Active learning in Partially Observable Markov Decision Processes

by

Robin Jaulmes

Department of Computer Sciences
McGill University
Montreal

A thesis submitted to
McGill University
in partial fulfillment of the requirements for the degree of
Master of Science

Your file Votre référence
ISBN: 978-0-494-24702-0
Our file Notre référence
ISBN: 978-0-494-24702-0

ABSTRACT

People are efficient when they make decisions under uncertainty, even when their decisions have long-term ramifications, or when their knowledge and their perception of the environment are uncertain. We are able to experiment with the environment and learn, improving our behavior as experience is gathered. Most of the problems we face in real life are of that kind, and most of the problems that an automated agent would face in robotics too.

Our goal is to build Artificial Intelligence algorithms able to reproduce the reasoning of humans for these complex problems. We use the Reinforcement Learning framework, which allows to learn optimal behaviors in dynamic environments. More precisely, we adapt Partially-Observable Markov Decision Processes (POMDPs) to environments that are partially known.

We take inspiration from the field of Active Learning : we assume the existence of an oracle, who can, during a short learning phase, provide the agent with additional information about its environment. The agent actively learns everything that is useful in the environment, with a minimum use of the oracle.

After reviewing existing methods for solving learning problems in partially observable environments, we expose a theoretical active learning setup. We propose an algorithm, MEDUSA, and show theoretical and empirical proofs of performance for it.

RÉSUMÉ

Les humains sont efficaces lorsqu'ils prennent des décisions en présence d'incertitude, y compris dans les cas où leurs actions peuvent avoir des conséquences à long terme, ou lorsque leur connaissance et leur perception de leur environnement est partielle. Nous sommes capables d'expérimenter avec notre environnement et d'apprendre. La plupart des problèmes auxquels l'on fait face dans la vie sont de ce type, et pour un agent automatisé, la problématique est souvent rencontrée en robotique.

Notre but est de construire des algorithmes d'Intelligence Artificielle capables de reproduire le raisonnement des humains pour ce type de situation. Pour cela nous utilisons l'Apprentissage par Renforcement, qui permet d'apprendre des comportements optimaux dans les environnements dynamiques, et plus précisément, nous utilisons les Processus Décisionnels de Markov Partiellement Observables (PDM-POs) et les adaptons aux environnements incertains.

Nous utilisons les techniques d'Apprentissage Actif : nous faisons l'hypothèse qu'il existe un oracle capable, pendant une courte période d'apprentissage, de fournir sur demande de l'information additionnelle à l'agent. L'agent apprend activement tous les paramètres utiles de son environnement en utilisant un nombre minimal de requêtes.

Nous présentons les méthodes qui existent pour résoudre les problèmes d'apprentissage dans les environnements non-stationnaires, puis nous expliquons comment nous envisageons la résolution de la problématique d'un point de vue théorique. Nous proposons ensuite un algorithme, MEDUSA, et fournissons des preuves théoriques et expérimentales de sa performance.

Contents

List of Figures

List of Tables

Aknowledgements

I first want to thank McGill University for giving me the possibility of doing graduate studies and research. I thank professors Joelle Pineau and Doina Precup for their advice and their supervision during my stay, and for their financial support during my research, as I thank McGill for its Top Up fellowship.

Furthermore, I thank everyone in the Reinforcement and Learning laboratory, for their help and moral support: Marc, for letting me attempt to play tennis, and Mark for his efficient management of the computational ressources of the lab, Philipp for its fantastic implementation of the Walkthrough game and help in the COMP-424 management, Erick for being a great billiard player, Masoumeh for keeping me company in Porto, and her graduate knowledge and Bohdana, whose thesis was of great help for the making of this document. I don't forget Dan, Adrian, Melanie, Amin, Norm, Chris Pablo, and of course professor Prakash Panagaden for his good humor and efficient presentations. I also thank the other graduate students who shared their courses with me: Derek, Ryan, Dave and Ran, and all those that spent time with me in Montreal. I also thank everyone I met at ECML, whose comments and questions about my work where of great help for improving it.

I also thank Jean-Baptiste for allowing me to go skiing by the coldest temperature I had ever experienced, Nicky for getting me to see whales, Isabelle for moral support, and my parents, for their love and care, and I also thank all my family and friends who stayed in France and kept in touch with me.

Chapter 1

Introduction

Making an optimal decision is a very complex process. Humans are usually good at this. But from a theoretical point of view the problem can be very hard, and finding an automated way to make optimal decisions can be challenging. The main difficulties encountered when someone makes a decision are the following:

1. **Complex goals.** It can be hard to specify what exactly is an optimal decision. The goal can be to maximize immediate gain, long-term gain, a combination of the two, or to reach a particular state of the environment.

2. **Stochasticity.** Actions can have non-deterministic consequences. All the possible outcomes of a given action may need to be considered.

3. **Long-term consequences** Actions can have long-term consequences, that might not be easy to determine at first thought. A good action in a short-term horizon may end up being bad if we don't consider long-term consequences.

4. **Partial Observability** The current state of the environment might be only partially observed. Actions might need to be taken for the sole purpose of

bringing more information about the state of the environment and help improve future decisions.

5. **Uncertain Environment** The knowledge of the environment might be uncomplete. Actions might need to be taken for the sole purpose of finding out their consequences, so that the model of the environment is improved, and better decisions be taken in the future.

The issue of optimal decision-making has been well studied in the literature and solutions have already been found for these mentioned facts (?). Classical planning techniques and Markov Decision Processes (Sutton and Barto, 1998) are among them.

This thesis considers problems in which all these points arise at the same time. We actually believe that to be effective a decision-maker has to take them all into account, as all of them arise very often in real life, and can arise with complex robotics settings.

In order to solve this problem we focus on two fields of the literature: the first is the field of Partially Obsevable Markov Decision Processes (POMDPs), which allow to behave optimally in partially observable environments, and the second is Active Learning, which allows to learn optimally in uncertain environments. Our goal is to combine them.

1.1 Active Learning and POMDPs

Reinforcement Learning (Sutton and Barto, 1998) is a computational approach that allows a software agent to learn a way of maximizing its reward by making sequential decisions in stochastic dynamic environments which are characterized by a particular *state* at any given time. The Reinforcement Learning field gives a way of choosing

actions, called a *policy*, so that the overall reward is maximized. The agent's policy takes into account the stochasticity of the environment's dynamics.

Within the reinforcement learning field, **Partially Observable Markov Decision Processes** (Kaebling *et al.*, 1998) **(POMDPs)** allow an agent to make optimal sequential decisions in partially observable environments. They combine complex goals, stochastic effects of actions, long-term consequences of actions and partial observability of the environment and can produce efficient policies. However POMDPs cannot give an optimal behavior for cases where the environment is not perfectly known.

On the other hand, the field of **Active Learning** (Cohn *et al.*, 1996) is able to give methods to try the most useful actions from a learning point of view. Given an uncertainty model, active learning approaches are able to find the experiment that most reduces uncertainty in the model.

Active learning methods usually suppose that perfect information can be obtained, and obtaining this information can be difficult in partially observable problems. In order to get this information, and ensure that the parameters are learned correctly, we need to make an assumption, which is the existence of an **oracle**. This oracle can reveal the hidden state of the POMDP, upon request.

Using this assumption allows an optimal active learning approach. With it, we can guarantee that the parameters of the environment are learned precisely and fast. We are able with this setting to make the perfect trade-of between the active learning of the environment and its exploitation. However, even if the method would be theoretically optimal, it is intractable from a computational point of view.

In order to correct this, we build an algorithm able to find an approximate solution to the problem. This algorithm, called **MEDUSA**, for "Markovian Exploration with Decision based on the Used of SAmples", inspired from the Bayesian Learning

techniques, uses combinations of approximate solutions of POMDPs, combined to a model of the uncertainty, to find decisions that are good from an exploratory point of view, while using the oracle to learn the model of the environment.

1.2 Thesis Outline

The outline of our thesis is the following:

- In Chapter 2, we explain the elements of background material. We explain the POMDP framework in detail, we present methods that can be used to act optimally and to learn in a POMDP framework. We also present the Active Learning field and the Bayesian Learning perspective.
- In Chapter 3, we explain how we can theoretically combine Active Learning and POMDPs, and why it is a very difficult problem to solve exactly.
- In Chapter 4, we explain the MEDUSA algorithm for Active Learning in POMDPs, how it works and intuitive justification of its mechanisms.
- In Chapter 5, we explain why MEDUSA is guaranteed to converge under certain conditions, and we show theoretical properties of the policy it executes in the limit.
- In Chapter 6, we present our experimental results with MEDUSA on different problems, from very simple settings to problems inspired from possible robotics applications.
- In Chapter 7, we conclude our Thesis by explaining what our contributions are, and what would be possible areas of future work.

Chapter 2

POMDPs and Learning

Our work builds on several frameworks and ideas. This chapter describes them in detail. Our main basis is the POMDP framework, which allows optimal sequential decision-making in partially observable stochastic domains. We show existing methods to solve and learn POMDPs, and discuss their limitations. Then, we explain what are Active Learning and Bayesian Learning.

2.1 POMDPs

When an agent is confronted with a decision, there is a number of different alternatives (**actions**) it can choose from. Choosing the best action requires thinking about more than just the immediate effects of its actions. The immediate effects are often easy to see, but the long term effects are not always as transparent. Sometimes actions with poor immediate effects can have better long term ramifications. An gent should choose the action that makes the right tradeoffs between the immediate rewards and the future gains, to yield the best possible solution.

What usually makes this particularly difficult is that there is uncertainty about the future. The outcome of certain actions may not be entirely predicable, and sometimes one doesn't even know if the action will even matter in the future.

The **state** is the way the world currently exists and an action will usually have the effect of changing the state of the world. Note that in the POMDP framework, we make the ***Markovian* assumption**, which states that the state is sufficient to summarize everything that happened in the past. The previous states we went through, in a POMDP, need not be considered.

In a POMDP the state of the world is not perfectly known to the agent. However it is partially observed. At each time step, before each decision, the agent obtains an **observation**, which is probabilistically linked to the current state of the world and the action that was last done.

POMDPs are able to model problems so that the process of decision making in uncertain environments is formalized. If the problem is modelled as a POMDP, one can use algorithms to automatically solve the decision problem.

2.1.1 Definition

A POMDP is made of the following components (Sondik, 1971):

A set of States S: This set gathers all the possible ways the world could be. It is usually assumed to be discrete and finite.

A set of Actions A: The actions are the set of possible alternative choices the agent can choose to make at each time step. This set is also assumed to be discrete and finite.

A set of observations Z: These are the set of all the different outputs the agent can receive at each time step from the environment. This set is also assumed to be discrete and finite.

Transition probabilities:
$\{P^a_{s,s'}\} = \{p(s_{t+1} = s' | s_t = s, s_t = a)\}, \forall s \in S, \forall a \in A, \forall s' \in S$
These parameters describe how the different actions affect the state of the world. In a POMDP the effects of an action can be probabilistic: for each state/action pair $[s, a]$, the parameters $P^a_{s,s'}$ correspond to the probabilities of the state of the world resulting in each of the different states s'. We assume that time passes in uniform, discrete intervals, at each transition.

Observation probabilities:
$\{O^a_{s,z}\} = \{p(z_t = z | s_t = s, a_{t-1} = a)\}, \forall z \in Z, \forall s \in S, \forall a \in A$
These parameters describe how the environment outputs its different observations. In a POMDP the observation is linked to the last executed action a and to the current state of the world s. For a given state/action pair $[s, a]$, the probabilities $O^a_{s,z}$ correspond to the probabilities for each of the possible observations z.

Immediate Rewards: $R : S \times A \times S \times Z \rightarrow \mathbf{R}$ This is some measure of either the cost or the reward an action gives when a certain transition occurs. This is usually a function of s,a, s' and z. When the transition $[s, a] \rightarrow s'$ occurs and observation z is perceived, the immediate reward is equal to $R(s, a, s', z)$. In order to use the resolution algorithms, it is usually required that these immediate rewards be positive. However if they are negative they can be easily shifted.

