) sensor to answer queries while the model is being built, and then remove the sensor for standard operation. We don't yet have results for robustness to query noise in the Carmen simulator. However we have investigated the issue in the context of the standard Hallway domain (9). We consider two cases. In the first, whenever a query is made, there is a 10% probability that the oracle will identify the wrong state (picking one at random), and a 90% that the oracle will give the correct answer. In the second case, the probability of a correct answer is 80%. As we see from Figure 4, the quality of the learned model is quite good when the oracle is correct in 90% of queries, though it takes more steps of query learning (and thus fewer steps of non-query learning) than when the oracle is always correct. The performance is significantly degraded for the higher error rate. However the 80% query precision model may not have converged yet, as we see in Figure 4b that the number of queries is still climbing.

5. Discussion

The work we describe bears some resemblance to earlier techniques for decision-making in model-free POMDPs, which can learn and plan despite uncertainty in both state and model (3; 10; 15; 17; 2; 16). However these approaches have not scaled to large-scale domains, due to their intensive data requirements. MEDUSA offers a more flexible trade-off between knowledge engineering and data requirements. Initial knowledge can be introduced through model priors. In addition MEDUSA narrows the learning requirements by selecting when to do queries. Finally, unlabelled data can also be leveraged through non-query learning. We show that this work is applicable to learning a complex robot task. MEDUSA can also solver a broader class of decision-making problems than earlier experience-based since it can handle non-stationary environments. And while the oracle assumption is a strong one, we show results demonstrating that MEDUSA is robust to noise in the query answers. We are currently extending the results for the non-stationary case and the noisy queries to the robotics domain.



(a) (b)

Figure 4: Results for queries with a noisy oracle. (a) Plot of the discounted reward as a function of the number of time steps. (b) Plot of the number of queries as a function of the number of time steps.

References

[1] Anderson, B. and Moore, A. "Active Learning in HMMs" ICML, 2005.

[2] Brafman, R. I. and Shani, G. "Resolving perceptual aliasing with noisy sensors" NIPS, 2005.

[3] Chrisman, L. "Reinforcement learning with perceptual aliasing: The perceptual distinctions approach" AAAI, 1992.

[4] Cohn, D. A., Ghahramani, Z. and Jordan, M. I. "Active Learning with Statistical Models" NIPS, 1996.

[5] Dearden, R.,Friedman, N.,Andre, N., "Model Based Bayesian Exploration" UAI, 1999.

[6] Jaulmes, R.,Pineau, J.,Precup, D., "Active Learning in Partially Observable Markov Decision Processes" ECML, 2005.

[7] Jaulmes, R.,Pineau, J.,Precup, D., "Learning in non-stationary Partially Observable Markov Decision Processes". ECML Workshop on Reinforcement Learning in non-stationary environments, 2005

[8] Kaelbling, L., Littman, M. and Cassandra, A. "Planning and Acting in Partially Observable Stochastic Domains" Artificial Intelligence. vol.101, 1998.

[9] Littman, M., Cassandra, A. and Kaelbling, L. "Learning policies for partially obsevable environments: scaling up". ICML, 1995.

[10] McCallum, A. K. "Reinforcement Learning with Selective Perception and Hidden State" Ph.D. Thesis. University of Rochester, 1996.

[11] Montemerlo, M., Roy N. and Thrun, S. "Perspectives on Standardization in Mobile Robot Programming: The Carnegie Mellon Navigation (CARMEN) Toolkit". IROS, 2003.

[12] Pineau, J., Gordon, G. and Thrun, S. "Point-based value iteration: An anytime algorithm for POMDPs" IJCAI, 2003.

[13] Poupart, P. and Boutilier, C. "VDCBPI: an Approximate Scalable Algorithm for Large Scale POMDPs" NIPS, 2005.

[14] Roy, N., Gordon G. and Thrun S. "Finding Approximate POMDP solutions Through Belief Compression". Journal of Artificial Intelligence Research, 23: 1-40, 2005.

[15] Shatkay, H., Kaelbling, L. "Learning topological maps with weak local odometric information" IJCAI, 1997.

[16] Shani,G.,Brafman,R.I.,Shimony,E. "Model-Based online learning of POMDPs", ECML, 2005.

[17] Singh, S., Littman, M., Jong, N. K., Pardoe, D., and Stone, P. "Learning Predictive State Representations" ICML, 2003.

[18] Spaan, M. T. J. Spaan, and Vlassis, N. "Perseus: randomized point-based value iteration for POMDPs". Journal of Artificial Intelligence Research, 2005.