The Discount Factor: $\gamma \in [0; 1]$ This factor represents the decrease of utility when a given immediate reward is received at time $t + 1$ instead of time t. It is an indicator of the trade-of between long-term reward and short-term reward.

The Initial Belief: $b_0 \in \mathbf{R}^{|S|}$. It is the distribution over the possible states of

the environment at time 0.

At each time step, the agent is in an unknown state $s_t \in S$. It chooses and executes an action $a_t \in A$, arrives in an unknown state $s_{t+1} \in S$ and observes $z_{t+1} \in Z$, which is probabilistically linked to s_{t+1} and a_t.

Agents using the POMDP framework to make their decisions usually maintain a **belief state** $b \in \mathcal{B}^1$, which is the distribution over the states, taking into account the initial belief state b_0 and the whole history of actions and observations. In order to maintain the belief state, we need to do, each time action a is executed and observation z is obtained, a **Bayesian update**, which has the following equation: ($b_{a,z,b}$ is the new belief state and b is the old belief state):

$$\forall s \, b_{a,z,b}(s) = \frac{O^a_{s,z} \sum_{s0 \in S} P^a_{s0,s} b(s0)}{\sum_{s1,s2 \in S^2} O^a_{s2,z} P^a_{s1,s2} b(s1)} \tag{2.1}$$

This allows to compute, for a given initial belief and a given history of actions and observations, the current belief b_t.

A **policy**: $\pi : [0;1]^{|S|} \rightarrow A$ is a function that gives for each possible belief state the corresponding action.

To **solve** a POMDP is to find the policy π^* that maximizes the **return** ρ,defined by:

$$\rho(\Pi) = E[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t = \Pi(b_t), s_t + 1, z_t)]$$

[1]The belief space $\mathcal{B}$ is actually the subset of $[0;1]^{|S|}$ where the condition $\sum_{i=1}^{|S|} b(i) = 1$ is verified. $b(i)$ is equal to the probability of the agent being in state i

Note that since R is bounded and $\gamma < 1$ the return is bounded.

The value function in the type of $V : [0;1]^{|S|} \rightarrow \mathbf{R}$ associates to a given belief state the expected return the agent can have if it follows an optimal policy starting in it. Its main property is that for a given horizon length the value function is a piecewise-linear convex function over the belief state (Sondik, 1971).

POMDPs can be used for many different applications, in domains like industry: elevator control policies (Crites, 1996), network maintenance (Thiebeaux *et al.*, 1996), helicopter control (Bagnell and Schneider, 2001), accounting (Kaplan, 1969)), high-level decision making in robotics (Simmons and Koenig, 1995; A. Cassandra *et al.*, 1996; Nourbakhsh *et al.*, 1995), gesture recognition (Darrell and Pentland, 1996), dialogue management systems (Roy *et al.*, 2000), medical diagnosis (Hauskrecht, 1997), and medical treatment policies (Smallwood *et al.*, 1971).

For example, in a robotics navigation task, we use the following setting. The state of the world is the position and orientation of the robot. The robot itself cannot know it directly, since it only has access to its sensors. All the possible sensor outputs (laser sensors, sonar sensors) become the set of observations. The set of actions is the set of all the possible control inputs we can give. A positive reward can be associated to the robot reaching a given position.

The POMDP framework can allow the robot to infer what its current believed position is by considering the sequence of actions and observations it experienced since the beginning of the run. Furthermore, an optimal policy in a POMDP framework can allow the robot to be cautious in its movements, because actions like staying in the same place in order to have more precise sensor readings can become optimal. Therefore the POMDP framework is very well adapted in cases where the sensors are not of good quality. It can also solve problems like perception aliasing (in which different parts of the environment produce the same observation).

We now describe a classical simple POMDP that will be used as an example to illustrates our ideas and algorithms. It is the **Tiger** problem (Kaebling *et al.*, 1998). In this problem, we have two doors. Behind one of them we have a tiger and we don't know behind which one. The tiger does not move. We can try to listen in order to determine behind which door is the tiger, but we're not guaranteed to hear correctly. We have a reward of -100 if we open the door with the tiger and $+10$ if we open the correct one. Each time we open a door the problem resets.

More formally, the parameters of the Tiger POMDP are the following:
$S = \{\text{TigerLeft,TigerRight}\}$
$A = \{\text{Listen,OpenLeft,OpenRight}\}$
$Z = \{\text{HearLeft,HearRight}\}$
The parameters of the model are described in Table 2.1.

The initial belief is $\{0.5; 0.5\}$. Each time one of the two "open door" actions is chosen the belief state comes back to $\{0.5;0.5\}$[2]. However, when the Listen action is performed, the belief state is modified. If HearLeft is obtained, according to the Bayesian update rule, the belief state becomes $\{0.85; 0.15\}$ (it is likelier that the state is TigerLeft). On the other hand, if HearRight is obtained it is likelier that we are in TigerRight. The more we obtain HearLeft the likelier the state TigerLeft is, and it is the same for HearRight and TigerRight. The optimal policy actually, whose policy graph is shown on Figure 2.1 (taken from (Kaebling *et al.*, 1998)) takes count on how many HearLeft and HearRight are obtained. When the difference between the counts is greater than 2, we open the corresponding non-tiger door.

[2]Because the received observation is uniformly random and the transitions are $\{0.5; 0.5\}$.

Transitions	Action Listen		Action OpenLeft		Action OpenRight	
From \ To	TigerLeft	TigerRight	TigerLeft	TigerRight	TigerLeft	TigerRight
TigerLeft	1.0	0.0	0.5	0.5	0.5	0.5
TigerRight	0.0	1.0	0.5	0.5	0.5	0.5
Observations	Action Listen		Action OpenLeft		Action OpenRight	
State \ Obs.	HearLeft	HearRight	HearLeft	HearRight	HearLeft	HearRight
Tigerleft	0.85	0.15	0.5	0.5	0.5	0.5
TigerRight	0.15	0.85	0.5	0.5	0.5	0.5
Rewards	Action Listen		Action OpenLeft		Action OpenRight	
Tigerleft	−1		−100		+10	
TigerRight	−1		+10		−100	

Table 2.1: Parameters of the Tiger problem

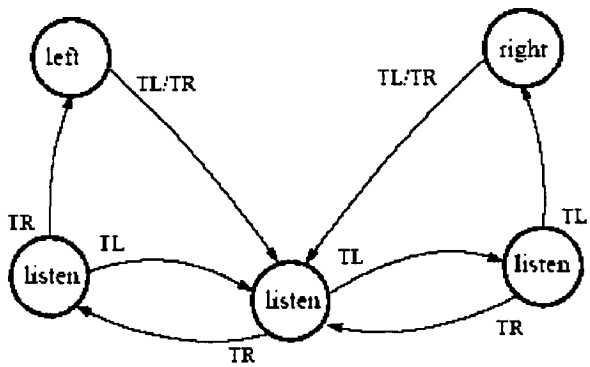


Figure 2.1: The optimal policy for the Tiger Problem.

2.1.2 Solving POMDPs

Finding the exact solution in an infinite horizon POMDP is very costly, but many methods can find approximate solutions. We distinguish three kinds. Heuristic methods (Cassandra, 1998; Roy and Thrun, 1999) are fast methods that focus on solving the fully-observable problem and then using heuristics to make the decision. They are very fast, but completely fail in certain cases; in particular, they don't consider the effect an action has on the precision of the knowledge. Value-based methods (Kaebling *et al.*, 1998), which are more interesting, compute an exact or approximate value function in order to determine the policy. They are intractable on big problems when exact, and suboptimal when approximate. Policy search methods (Ng and Jordan, 2000), on the other hand, search through the policy space without computing values. They give near-optimal policies fast, but necessitate a prior expression of the optimal policy.

Value-based methods Most methods to solve POMDPs estimate the **value function** over the belief state.

For a given horizon length the value function is a piecewise-linear convex function over the belief state (Sondik, 1971), which is therefore usually represented as a **set of hyperplanes** $\mathbf{H} = \{H_j\}$. Each hyperplane is represented as a vector H_j of dimension $|S|$ such that $H_j(b) = \sum_i b(i)\mathrm{H}_j(i) = \mathrm{b} \cdot \mathrm{H}_j$. The value at a given belief state point is equal to the value on the highest hyper-plane.

$$V(b) = \max_{H_j \in \mathbf{H}} (\mathrm{b} \cdot \mathrm{H}_j) \tag{2.2}$$

Note that each hyperplane is also associated with an action. So the optimal policy associated to a given value function is to execute the action corresponding to the highest hyperplane in the current belief point.

When we initialize the algorithm, we build $|A|$ hyperplanes, each hyperplane being associated to one of the actions: hyperplane H_i is such that its value in the parts of the state space where a state is certain it is equal to the expectation of the immediate reward associated to action a_i for that state.

$$\forall a \in \{1 \dots |A|\} \mathrm{H}_a(i) = \sum_{z \in Z} \sum_{s' \in S} P^a_{s,s'} O^a_{s',z} R(a, s_i, z, s') \tag{2.3}$$

Then at each time step we consider a finite number of belief points, which are usually a representative set of the belief points that can be reached. For each of these, we consider, for each action-observation pair associated with a non-zero probability, the associated resulting belief states, and see what the current value is for this state (as we said before, this value is equal to the maximum value over all the different hyperplanes. Then we add this value, multiplied by the discount factor, to the value of the point we were considering. To obtain the new value for our point for a given action, we simply do the weighted sum over the possible observations of these resulting action-observation values. The values are weighted by the probability of each observation. Then, to finally find the value of our belief point, we pick the value of the best action, which becomes the action associated to the hyperplane that goes through our point. The value update equation is the following:

$$V^{new}(b) = \max_{a \in A} \sum_{z \in Z} \sum_{s,s' \in S^2} b(s) P^a_{s,s'} O^a_{s',z} [R(a, s, z, s') + \gamma V old(b_{a,z,b})] \tag{2.4}$$

Here, $b_{a,z,b}$ is defined according to Equation 2.1.

Using Equation 2.4 for every belief point allows to find the new value function.

Usually, algorithm prefer to manipulate directly hyperplane vectors. The setup is usually the following: The intermediate hyperplane sets $\Gamma^{a,o}$ are computed:

$$\forall a \in A \forall z \in Z \; \Gamma^{a,o} = \{\gamma \sum_{s' \in S} P^a_{s,s'} O^a_{s',z} \mathrm{H}_j(s'), \forall \mathrm{H}_j \in \mathbf{H}\} \tag{2.5}$$

The new set of hyperplane becomes:

$$\forall a \in A \, H^{new} = \cup_{a \in A}[R(\cdot, a) + \Gamma^{a,o1,j} \oplus \Gamma^{a,o2,j} \ldots \Gamma^{a,o|Z|,j}] \tag{2.6}$$

Here, $\oplus$ represents the cross-sum operator[3], $R(\cdot, a)$ represents the vector such that $R(\cdot, a)(i) = \sum_{s' \in S, z \in Z} P^a_{i,s'} O^a_{s',z} R(i, a, s', z)$.

The resulting set of hyperplanes unfortunately contains too many hyperplanes. Some of them are completely dominated. Finding out which one are completely dominated to prune them out is necessary, and the easiest way to do it is to use linear programming.

Algorithms can use different variations of this method. Instances of such algorithms include: witness (Kaebling *et al.*, 1998), incremental pruning (Zhang and W.Liu, 1996),two-pass (Sondik, 1971), PBVI (Pineau *et al.*, 2003), heuristic value iteration (Smith and Simmons, 2004), PERSEUS (Spaan and N.Vlassis, 2005), grid-based value iteration (Brafman, 1997). PBVI and grid-based value iteration and heuristic-based value iteration are the most efficient algorithms the literature can offer. However, they are usually tractable only when the horizon is finite.

Heuristic methods The idea behind these methods is simple. First, the optimal policy for the fully-observable problem is computed, using a classical MDP-solving method (Singh *et al.*, 2003). The policy π giving the best action for any given state is obtained. The following decision-making heuristics can then be used.

[3]For instance let $A = \{a1, a2\}$ and $B = \{b1, b2\}$. Then $A \oplus B = \{a1+b1, a1+b2, a2+b1, a2+b2\}$.

- **Most likely**: $\Pi(b) = \pi[\arg\max(b(s))]$ This heuristic takes the action recommended by π for the most likely state.
- **Voting**: $\Pi(b) = \arg\max[\sum_{s,\pi(s)=a} b(s)]$ This heuristic takes the action that is recommended in the majority of cases.
- **Q-MDP**: $\Pi(b) = \arg\max \sum_s b(s)Q(a,s)$. $Q(s,a)$ here is the expected return of performing action a in state s. This heuristic allows to consider all the possible states and all the possible values a given action could have in these states.

There also exist other heuristic methods, like the **Augmented MDP** (Roy and Thrun, 1999) method, that introduces in a MDP an additional state feature corresponding to the amount of entropy[4] in the belief state, which is a measure of state uncertainty, and plans on the augmented state space.

Policy search methods Some methods do not try to learn the value function of the POMDP. For instance, the PEGASUS algorithm, designed by Ng and Jordan (Ng and Jordan, 2000), considers a class of policies and optimizes its parameters. Their method involves the transformation of the stochastic POMDP into a deterministic POMDP, in which all the transitions are deterministic (a new feature, which represents the "dice rolls" of the different transitions, is added to the state space). Using this deterministic POMDP they are able to maximize the return by varying the parameters of their policy class. This method is able to solve very large POMDP problems. However a good policy class needs to be provided.

[4]The entropy is a classical measure of uncertainty, it is defined by $E = -\sum b(i)\log(b(i))$.

2.1.3 Learning POMDPs

We now study methods that can learn POMDP parameters. These methods usually learn the parameters by interacting with the environment. We present three types of methods. The first kind try to find out, for a given trace of experimentation through the environment, what was the most likely hidden states visited. From them it computes what are the most likely parameters of the model. It is a simple learning algorithm that may find unprecise values for the parameters. The second type finds out the underlying structure of the POMDP by resolving perceptual aliasing ; it can learn the setup of the states but need to have a prior knowledge of the observation probabilities. The third type is purely experience-based. It maintains the probabilities of "core tests" of succeeding and how the different actions affect the core test probabilities. It is able to learn a POMDP without any prior at all but needs large amounts of experimentation.

Expectation-Maximization methods This straightforward method uses gradient descent to find the model that explains best the sequence of actions and observations (Chrisman, 1992; Shatkay and Kaelbling, 1997; Kaebling *et al.*, 1998). The method alternates between: (a) using the Baum-Welch algorithm (Baum, 1972) to find the most likely underlying states at each time step in the experiment given the current set of parameters and (b) computing the most likely parameters for the transitions and observations by taking counts, considering the current assignment of real states. Such a setting is prone to local minima. There is no guarantee that the obtained model will be correct or that the corresponding policy will be optimal, and for complex problems with many local minima the correct policy/model is rarely found.

Utile Suffix Memory methods McCallum built the *Utile Suffix Memory* algorithm (McCallum, 1995) to learn Perceptual Aliasing Problems, which are a sub-class

of POMDPs in which the observation is always deterministic. Recently, Shani et al. presented methods to adapt this setting to learn POMDPs. In (Shani and Brafman, 2004) they analyzed how they could modify this framework to take into account probabilistic observations. In (Shani *et al.*, 2005b) they showed how POMDPs could be learned from experience when the learner has a prior knowledge of the observation probabilities of the different kinds of states there can be in the environment. In their method they use USM to determine what the different states and state transitions probabilities are, and an on-line version of the Perseus algorithm (Spaan and N.Vlassis, 2005) to build a policy for the POMDP model of the environment. The same group also has a method to adapt a POMDP policy to non-stationary environments with small changes (Shani *et al.*, 2005a) which uses an online fixing of the policy produced by the Perseus algorithm. These are good algorithms which address topics similar to ours, yet they cannot be applied in every case (we might not know perfectly the observation probabilities of the different kind of states, the changes in the environment might not be small). Furthermore their approach also lack theoretical guarantees, although it shows good experimental performance on the classical POMDP domain (Cassandra, 2004).

Predictive State Representations The other main approach is to use a slightly different framework to represent the problem, which is Predictive State Representations (PSRs) (McCallum, 1996; Shani and Brafman, 2004; Singh *et al.*, 2003). It is a model-free experience-based framework, which allows to learn how the environment behaves without finding out its parameter. It is a very reasonable approach, since series of actions and observations can be produced by *many* different underlying models; this set of methods find the underlying dynamics of the system, without explicitly keeping track of an underlying state. Instead it keeps track of the probabilities of core tests succeeding.

Unfortunately, these approaches need huge amounts of training samples (even for

simple problems), over a very stable model of the environment. And what they learn cannot be easily re-used, as it is difficult to understand what the resulting parameters mean in the real world. The learned parameters are evolutions in the probabilities of core tests and not state transitions. PSRs does not allow easily to inject prior knowledge about the world, and we may possess such knowledge.

Limitations of existing learning methods All existing methods have one of the following limitations. They are either not guaranteed to converge and learn the exact model at every run (like EM methods), or they need a huge amount of queries (like PSR methods). They might need some prior information about the model (like USM methods). Furthermore it might be difficult to incorporate prior information about the model in them or to understand what they learn (like with PSR methods). We would like to build a learning method that does not have these limitations.

2.2 Active Learning

Active Learning focuses on finding out what is the best learning action to take given a prior knowledge. In an active learning setup, we consider how best we could reduce our uncertainty in order to accomplish several tasks. What we would like to do is to use the active learning techniques to act optimally in a partially-observed Markovian environment (a POMDP) when the parameters of the environment are uncertain. Active Learning should be able to tell us how to learn optimally.

Efficient *active learning* techniques have been proposed by Cohn (Cohn *et al.*, 1996), for classification tasks in which some examples don't have a label. The agent has the possibility of obtaining the label for some of them, but this is costly. Solutions that are usually proposed for these problems involve evaluating the information gain that obtaining a label would bring, which actually is the variance the learner has con-

sidering this label. As Anderson and Moore show (Anderson and Moore, 2005), these ideas can be extended to dynamical models like Hidden Markov Models (HMMs).

A Hidden Markov can be seen as a POMDP in which there is only one action. In a HMM we usually try to find what is the underlying state at each time step by considering the whole sequence. The Active Learning techniques consider queries that *reveal* the hidden state at a given time step. The method Anderson and Moore proposes is able to find the query answer that would reduce the variance of the learner the most.

Our approach is to adapt these kind of learning techniques to POMDPs. However, the problem is slightly different since in a POMDP what we're interested in is to find the optimal policy and not to determine the sequence of underlying states. Furthermore, determining the exact value of the underlying parameters may be useless, since some of them may have no influence over the optimal policy. Furthermore, we have a control over the sequence of actions we execute, so we have an influence over the states that are visited. This makes the problem far more complicated, since we have two decisions to make: the action we take and the time step at which we should identify the underlying state. So unfortunately, Anderson and Moore's method (Anderson and Moore, 2005) cannot be applied directly.

Our goal will therefore be to use an active learning setting that is able to find the optimal policy by selecting sequences of actions and selected identifications of the hidden state.

2.3 Dirichlet distributions and Bayesian Learning

Our work also inspires from the methods called Bayesian Learning (Dearden *et al.*, 1999; Strens, 2000), which use Dirichlet distributions to model the uncertainty over

a model and use them to have an efficient exploration/exploitation tradeof. Their method is used for completely observed Markovian environments (MDPs) but we believed that they could be applied to POMDPs.

Dirichlet distributions are distributions *over multinomial distributions.* Since, in a POMDP,the transition and observation probabilities are specified according to multinomial distributions, we need to use Dirichlet distributions to model uncertainty over them.

Definition Let us consider a multinomial distribution over N values, of parameters $(\theta_1, \ldots \theta_N)$. A Dirichlet distribution is a distribution over these parameters θ. It is defined by the hyper-parameters $(\alpha_1, \ldots \alpha_N)$. With this setting, the likelihood of a multinomial distribution given the Dirichlet distribution is equal to:

$$p(\theta_1 \ldots \theta_N | D) = \frac{\prod_{i=1}^{N} \theta_i^{\alpha_i - 1}}{Z(D)}, \text{where} Z(D) = \frac{\prod_{i=1}^{N} \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^{N} \alpha_i)} \tag{2.7}$$

We precise that the most likely multinomial distribution has the following parameters: $\theta_1^* \ldots \theta_N^*$, where

$$\forall i = 1, \ldots N, \theta_i^* = \frac{\alpha_i}{\sum_{k=1}^{N} \alpha_k} \tag{2.8}$$

Updating Dirichlet distributions Dirichlet distributions can be used to model the evolution of our uncertainty as learning progresses. Let us suppose that we are trying to estimate the parameters of an unknown multinomial distribution. Then, the following setting can be used to set the evolution of the Dirichlet as we accumulate experience. Each time i is sampled from our distribution, we increment hyper-parameter α_i by an amount of $\lambda \in \mathbf{R}_+^*$. With this setting, the most likely distribution will be

equal to:

$$\begin{aligned} \forall i = 1, \dots N \ \theta_i^* &= \frac{C_i\lambda + \alpha 0_i}{\sum_{k=1}^{N} \alpha 0_k + \lambda \sum_{k=1} NC_k} \\ &= \frac{C_i + \frac{\alpha 0_i}{\lambda}}{\sum_{k=1}^{N} C_k + \frac{\sum \alpha 0_k}{\lambda}} \\ &= \frac{C_i + C0_i}{\sum_{k=1}^{N} C_k + C0_k} \\ &\quad \text{,where } \forall k \in \{1 \dots N\}, C0_k = \frac{\alpha 0_k}{\lambda} \end{aligned} \tag{2.9}$$

Here, C_i represents the number of times we have experienced the event i. The $\alpha 0_i$ represents the values at which the hyper-parameters of our distribution are initialized. We can see $C0_i$ as "fake" experience counts which depends of the initialization of the parameters. The way we initialize our parameters depend on the *prior* we have over the value of the parameters of our multinomial distribution. For example, if according to our prior the distribution is uniform, we initialize all the α_i to the same value.

Then the above equation can be seen as having the following meaning: the most likely distribution has for each of its parameters a value equal to the number of time the value was obtained (counting the fake counts of the prior) divided by the total number of overall samples (counting the fake counts of the prior).

Variance in Dirichlet distributions It can be shown that the expression of the variance on one of the parameters in the Dirichlet has the following expression.

$$\forall i = 1, \dots N \ \text{Var}(\theta_i) = \frac{\theta_i^*(1 - \theta_i^*)}{C0_k + C_k} \tag{2.10}$$

So if we call $C0_k + C_k$ the overall **confidence** we can say the following: for a given value of the estimation, the variance is *inversely proportional* to the overall

confidence. We also see that according to this the variance is always higher when we are near 0.5 than when we are near 0 or 1.

Sampling from Dirichlet distributions We can sample from a Dirichlet distribution by proceeding as follows: we consider the Gamma distribution of parameters $\alpha_1 \dots \alpha_N$ and sample N values $x_1 \dots x_N$ from them[5]. Then the multinomial distribution of parameters $\forall i\ \theta_i = \frac{x_i}{\sum_{k=1}^{N} x_k}$ is a sample from our Dirichlet distribution.

Bayesian Learning Dirichlet distributions are used in Bayesian Learning methods of Markov Decision Processes (Dearden *et al.*, 1999; Strens, 2000). In these methods, models are sampled from the Dirichlet distributions and the agent follows policies that are computed according to the sampled models. The parameters of the Dirichlet are updated as the experiment goes. The event $[s, a] \rightarrow s'$ increases the hyper-parameter corresponding to $P^a_{s,s'}$ by a fixed amount. They show that their method allows a good tradeof between exploration and exploitation and that they can learn optimal policies for MDP models quickly while obtaining a reasonable reward through the learning phase.

These methods are convenient in MDPs, because we know the underlying state at every time step, so we know which parameter in the Dirichlet distributions to update. However, in a POMDP, we don't know the underlying state, so updating these parameters can be hard. However we will use the idea and try to adapt it. Since we will use an active learning and allow the hidden state to be perfectly identified by an oracle, we will know in some cases which is the right parameter to update and therefore we will manage to adapt these methods.

[5]To sample from Gamma distribution for our algorithm, we used a code written by Radford M. Neal. which uses the method described in (Devroye, 1986).

2.4 Conclusion

We have presented Partially Observable Markov Decision Processes, which can efficiently model problems of decision-making in partially observable sequential settings. We have presented the main methods one can use to solve them. We will not use heuristic methods, since they are too unprecise and fail in many cases, and we will not use policy search methods, since they need prior information about the optimal policy. We will focus on approximate value-based methods.

However, the setting we presented in Chapter 1 supposes that we are in a partially observable environment that is partially known. To address this problem, we reviewed three approaches.

POMDP learning methods POMDP learning methods usually don't have a way of making optimal decisions, since they rely mostly on learning samples in which random actions are performed. Furthermore, we would like to have a method that uses only a reduced number of experimentation, that is guaranteed to learn the true model, that requires no particular prior knowledge, and that gives information easily understandable and reusable. We have shown that the main existing methods (EM, USM, and PSR) all fail for at least one of these conditions.

Active learning methods The active learning setup gives methods to find the best possible learning action. This would be interesting, however our problem is only to learn the parameters well enough so that the optimal policy is learned, and current Active Learning methods cannot help us for this particular. However Active Learning methods suggest that we should require in our learning the existence of an oracle, that can provide the identity of the Hidden State.

Bayesian learning methods Bayesian learning allows to represent explicitly the uncertainty in the model that is currently learned, and to incorporate past experimentation and prior information to the uncertainty model. Furthermore, these methods can give efficient methods to produce near-optimal policies for uncertain environments. Unfortunately, these methods can be applied only to Markov Decision Processes where the hidden state is perfectly known.

Chapter 3

Adapting Active Learning to POMDPs

We have explained the POMDP framework and different learning methods to learn these models. However the existing methods have many drawbacks: they did not provide an optimal way of acting during learning. Some of them were prone to local minima. Some of them required huge amounts of experimentation. Some of them required specific conditions about the prior information. In this chapter we want to introduce a method, based on active learning, that is theoretically guaranteed to find the optimal way to behave and learn, so that the learning phase is minimal and so only the most useful features are learned.

The idea of active learning is to explore the environment intelligently, so as to make the learning as quick as possible. Active learning also introduces the concept of active learning queries, that allows the environment to be learned in an efficient way. We present in this chapter a decision-theoretic approach to solve the problem of applying active learning to POMDP environments, and present experimental results of this method on a simple problem.

3.1 The active learning setting

POMDP models are very hard to learn because they have the hidden state, that sometimes can never be known with full certainty when one looks at the sequence of actions and observations.

Yet we do assume the existence of an underlying state. It means that, even though this state is at a given time only partially known, it is the indicator of something that happens in the world beyond the agent's perception.

Because there is something beyond the agent's perception, it means that someone can have access to it. If not the agent itself, maybe an external agent. This brings us to our main assumption. We will assume the existence of this external agent, that can identify the hidden state. It will be called the **oracle**.

This external agent will play the role of a teacher, and it will help our agent in its learning. First, we will assume that the teacher already has some knowledge (even limited) about the dynamics of the world, and will give it to the agent.

Then, and because we want the learning to be as fast and efficient as possible, we want our agent to act in the environment and we will give him, in order to learn more than what it immediately perceives, the possibility to ask queries to the external agent. This process, of experimenting certain actions under certain circumstances and asking to the external agent the resulting hidden state, is what we call active learning for POMDPs.

3.1.1 Our goal

We focus on learning the sets of parameters $\{P^a_{s,s'}\}$ and $\{O^a_{s,z}\}$, since they are characteristics of the real world that have a big impact on the optimal policy of our

problem.

But we have to state that it is not necessary to learn them all. The reason is that in some cases some of the parameters have no influence on the optimal policy. For example, there might be some states that can never be reached. The value of the parameters corresponding to them has therefore non influence at all on the value function. We call these parameters **useless parameters**. More formally, a set of parameters $\{P^a_{s,\star}\}$ is useless if and only if for all possible series of actions state s has a probability of zero of being obtained, and parameter $\{O^a_{s,\star}\}$ is useless if and only if for all possible series of actions in which the last action is a, state s has a probability of 0 of being obtained. It is useless to learn these parameters, and furthermore it might be impossible to learn them. So our active learning should not try to learn them at all.

There are other parameters that are not useful to be learned; these are the parameters that have *no influence on the value function.* More formally, the set of parameters $\{P^a_{s,\star}\}$ has no influence on the value function if and only if for all the possible values this parameters have, the value of the optimal policy for a given history is the same. We can easily prove that if a parameter is useless, it has no influence on the value function. Our algorithm will try to avoid learning these parameters. We precise that these parameters can, for example, correspond to actions that would not be taken under an optimal policy. It is common that in a POMDP problem some of the parameters may be completely irrelevant.

Furthermore, we have to remark that some parameters need to be learned more precisely than others. A small variation of one parameter could bring a huge difference in the policy, whereas some other parameter may take a large interval of possible values without changing the optimal policy. For instance, in the Tiger problem, the most important parameter is the probability of receiving the correct observation when the Listen action is performed. However, a clearly non-critical parameter would be the

transition probabilities when an incorrect door is opened. Since this event happens only rarely under an optimal policy, the associated parameters need not be estimated precisely, as its exact value may have only little influence over the resulting policy. So what we really want to do is to learn each useful parameter *with a precision that is proportional to its influence on the optimal policy.*

To help the agent learn we assume the existence of an external agent that can provide information during the learning phase. Ideally, an application of our desired setting would give the following procedure: during a first, **active learning phase**, our agent would be in partial autonomy, an external agent giving it information about the underlying state of the world. Once this phase is over, the agent could finish its learning by itself and acquire full autonomy.

Such a setup is desirable in many applications. For instance, in robotics, where we can afford to make someone spend some time to calibrate the robot before letting it act autonomously, such an algorithm could have good applications.

Our goal is to minimize the length of the active learning phase, and to minimize the number of requests that are asked, so that the most relevant parameters of the model are learned as quickly as possible.

3.1.2 Our assumptions

We will assume the existence of an **oracle**. This entity can, upon request, provide the identity of the underlying state.

In most tasks, it is possible, for a certain cost, to find the identity of the hidden state. Because we use the POMDP framework, and because we use a model-based approach, the states do have a signification in the real world. And we believe that in most cases a human (or another automated system) can identify it, either during the

experimentation or after. It should merely be consulting information that are beyond the agent's sensors.

Querying the oracle is costly since there is additional work needed to answer them so we should try to minimize their number. Actually, we explicitly associate the querying to an additional cost. We underline that querying the oracle has a role equivalent to doing the active learning queries in the classical active learning setting.

Assuming the existence of an oracle will allow a learning algorithm to avoid undesirable local minima, will guarantee the convergence of the algorithm to the true model, and will also accelerate the learning process greatly.

We are not learning the reward function. We assume that it is perfectly known. The reason for this is that the reward function is not necessarily part of the environment; it is a function we build so that our agent executes what we want. Furthermore, if we had indeed non-deterministic rewards, they could be incorporated into the set of observations, and we would fix our reward function so that each of these "reward observations" has according to the function R a reward that is associated to their value. This would have the advantage of automatically include the information brought by the reward in the belief updates, and the information we would need to know would only be the different set of values the reward can be. As a result, we assume in this thesis that the reward function R is perfectly known.

We are not learning b_0, since we assumed it is known. Note that our algorithm could easily be modified to learn b_0. We just made this assumption to simplify our implementation a little. Theoretically, b_0 could be learned as one of the parameters[1].

[1]Actually, the algorithm we present in Chapter 4 does regular "restarts" of the POMDP during learning. Therefore, querying after each restart brings information about the initial belief, so our algorithm can very easily be adapted. The solution proposed in Section 3.2, however makes no restart.

We will also assume that we have prior knowledge about the environment. At the very least, we will assume that the number of states, actions of observations is known. Whereas it is reasonable to assume that the number of actions and observations is known beforehand, it is a strong assumption that the number of states is known. This assumption is not entirely necessary in model-free methods (McCallum, 1996) or in Shani's method (Shani *et al.*, 2005b). But we believe that in most realistic POMDP applications we already have prior information about the states anyways. Our algorithm is specially made for the case where we have an agent in a partially observable stochastic environment in which we already have an idea of what each state means and what the values of the parameters can be. We mostly want to *correct* their estimation by interacting with the environment.

3.2 Decision-Theoretic planning

We will now study what would be the optimal way of solving the active learning problem. We propose the Decision-Theoretic Planning method, which builds a larger POMDP from the initial one and solves it in order to find the best way to combine queries and actions so that learning is optimal.

Therefore, we integrate into our initial POMDP that (1) we do not know exactly some of its parameters. (2) The agent can do active learning requests to obtain the hidden state of the POMDP. (3) These requests have an associated cost, and we want to minimize their number.

3.2.1 The algorithm

A theoretically optimal solution is to modify our initial POMDP model by doing the following modifications: (1) For each uncertain parameter, we add a feature to our

state space. It represents the value the uncertain parameter has. We set the initial belief such that the distribution of the state space over this feature is uniform (or equal to an initial prior if we have one). The transition parameters are set so that the transitions occur only between states that have the same values for each of the features. (2) Then, we add to the action set an action called "Query", which does not change the state but give as an observation the identity of the real state. (3) Finally, we set the reward function such that the "Query" action is always associated with a penalty. This gives a meta-POMDP that can be solved according to classical solving method (Kaebling *et al.*, 1998).

Unfortunately, this theoretically optimal setup gives a meta-POMDP with an infinite number of states. Since few methods can solve such POMDPs exactly, we need to consider approximations. We will therefore discretize the additional features into n levels. The number of states will be multiplied by n each time we have an uncertain parameter, so that we obtain n^k group of states, where k is the number of incertain paramers. Each of these groups will correspond to a different assignment for the values of the unknown parameters, and within a group the transitions and observations will be made according to these values.

Note that the method not only gives a theoretically optimal way of making the decisions, but also gives the most likely value of the parameters. One just has to check what the belief state is at the end of the experiment, to find what is the probability distribution among possible parameter assignments. This method is therefore theoretically very powerful, as it gives the perfect method between learning and exploiting.

3.2.2 Experimental results

We present an example of the application of this method to the Tiger POMDP problem (Kaebling *et al.*, 1998), presented in Section 2.1.1.

We use the following setting: we suppose that the parameter p (which is the probability of having a correct observation when the "Listen" action is performed) is unknown. We consider that its value is in the range $[0.7 \dots 0.9]$. We consider three hypotheses for the value of p: $p \in \{0.7, 0.8, 0.9\}$. All the other parameters are supposed to be certain.

We produce a POMDP according to the technique described in Section 3.2.1: $R_q < 0$ is the penalty associated to the query. The initial belief is uniform among the three possible values of p.

To solve the resulting POMDP, we used the *finite grid* method, from Cassandra (Cassandra, 2004), with a bounded number of grid points and a finite horizon (the grid points we consider are a representative set of the belief states that can be reached, and we always update the value for the same belief points). We used a number of grid points around 1000 and horizons around 30. This method is also used in the next chapters of the thesis, each time we compute an approximate policy for a POMDP.

Figure 3.1 summarizes the behaviors obtained for different values of R_q. There is mostly two kinds of policies: if R_q is low, the agent alternates between the query action and the door-opening action (and therefore it never learns the value of p). On the other hand, if R_q is high it never uses any query: however the return observed is still quite high: the agent manages to behave optimally even though it does not use queries to learn what the value of p is. There is only a small range of values of R_q where we have mixed policies, that use queries at first and then switch to a classical

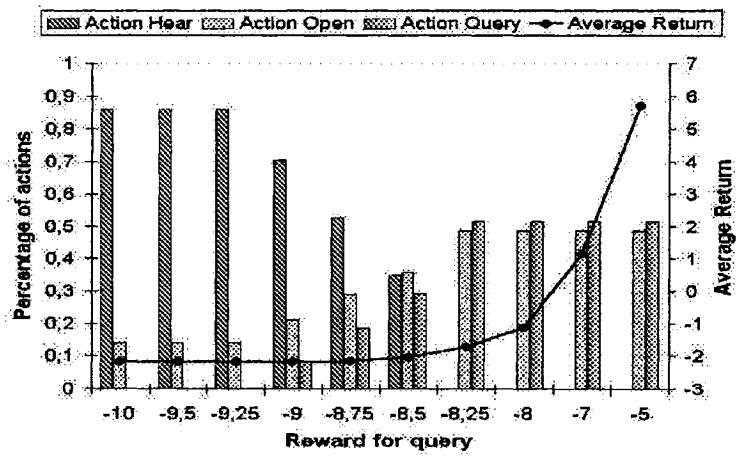


Figure 3.1: Experimental results with the "Tiger" problem, with the setting described in Section 3.2.1, for different values of R_q.

policy.

This suggests that non-query learning can indeed work when the setting is simple, and points out how it is delicate to set up the query penalty value when we use Decision-Theoretic planning.

We can see that we can get a good performance even if we never do any query. The queries become useful only if their cost is low enough: then they can slightly increase the resulting reward.

3.3 Conclusions

We have described our main ideas for doing a decision-theoretic active learning of POMDPs. We involve the use of an oracle, that can provide upon request the identity of the hidden state in the POMDP. We focus on learning the parameters that have the greatest influence on the learning policy

We have shown a theoretically optimal method to extend active learning to a

POMDP model. We have introduced the oracle assumption, and described exactly the learning problem we were trying to solve. We have presented a theoretically optimal method to solve it, Decision-Theoretic Planning, to solve it, and we have applied it successfully to a simple problem.

However, even if it is theoretically sound, this approach has many disadvantages. First, it can't be used for POMDPs with a large number of states. Since we multiply the number of states by n^k, the research of the optimal policy becomes quickly intractable (the complexity of the belief space increases by $\exp(n^k)$). Furthermore, even with small POMDPs we have the following problem: it is very hard to specify the penalty that we should associate to the query action. It needs to be high enough so that it is used only for learning and not as a part of a permanent policy, and it needs to be low enough so that queries can be used to do learning.

Chapter 4

The MEDUSA algorithm

In the previous chapter we have presented a setup for active learning in POMDPs, and a simple algorithm to solve it optimally. Unfortunately, this algorithm fails to scale to complex environments, and we have to find a different approach. The algorithm proposed in this chapter, MEDUSA, is more robust, and can scale to larger problems.

MEDUSA uses the Bayesian Learning techniques (Dearden *et al.*, 1999; Strens, 2000), in order to combine a partial model of the environment with direct experimentation, so that we execute policies that can resist to model uncertainty. Bayesian Learning could not be applied to POMDPs because in order to update the correct parameters we needed to know the underlying state. But since we make the oracle assumption, there is a way to know the underlying state with full certainty, and so we can adapt Bayesian Learning methods to POMDPs.

With the Bayesian Learning setting, our agent is able to handle uncertainty explicitly, as a part of the model and to make decisions that take it into account. We allow the agent to learn and to correct its model as it gather experience through the environment and with the help of the oracle.

4.1 Main features

Our algorithm, called MEDUSA for "Markovian Exploration with Decision based on the Use of Sampled Models Algorithm" is an approach based upon the following facts:

- It is not necessary, if we want to find the optimal policy of a POMDP, to learn perfectly all its parameters. To evaluate if the parameters we have learned are good enough, we can estimate the variance over the value function taking our prior information and our experience into account.
- Each query brings information about the transition and the observation which just happened. This information, used with the prior information we have, allow us to update the information we have over the real model. The more we experience some transitions, the more our estimation of the corresponding parameters are precise.
- We want to minimize the amount of data needed for learning and the number of queries. Since a "random" exploration of the environment would not be optimal to evaluate the important parameters, it is necessary to have an approach that can explore the environment in an intelligent manner, in order to provide a more precise evaluation of the important parameters. We will therefore do queries on runs where what we believe can be a reasonable policy is executed.
- It is necessary to consider a pool of models instead of just one that would be evolving because of the following facts: different sampled models are *hypotheses* about the world, that can have very different resulting policies. The actions we try are actions that are likely to be optimal considering our current model of the uncertainty. The only actions we never try are actions that could never be optimal, for any value of the parameters. If we considered only one model, we would always follow the policy corresponding to the most likely hypothesis: but it might always be the wrong policy and we might never try actions that

could bring more knowledge about our environment; some of the optimal actions would never be taken in some cases. So, as one model would mean only exploitation, considering several models bring something that does a very clever tradeof between exploration and exploitation.

The setting of the algorithm is the following. First, we build a set of Dirichlets that gathers the prior information we have on our POMDP model. For each uncertain transition distribution and for each uncertain observation distribution we build a Dirichlet distribution, whose initial parameters depend on our prior over the model, as explained in 4.2. We can also use a smaller number of hyper-parameters.

We maintain a pool of POMDP models which are drawn according to the Dirichlet distributions. We compute the optimal policy π_i for each of these. Each of the models maintains its own belief state b_i. At each time step, we choose an action according to the methods described in Section 4.3.

Each time our agent acts, it receives an observation. It can then decide to make a query in order to identify the hidden state of the system. Whether or not the query is made, the parameters of the corresponding Dirichlet distributions are updated.

At regular intervals, we improve our pool of models by resampling POMDPs according to the current prior and eliminating the most unlikely models. Each time a model is sampled, we compute its optimal policy. Since it is used to improve the choice of actions, the quality of the overall policy increases. Therefore, the learning takes place mostly in regions of the state space visited by policies that are optimal in at least one of the likely environments.

We precise that at regular intervals we restart the problem. This is needed if we want to have the theoretical guarantees of convergence (see 5.2).

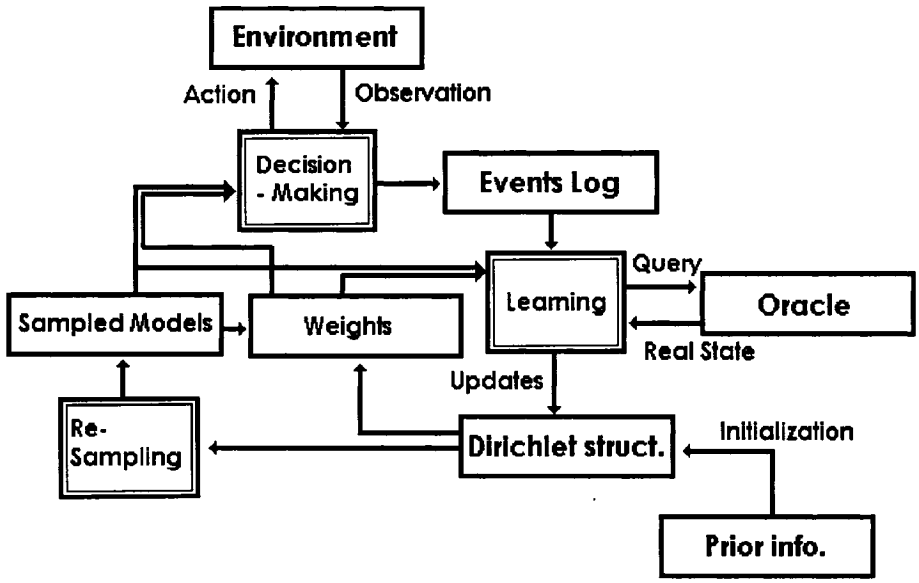


Figure 4.1: This chart summarizes the MEDUSA algorithm.

The script is detailed in Table 4.1. Figure 4.1 is a chart that illustrates the mechanics of the algorithm.

4.2 The integration of prior information

An important asset in our approach is that it is easy to use prior information about the model if we have some. This takes place at the beginning of our algorithm, when we build the Dirichlet structure. First, we needn't use one Dirichlet distribution for each uncertain distribution. We may use a restricted number of hyper-parameters. Secondly, we can set the initial values of the hyper-parameters as to incorporate our prior knowledge.

Certain parameters If we consider that in our model some parameters are certain, we can sample models that always have the corresponding values for this parameter.

For example, if an agent is navigating through a labyrinth, we can consider that we know that the action "do not move" does not change the hidden state. We can also incorporate the idea that one action cannot move the agent to more than two cells of distance.

Therefore, in our experiments, we usually considered that parameters having the values 0 or 1 in the true model are known with full certainty. Each time we sample a model from our Dirichlet system, we set the corresponding parameters to their certain value. When we would update the hyper-parameter corresponding to it we just do nothing.

Related parameters There are other ways to diminish the number of hyper-parameters. For instance, the observation distributions may be identical for the same kinds of states. Or, in some cases the observation may be independent of the last executed action, and we might know these facts beforehand. If it is the case, we just use the same Dirichlet distribution D for all the similar distributions. That is, when we sample a model, we sample a multinomial distribution from D, and we set all the parameters of the POMDP that corresponds to this distribution to the sampled values. Furthermore, when we would update the hyper-parameter corresponding to it we just update the right hyper-parameter of the Dirichlet distribution.

If we take all this into account, we actually obtain a *Dirichlet structure*, which consists of a set of Dirichlet distributions, represented by their hyper-parameters, and an array of pointers, that give, for each parameter in the POMDP:

- Either its value, if it is certain.
- Or a pointer to one of the parameters corresponding to a Dirichlet in the Dirichlet structure.

We explain in the experimental section examples of Dirichlet structures on some POMDPs. The ability to diminish the number of alpha-parameters usually make the learning problems more tractable.

Priors on the parameters When we have prior information about the estimate of some of the parameters, it is very easy to use it when we set the initial values of the hyper-parameters in the Dirichlet distributions (see Section 2.3). Let's say that given our prior information we suppose that the value of parameter p is $\tilde{p}$, and that the confidence we have over this value and all the values corresponding to the same distribution is $C_{\tilde{p}}$. By confidence, we mean a number that is inversely proportional to the variance we have in our estimation (since both are linked according to Equation 2.10.

If it is the case, we initialize the parameters with the following setting: $\alpha_0 \leftarrow \tilde{p} * C_p$. In our experiments we used a confidence of 1 and uniform distributions.

Note that since we used a model-based approach, the models that we have learned for one task can easily be used as priors for other tasks. This is a very useful asset for robotics applications.

4.3 Action selection during learning in MEDUSA

In this Section we describe the different Decision-making functions we can use in MEDUSA. Even though with queries at every step we would converge to the real model if we did random actions, we might want to accumulate reasonable reward through the process, and we might want to have a minimum number of time steps and queries. We present three methods that we tested.

4.3.1 The exploratory decision-making

Each of the sampled models in the representative pool can be seen as an expert. The agent maintains a certain confidence in each of the experts, which depends on its knowledge of the world. The experts have different views of the world, and always recommend the action that is optimal according to their view of the world. One of them might be right and recommend the action that is really best but the agent doesn't know which one. So if we want to have a chance of trying the really best action, the best way to do it is to choose randomly an expert (according to the confidence the agent has in it). If we had only one expert which would have the most likely vision of the world, he might always recommend a wrong action, as *most likely* is different from *true*. We might end up always doing a wrong action and never learning anything.

The exploratory decision-making relies on a simple assumption. If an action can be good, it needs to be tried, only then can we know if it was good or bad. More formally, an action that is optimal for a certain setting of the uncertain parameters should have a non-zero probability of being taken. Furthermore, the probability should be proportional to the probability this action has of being optimal, according to our current uncertainty model.

As we show in the theoretical Section 5.1.2, we can approximate this formal setting by using the following stochastic policy:

In it, $a_i = \pi_i(b_i)$ is chosen with probability w_i, where

$$\forall i \; w_i = \frac{p_i}{p0_i} \frac{1}{\sum_{k=1}^{n} \frac{p_k}{p_{0k}}} \tag{4.1}$$

Here p_i is the *current* likelihood of model i according to the set of Dirichlets and $p0_i$ the initial likelihood this model had when it was sampled.

4.3.2 The safe decision-making

Note that some of the experts can be completely wrong and have very bad visions of the world and would recommend disastrous actions. So if we want to have a risk-free behavior in our environment, at the cost of less exploration, the best is to listen to the advice of every expert for every actions, weight the experts' advice by the confidence the agent has in them and take the action that is the best in the average (though maybe not a single expert recommends it). This is also still better than following the policy corresponding to the most likely expert. The problem is that the expert corresponding to the most likely model does not have any idea about what would happen if his view of the world was different, even slightly. He does not know that the action he recommends, which seems optimal to him, could be disastrous with slightly different parameters of the world.

Whenever we are not in a learning phase (for instance, if at the moment queries can't be processed or if we have finished our learning phase and begun our exploitation phase), we should use this safer policy. The process is the following: each POMDP, once solved, can give the value associated to each action for the current history. Let $V : A \times M \times H \rightarrow R$ be the function that returns, for an history $h \in H$, a model $m \in M$, an action $a \in A$, the optimal associated value. Let M_t be the current population of models and w_m the weight defined by Equation 4.1

According to safe decision-making, the chosen action is $a^\star$ such that:

$$a^\star = \arg\max[\sum_{m \in M_t} w_m V(a, m, h)] \tag{4.2}$$

This decision making procedure has a performance always higher to the exploratory policy. However, if we consider long term gain and learning, we have to say that this procedure will never take a "risky" action to do exploration. That's why this procedure can only be used when we exploit our learned model and not while learning.

Therefore, in each of our experiments, when we stop the learning process to evaluate the quality of the learned model, we evaluate the return by using this safe policy.

4.3.3 The Boltzman decision-making

We have also implemented a Boltzman decision-making, because it is a classical way of solving the exploration/exploitation tradeoff. Here, we determine the action taken stochastically in the following way.

$$p(A = a) = \exp(\frac{\sum_{m \in M} w_m V(a,m,h)}{T}) \tag{4.3}$$

T is the temperature factor, which decreases with time and the w_i are defined according to Equation 4.1. T decreases by a fixed amount at each step and when it reaches 0 the safe decision making is used.

Experimental results show that the performance of the Boltzman decision-making is not good, especially because at the beginning of the experiment the agent almost always does a random action, which can get the agent to be trapped in some bad states of the POMDP: under this setting, actions that are sub-optimal in every possible parameter setting can still be chosen, and taking these actions might be disastrous. Furthermore, when the temperature becomes lower, the agent becomes too concerned about exploitation to do any exploration. That's why the final version version of MEDUSA doesn't use it.

4.4 Learning in MEDUSA

We now describe how the learning is made in MEDUSA. We consider two types of learning. The first has a convergence to the true model guaranteed, as shown in 5.2 and uses the oracle queries. The second has no convergence guarantees but does not require queries. In practice, the second approach converges to a local minimum of the space of POMDP models.

In order to easily make the learning updates we use for each model an **alternate belief state** β_i. Its purpose is to keep track of how we can use the latest query to get an idea of what the current underlying state is.

The alternate belief of model i β_i is defined by the following rules.

1. At each query, when the result is s_q, $\beta_i(s) = \delta(s, s_q)$, where $\delta(i, j)$ is equal to 0 when $i \neq j$ and to 1 when $i = j$

2. When an action is made and an observation received, a Bayesian update (see Equation 2.1) is performed on β_i, according to the parameters of model i.

This alternate belief state allows us to follow, for each model, the information still available since the last query. The consequence is that if some actions or observations are deterministic, then the alternate belief state may stay certain during several steps after a query[1].

[1]For instance, in the Tiger problem, if a query is made at a given time step, the alternate belief state will remain certain as long as Listen actions are performed.

4.4.1 Query learning

Equations 4.4 and 4.5) show the updates executed when we do a query-learning update.

$$\forall s\, \alpha(s,a,s') \leftarrow \alpha(s,a,s') + \sum_i w_i \beta_i(s) \lambda \tag{4.4}$$

$$\alpha(s',a,z) \leftarrow \alpha(s',a,z) + \lambda \tag{4.5}$$

By $\alpha(s,a,s')$ we mean the hyper-parameter that corresponds to the transition parameter $P^a_{s,s'}$ when the parameter is uncertain (if it is certain then no update is made). By $\alpha(s',a,z)$ we mean the hyper-parameter corresponding to the observation parameter $O^a_{s,s'}$. When this setting is used only one observation hyper-parameter is updated. The alternate belief state β is used to update the transition parameters. Note that if a request had been done at the previous time step (or if the alternate belief state is at 1, only one transition parameter is updated.

If our algorithm uses a query at every step it can converge to the true model and to the optimal policy (as shown in Section 5.2.1. However we have studied how it was possible to learn without query. Actually the sole *action-observation* information can be used to update the parameters.

4.4.2 Non-Query learning

If we can't or don't want to use queries, or when the query result is invalid or if the query fails, we can make a non-query learning update. For this update, we use the *alternate transition belief* $B_t(s,s')$ which is defined according to Equation 4.6: it is the distribution over the transitions that could have occurred during the last step. The non-query learning updates the transition hyper-parameters according to

Equation 4.7. For the observation hyper-parameters, the update is done using the mean alternate belief state $\tilde{\beta}'$ which is defined according to Equation 4.8: 4.9.

$$\forall s, s' \; B_t(s, s') = \sum_{i=1}^{n} w_i \frac{[O_i]^z_{s',A} \, [P_i]^{s'}_{s,A} \, \beta_i(s)}{\sum_{\sigma \in S} [O_i]^z_{\sigma,A} \, [P_i]^{\sigma}_{s,A}} \tag{4.6}$$

$$\forall i \in [1 \ldots n] \; \forall (s, s') \;\; \alpha_t(s, A, s') \leftarrow \alpha_t(s, A, s') + \lambda \, B_t(s, s') \tag{4.7}$$

$$\forall s \; \tilde{\beta}'(s) = \sum_{i=1}^{n} w_i \beta'_i(s) \tag{4.8}$$

$$\forall i \in [1 \ldots n] \forall s' \in S \;\; \alpha_z(s', A, Z) \leftarrow \alpha_z(s', A, Z) + \lambda \, \beta'_i(s') w_i \tag{4.9}$$

Another method is to sample a query result according to the alternate belief and use a query learning update as if the sampled query result was a real query result. But our experimental results show that this offers no better performance than the above setting.

The properties of the non-query learning are the following. Contrary to query learning, we lose the guarantee of convergence towards the real model. However, experimental results show that we always do observe convergence, but we converge to local minima. It is logical, since that in most cases the action/observation sequence we observe can be explained by *several* models (they don't produce the same policies and rewards, however). The problems encountered here are the same that EM methods have to deal with. We converge to a model which is not necessarily the true one.

We also observe the following facts: the variance of the learning increases a lot. In order to have a convergence we need to use a lower learning rate (on the order of 100 times lower). We also observe that if the initial prior is good enough at the start, we obtain the convergence to the true model.

4.4.3 The heuristics

We present here heuristic methods we tested to decide whether or not a query should be made. Not all of them were selected, but all of them were studied. In all these equations, the w_i are the weights defined in 4.3.1.

Do N queries then stop This heuristic is simple: it just recommends doing queries at every step for N step and then not do queries anymore. Since a near-optimal policy is executed, the traces of experience through information should always bring good information and the quality of query learning is superior to the quality of non-query learning in all cases. So, this might not be a bad heuristic to use.

Decreasing frequency of queries p(Query) $= min(1, \rho\frac{1}{N})$

This approach is purely stochastic and therefore should bring no bias in the convergence. The idea here is simply to decrease the probability with the number of time steps N, so as to progressively replace query learning by non-query learning.

Variance on the belief state `BeliefVariance`$= \sum_{k=1}^{n} w_k \sum_{i \in S}(b_k(i) - \hat{b}(i))^2$, where $\forall i, \hat{b}(i) = \sum_{k=1}^{n} w_k b_k(i)$

This heuristic measures the variance on the belief state considering every model. It is high when the belief state uncertainty over models is high and it can be an indicator of the learning that still needs to be made.

PolicyEntropy `Entropy`$= -\sum_{a=1}^{|A|} p(\Psi, a) \ln(p(\Psi, a))$

This heuristic is the entropy of the resulting policy. The higher this is, the more stochastic our decision making is. Queries should be made as long as the models

recommend different actions, so as long as this heuristic is greater than 0.

Variance on the value function `Variance`= $\sum_i w_i(V(m_i, \Pi(h, m_i)) - \hat{V})^2$

$V(m_i, \Pi(h, m_i))$ is the value corresponding to the optimal action in model m_i. $\hat{V}$ is the mean value, $\hat{V} = \sum_i w_i V(m_i, \Pi(h, m_i)$.

This heuristic evaluates the uncertainty we currently have over the expected return at the current time. The intuition is that when this becomes 0 no more learning need to be done since the policy is optimal. This is one of the main heuristics we used. We show in Section 6.1.1 its evolution during an experiment and explain why it is useful.

Gain of information `InfoGain`= $\sum_{k=1}^{n}[w_k \sum_{i,j \in S^2}[B_t(i,j)(\frac{1}{\sum_{k' \in S} \alpha^t_{A,i,k'}} + \frac{1}{\sum_{k' \in Z} \alpha^z_{A,j,k'}})]]$
This heuristic evaluates how much information the answer to a query would expect to give. B_t is defined according to Equation 4.6. Furthermore, we will 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 corresponding transition/observation parameters are certain.

The "information" a query brings is, in this setting, inversely proportional to the confidence we have over the parameters that the query would help to refine. So it can also be seen as an indicator of the variance we have for the parameters that would be updated. Note that this heuristic is close to zero when a request would not bring new information, either because of a very high confidence in the parameters or because the parameters are certain (and in this case it is equal to 0). The higher this is, the more uncertain the parameters that could be updated by the query are.

Alternate belief state entropy `AltEntropy`= $\sum_{s \in S} -[\sum_{i=1}^{N} \beta_i(s)] \log(\sum_{i=1}^{N} \beta_i(s))$

This heuristic indicates how imprecise a non-query update would be. It uses the current alternate belief, which is an indicator of how much information can still be used from the last query result. This heuristic allows to extract as much information as possible from each query. A very interesting feature with this heuristic is that if the transition that just occurred was certain, and we had a full certainty in the alternate belief state, the full certainty remains and the value of the heuristic is zero. So when a sequence of deterministic transitions occur, only one query needs to be done.

4.4.4 The query decision

As we show experimentally in Section 6.1.1, the most useful heuristics are AltEntropy, Variance and InfoGain. Actually, we have found out that the most efficient query decision used the following setting:

$$\text{doQ} = (\text{AltEntropy} > \epsilon_1)(\text{InfoGain} > \epsilon_2)(\text{Variance} > \epsilon_3)||(\text{NReq} < \text{Nmin})) \quad (4.10)$$

The first condition ensures that a request can be made only if the state is sufficiently known because of a previous query result. The second condition ensures that a request is not made if we already know the parameters it corresponds too well enough. The third condition ensures that we will stop querying when the learning has stopped improving the value function. We also incorporate the number of queries to the decision in the third condition because in cases when we have nearly no prior information, all the drawn models are at the beginning uniformly bad and so the value of the Variance heuristic can be 0 for all of them. As we explain in Table 4.1, when we don't make queries the following behaviors are executed:

When the gain of information is negligible When `InfoGain`$< \epsilon_2$, no update is made. Actually if we are in a subpart of the model where the parameters are certain, there would be no update to make anyways. And if we are in a subpart of the model where the confidence we have in the parameters is already high, it is probably because we used a good number of requests already. Since the quality of non-query learning is lower than the quality of query-learning, we prefer to stop doing any learning instead of using non-query learning, since the only thing non-query would do would be risking to lower the quality of our estimation.

When the confidence in a parameter becomes above a threshold ($\frac{1}{\epsilon_2}$), we consider it as certain and stop trying to estimate it more precisely. We used $\epsilon_2 = 10^{-5}$, since we believe that a confidence of more than 10^5 in a parameter means that the parameter is certain. Actually, if the confidence is 10^5, if the value of the parameter is 0.5, the probability of the parameter having a difference of 1% with our estimate is approximately 0.01%. In a realistic problem this would be a very good precision rate, but if one believes that the precision should be higher one can lower ϵ_2.

When queries can be spared When `AltEntropy`$< \epsilon_1$, then doing a non-query update has nearly the same effect as doing a query update. (since we can determine with high certainty what the result of the query would be). In this case, the non-query update uses the same learning rate. We used $\epsilon_1 = 10^{-2}$: when a transition begins to be estimated as being more than 99.9% certain, we consider it certain. We acknowledge that we did not test this setting on POMDPs which had transitions in which some transitions occurred with less than 0.1% probability. However, if one believe this can be the case, one can simply lower ϵ_2. This heuristics works really well to economize queries in cases where some transitions are certain at the beginning.

When queries are not useful anymore When (Variance $> \epsilon_3$)||(NReq $<$ Nmin), the estimation of the value function is good enough, so we can begin to do a non-

query learning of the parameters. However, in the current setting we use a lower learning rate when this is the case. (we divide the normal learning rate by 100). This allows to compensate the great variance introduced by non-query learning. When the condition is fulfilled, having more precision on the value of the parameters do not influence the return much. Note that since it can be useful to still have finer values for the parameters (we might want to use the model we built later on for other tasks in the same environment), we still use non-query learning, so that we can learn a model that brings the optimal reward and that is coherent with the sequence of actions and observations. The value of ϵ_3 depends on the problem we are solving, and especially on the order of magnitude of the reward function, and on the quality we're willing to get by the use of queries. We mostly used values around 0.5 for the threshold of this heuristic.

4.4.5 The learning phases

A typical execution of the algorithm goes through the following phases: in a first phase, we use a learning of good quality, with a high learning rate. Queries are made at every step, except when we can avoid making them because of certainties in the model. The first phase ends when the variance of the value function becomes low enough. Then we go through a second-phase which is a learning with full autonomy, in which the parameters are still improved. When we believe our parameters are certain, we can then switch to a third phase in which no learning is made anymore and we can just use an optimal policy for our learned model. With such an algorithm, we can sometimes converge to a model that is not the true one, but that (1) is coherent with the sequence of actions/observations experienced and (2) has an optimal policy that is the same than the true optimal policy. If we really wanted to have a guaranteed exact model, we would need to do queries at every step.

4.5 Non-stationarity adaptation

We want our algorithm to adapt to non-stationary environments. In non-stationary environments, the parameters of our POMDP model may be varying in time. This is a setting to consider if we want to apply POMDP framework techniques to real-world robotics, since the precisions of sensors and the motion behavior of robots may very well vary with time. Very few algorithms in the literature are able to cope with a non-stationary partially-observable setting. But in our case, adapting MEDUSA so that it can manage non-stationary environment can be simple: we need to give more weight to recent experience in the Dirichlet structure. For this, we proceed to the following: each time a hyper-parameter α is increased by an amount of $x\lambda$, we decay all the hyper-parameters corresponding to the updated multinomial distribution by ν^x, where $\nu \in [0; 1]$ is the *decaying factor*; this decreases the confidence of the model but do not change the value of the most likely parameters[2].

$$\forall i = 1, \dots N, \theta^*_{i,new} = \frac{\nu^x \alpha_i}{\sum_{k=1}^{N} \nu^x \alpha_k} = \frac{\alpha_i}{\sum_{k=1}^{N} \alpha_k} = \theta^*_{i,old}$$

The consequence of these update rules is that the older an update is, the least its influence over the current uncertainty model is. We note that the equilibrium confidence C_eq, according to the way updates are made, has the following expression: $C_{eq} = \lambda \frac{1}{1-\nu}$. This confidence is obtained after an infinite number of time steps. It is an indicator of the confidence we globally have in past experience. The higher this value is, the more stable we expect our environment to be, since the confidence is inversely proportional to the variance we have.

[2]When a query is made x is equal to 1. In this case we multiply all the parameters in the modified distribution by ν, which (as shown) only modifies the confidence associated to the updated distribution. When a non-query update is made, x can be smaller than one. The setting will then guarantee that when the non-stationarity adjustment is made the most likely value in all the modified distributions does not change, but their confidence is decreased in an amount that is proportional to the amount of the update.

Note that whenever we *know* that our environment might have experienced a change we should decay all the parameters of the Dirichlet structure by some amount. We might know that our environment have experienced a change if we changed some parts of our robot in a robotics application; we might also expect our environment to change each time we turn off and on our system.

Note that when the environment is non-stationary, we should always set the threshold ϵ_2 of Equation 4.10 to 0, since an uncertain parameter is *never* completely learned. Non query learning should always be accomplished to keep track of the parameters.

Our experiments show that if the parameters vary slowly enough, it is indeed possible to follow the evolution of the model by using non-query learning. Our experiments also show that if there is a big change in the environment, the system can detect it and ask for queries until the parameters converge to their new values (see Section 6.1.5).

We specify that in a non-stationary environment in which we believe changes are small, the safe decision-making should be preferred.

4.6 Imperfect queries

We now investigate how MEDUSA can handle imperfect queries, since the oracle may make mistakes while identifying the hidden state. If we believe it might be the case, we need to consider the following modifications.

- Query results need to be filtered. If none of the models can explain the query result (if the belief of the returned state is 0 in every model), we proceed as if the request had not been answered.

- If a request remains without answer, we do a non-query update with a lower learning rate instead.
- Non-query learning updates should use the belief state instead of the alternate belief state in their process. Since query results may be wrong we do not want to use them more than once.
- The threshold ϵ_1 (see Equation 4.10should be set to a negative value (the Alter-Entropy heuristic should not be used at all). This a consequence of the previous point.
- The learning rate should be lower, since the variance in the learning is higher.
- Noise in the query answer introduces bias when query learning is made. We can make this bias slowly disappear if we consider that the model is non-stationary. Therefore, we should also use the non-stationary setting described in Section 4.5, even if we believe our model is stationary. Actually, the noisy query answers can be seen as an additional non-stationarity in the model.

Note that when the query is noisy we lose all theoretical guarantees of convergence, and the higher the noise is, the higher the variance of our learning becomes. We show experimental results with noisy queries in 6.4

4.7 Technical issues

In this section we discuss some implementation details that are pertinent to MEDUSA.

4.7.1 Solving the models

The method we used for the computation of the policies is the **finite grid** method, implemented by Cassandra (Cassandra, 2004): it is a grid-based method (see Section 2.1.2) that finds an approximate solution for a finite horizon by using a grid of a finite number of belief points. This method is able to find an approximate solution fast. It needs two parameters, which are the maximum number of grid points on which the value updates will be performed and the planning horizon. The higher the horizon is the more time the solving takes and the higher the number of grid points is, the more memory is needed. By tuning these parameters we can therefore do a tradeoff between computational requirements and the quality the solution for each sampled model.

In the setting we tested, we had very weak prior information at the beginning. As a result, the POMDPs sampled at the beginning produced very poor policies; yet if we supposedly have to solve all of them exactly nonetheless. On the other hand, in the end of the execution, we needed to have very precise value functions and to solve the new sampled models as well as possible.

The higher the planning horizon and the number of grid points are, the more precise our solutions are. We diminish computation time intelligently by starting with low values for this parameters, and increasing them progressively with time.

In the same logic, we also start with a small pool of models, that increases at each sampling, until a maximum number of samples is reached.

4.7.2 Lost models

In large state space problem, with uninformed priors, especially at the beginning, some models are very poor. According to them, a sequence of actions and observation they experience will have a probability of happening that is nearly zero. Since very low probabilities tend to become 0, the belief state ends up being at 0 for every state.

In order to take this into account, the first thing we do is to detect when this happens and flag the models as "lost". The models are then removed from all procedures and weight computations and wait to be pruned out and replaced by another model or wait for the next scheduled restart, to be re-activated. When too many models are lost, the decision making procedures take random actions until the next scheduled restart, and non-query learning is disabled.

We have experimented ways of improving this simple setting. The first way was to diminish the weight of the lost model each time it gets lost, and to put the belief to a uniform belief in order for its belief to recover a legal value. Instead of putting them to an uniform belief, we also considered setting them to the belief of the non-lost model that was the closest[3] to them.

Actually this does not improve matters, since models that get lost once tend to be lost again and again, so with our first setting their weight tend to 0 anyways: pruning them out is the best thing we can do with them. But it is still useful to detect these lost models and remove them from the computations because otherwise the procedures can be flawed. Note that this issue happens only on large domains, at the beginning of the execution, when the prior is poorly informed. In realistic settings (our prior would be more reasonable) this would probably not happen.

[3]We used a simple sum-of-squared-parameter-differences metric.

```
Nreq= 0; h = {};
D =InitStructDirichlet(PriorInformation); (Cf Section 4.2).
for (i = 1...n)-(n = 20) do
    P_i =Sample(D);
    p0_i = p_i =ComputeLikelihood(D); (See Eq. 2.7).
    π_i = ComputeOptimalPolicy(P_i); (Pineau et al., 2003).
    b_i = b_0;
    β_i = b_0; (b_0 is known).
end for
for N loops do
    a =DecisionMaking(D, h, P_1 ... P_n, π_1 ... π_n); (See Section 4.3.
    z =ExecuteAction(a); h = {h, a, z};
    for (i = 1...n) do
        b'_i =BayesianUpdate(b_i, a, z);
        β'_i =BayesianUpdate(β_i, a, z); (See Equation 2.1).
    end for
    if (InfoGain()> ε_2)-(See Section 4.4.3) then
        if ((Variance()> ε_3)&( NReq < Nmin))-(See Section 4.4.3) then
            if (AltEntropy()> ε_1)-(See Section 4.4.3) then
                s' = OracleActiveLearningQuery(); NReq++;
                if (QueryFilteringOk(s, P_1 ... P_n))-(See Section 4.6) then
                    ∀s α(s, a, s') ← α(s, a, s') + Σ_i w_i β_i(s)λ;
                    α(s', a, z) ← α(s', a, z) + λ;
                    ∀i ∈ {1...n}β'_i(s') = 1 ∀x ∈ S {s'}β'_i(x) = 0; (λ = 0.2).
                else
                    NonQueryUpdate (D, a, z, P_1 ... P_n, λ'); (See Eq. 4.7 and 4.9, λ' = 0.002).
                end if
            else
                NonQueryUpdate(D, a, z, P_1 ... P_n, λ); (See Eq. 4.7 and 4.9, λ = 0.2).
            end if
        else
            NonQueryUpdate(D, a, z, P_1 ... P_n, λ'); (See Eq. 4.7 and 4.9, λ' = 0.002).
        end if
    end if
    for (i = 1...n) do p_i =ComputeLikelihood(D) end for.
    D =NonStationarityUpdate(D,ν, misesAJour); (Cf Section 4.5).
    if (CurrentIntervalEnds) then
        P_{n+1} =Sample(D);
        n ← n + 1; p0_{n+1} = p_{n+1} =ComputeLikelihood(D); (See Equation 2.7).
        π_{n+1} = C(P_n + 1); (Pineau et al., 2003).
        b_{n+1} =BayesianUpdate(b_0, h);
        β_{n+1} =BayesianUpdate(b_0, h,LastQuery); (See Equation 2.1).
        if (n = n_max) then
            DeletePOMDP(arg min(p_1, ..., p_n));
            DeletedPOMDP← P_{n+1};
        else
            n ← n + 1;
        end if
    end if
end for
```

Table 4.1: The MEDUSA script.

4.8 Conclusion

We have proposed an algorithm, MEDUSA, that brings a nearly optimal learning of a partially known POMDP model with the help of an external agent (or oracle). The algorithm is an extension of Active Learning to the POMDP framework that uses Bayesian Learning ideas to produce a robust setup, able to learn quickly. Even though the algorithm is not theoretically optimal as Decision-Theoretic Planning was, its heuristics are clever, and it can scale easily to large problems, as we show in Chapter 6.

MEDUSA allows two phases of learning; the agent first goes through an active learning phase during which it follows an explorative policy and has access to the oracle to enhance its learning; once this phase is finished the agent can continue its learning by itself, so that it can refine his model of the environment and become fully autonomous.

The MEDUSA algorithm allows applications in robotics, where the first phase would simply be a short calibration phase, where a human can be the oracle. Furthermore, since it is a model-based approach, once a model is learned it can be used as a prior for another task; this is made easy by the fact that MEDUSA allows to easily use priors; in robotics, the algorithm would therefore allow the calibration done for the first use of a given robot to be applied for other uses of the robot, even if the task is different.

Chapter 5

Theoretical properties of the MEDUSA algorithm

We have proposed the MEDUSA algorithm for active learning in POMDPs but we have only justified it intuitively. In this chapter we study the theoretical properties of our proposed algorithm. We analyze how our decision-making procedures behave when the number of models is infinite, and how this behavior is actually the best exploratory behavior one could have.

Furthermore, we explain the theoretical conditions under which MEDUSA is guaranteed to converge and learn the true model. This is a very crucial thing to study, since if with some conditions the algorithm is guaranteed to converge, it will, as long as the conditions are verified, give guarantees of convergence to anyone that would apply our work.

5.1 MEDUSA with an infinite number of sampled models

We analyze in this Section the policies executed by MEDUSA when the number of sampled models is infinite.

We use the notations of Section 2.1 for POMDPs. $\mathcal{D}_t$ is the Dirichlet structure corresponding to the POMDP at time t of the process. We call $\mathcal{M}$ the ensemble of PODMPs having $|S|$ states, $|A|$ actions and $|Z|$ observations.

Let f be some function, mapping $\mathcal{M}$ to a subset of $\mathbf{R}$. $\mathcal{H}$ is the ensemble of all the possible histories (containing all the possible sequences of actions and observations). Let $E(f|\mathcal{D}_t)$ be defined by Equation 5.1

$$E(f|\mathcal{D}_t) = \int_{m \in \mathcal{P}} f(m) p(m|\mathcal{D}) dm \tag{5.1}$$

Here, $p(m|\mathcal{D}_t)$ is the likelihood of $m \in \mathcal{P}$ according to the distribution $\mathcal{D}_t$. It is equal to the normalized product over the different Dirichlet distributions of the individual likelihoods. The likelihood of a set of parameter given a Dirichlet distribution is given by Equation 2.7.

$E(f|\mathcal{D}_t)$ therefore represents the expectation of function $f(m)$ when m was sampled from the Dirichlet system $\mathcal{D}_{\sqcup}$.

5.1.1 Preliminary theorem

We first present a preliminary theorem, that proves that the way the models are sampled in MEDUSA, combined to the way the weights are defined, allows to approximate

without bias the probability an action has to be optimal given an uncertainty model.

Theorem 1: *Take* $\mathbf{N}$ *models* $m_1 \dots m_{\mathbf{N}}$, *which were sampled according to the Dirichlet systems* $\mathcal{D}_{t1} \dots \mathcal{D}_{t\mathbf{N}}$. *Let* $p0_1 \dots p0_{\mathbf{N}}$ *be the likelihood of each of the models according to their corresponding distribution. Let* $p_1 \dots p_{\mathbf{N}}$ *be the likelihood of the models according to the current distribution* $\mathcal{D}_t$. *Let* $F \in \mathcal{M} \times \mathcal{H} \times A \rightarrow \{0; 1\}$ *be the function returning, given a model* $m \in \mathcal{M}$, *a history* $h \in \mathcal{H}$, *and an action* $a \in A$, *the value* 1 *if the action is optimal and the value* 0 *if it is not. Then, as* $\mathbf{N} \rightarrow \infty$*:*

$$\forall h \in \mathcal{H}, \forall a \in A, \sum_{i=1}^{N_m} \frac{p_i}{K p0_i} F(h, a, m_i) \rightarrow E(F(h, a, m)|\mathcal{D}_t), \text{ with } K = \sum_{k=1}^{N_m} \frac{p_k}{p0_k} \quad (5.2)$$

$$\textbf{Proof: } E[\sum_{i=1}^{N_m} \frac{p_i}{K p0_i} F(h, a, m_i)] = \sum_{i=1}^{N_m} \frac{p_i}{K p0_i} E(F(h, a, m_i)|\mathcal{D}s_i) \quad (5.3)$$

We will make an application of the **importance sampling** techniques. It is necessary to use correction factors because we want to evaluate the expected value of a function over a distribution by using samples which were drawn from other distributions. According to the importance sampling techniques, we have:

$$\forall i E(F(h, a, m_i)|\mathcal{D}_t) = \frac{K}{N_m} \frac{p(m_i|\mathcal{D}_t)}{p(m_i|\mathcal{D}_i)} E(F(h, a, m_i)|\mathcal{D}_i)$$

$$\text{Therefore, } E[\sum_{i=1}^{N_m} \frac{p_i}{K p0_i} F(h, a, m_i)] = \sum_{i=1}^{N_m} \frac{1}{N_m} E(F(h, a, m_i)|\mathcal{D}_i) \quad (5.4)$$

So, the function applied to the samples is an unbiased estimation of the expectation of the function $F(h, a, m)$ when m is sampled by the Dirichlet system D_t.

So, when the number of samples is infinite, the mean of this estimation therefore converges towards the expectation.

Importance Sampling is an efficient method to reduce bias when a process corresponding to a given distribution is evaluated to another distribution, Usually this is done at the cost of increasing the variance. Actually in our case the variance is not increased that much, since models are re-sampled regularly: each time a new model is sampled, its weight before normalization is 1. Big variance problems happen only when all the weights before normalization converge to 0, which therefore does not happen. We also rarely have a weight that dominates all others since all the weights ultimately converge to 0 (this is not true however for the real model which, if it is sampled, will have its weight before normalization converge to ∞, yet this just implies that if the real model happens to be sampled, the policy executed by MEDUSA will converge to the deterministic optimal policy of the real model, and this is not such a bad setting).

Note that when we drop the model with the lowest likelihood this usually has no influence on the estimations, since most of the time the associated weight of the model is nearly 0. Yet we drop models mostly to keep memory requirements manageable.

5.1.2 Limit of policies and heuristics

In Theorem 1 the left side of the equation is the probability action a has to be taken according to the exploratory `DecisionMaking` procedure in MEDUSA if the current situation is $\{h, m_1, \ldots, m_{N_m}, \mathcal{D}_1, \ldots \mathcal{D}_{N_m}, \mathcal{D}_t\}$ (see Section 4.3.1). The Theorem therefore implies that in the limit of an infinite number of models, this probability is the probability a model drawn according to $\mathcal{D}_t$ would have a as an optimal action.

This means that the exploratory policy defined in Section 4.3.1 is, in the limit of

an infinite number a policy that selects actions proportionally to the probability they have of being optimal according to the current uncertainty model.

Because executing a policy allows MEDUSA to estimate more precisely the parameters the policy "uses", this is therefore an optimal method to learn.

***Theorem 2**: The value function estimated by MEDUSA for history h, action a is unbiased and therefore converges to its expected value given an infinite number of models.*

$$\forall h \in \mathcal{H}, \forall a \in A, \sum_{i=1}^{N_m} \frac{p_i}{K\, p0_i} V(h,a,m_i) \rightarrow E(V(h,a,m)|\mathcal{D}_t), \text{ with } K = \sum_{k=1}^{N_m} \frac{p_k}{p0_k} \quad (5.5)$$

The proof is identical to the proof of Theorem 1, we just have to replace function F by function V.

Theorem 2 implies that if we pick the action with the greatest estimated value function like we do in the safe decision-making case (see Section 4.3.2), it is actually the action whose expectation has the greatest value. So it is indeed, if we don't consider learning perspectives, the best possible action.

Furthermore, if we consider Boltzman decision-making, this also guarantees that the values that are put into the computation of the probabilities are unbiased.

If we use proofs analog to the proof of Theorems 1 and 2, we can also prove that the heuristic functions `InfoGain`, `AltEntropy` and `PolicyEntropy` are also unbiased estimators. It means that the information gain estimation converges to the expected InfoGain given the Dirichlet, that the AltEntropy estimation converges to the expected alternate state entropy given the Dirichlet, and that the PolicyEntropy heuristic really converges to the real expected policy entropy.

However, the proof does not hold for BeliefVariance and ValueVariance. For these, we have no guarantee that our variance estimation will converge to the real variance.

5.2 Convergence of MEDUSA

We now study the conditions needed for the convergence of MEDUSA. We first explain what theoretical conditions are needed, and then explain how MEDUSA can meet them.

5.2.1 Conditions of convergence

Consider the following conditions: (C1) The learned model is stationary (and therefore the parameter ν defined in Section 4.5 is set to 1). (C2) through the Dirichlet system structure we do not impose any conditions that contradicts the true model.

Theorem 3: *If (C1) and (C2) are met, and if a query is made at every step, the estimation we have of the* ***useful*** *parameters converges with probability 1 to their true value if: (1) at each time step, each action has a non-zero probability of being taken. (2)the POMDP is reset to its initial state at regular intervals.*

As we said in Section 3.1, we call useful the parameters that can be evaluated through certain action sequences. Parameters that are not *useful* do not influence neither the value function nor the optimal policy.

Proof: The estimation we have for each parameter (the most likely value given the Dirichlet) is equal to the maximum likelihood estimation of the parameters given the samples seen since the beginning of the algorithm and given the prior which is $\forall i \in [1 \dots N]\, \theta_i^* = \frac{\alpha_i}{\sum_{k=1}^{N} \alpha_k}$. We know that the maximum likelihood estimate always

converges to the true value if there is an infinite number of samples. Note that the prior does not matter here: an infinite number of samples weights more than any prior we may have set. If a given parameter p is useful, there exists a sequence of action that from a starting state has some probability of taking the agent to a point where it can estimate parameter p. Since every action has a non-zero probability of being taken, this sequence has a non-zero probability of being performed. Furthermore, since we perform an infinite number of traces, we do obtain an infinite number of sample for parameter p. Therefore all useful parameters get an infinite number of samples and the convergence is proven.

5.2.2 Theoretically-guaranteed MEDUSA

The script presented in Table 4.1, with the exploratory decision-making, does not guarantee that each action has a non-zero probability of being taken at each time step. However, we can build a Theoretically-guaranteed MEDUSA by doing this small modification. Let ϵ_E be the *exploration rate*: at each time step the agent has the probability ϵ_E of accomplishing a completely random action.

***Theorem 4**: If (C1) and (C2) are verified, if we use the values $\epsilon_1 = 0$, $\epsilon_2 = 0$ et $\epsilon_3 < 0$, and if we add an exploration rate $\epsilon_E > 0$ then the useful parameters converge towards their true values given an infinite run.*

Proof: If `InfoGain`$= 0$, querying has no effect on the model we're learning, so skipping a query does not change anything. Furthermore, if `AltEntropy`$= 0$ it means that we do a non-query update with exactly the same effects as if we did a query. $\epsilon_3 < 0$ implies that we do not use the Variance heuristic. So, the algorithm we use is in this case is equivalent to doing a query at every step, except that we remove the queries that can be economized. Since using an exploration rate guarantees that each action has a non zero probability of being taken at every time step, the

conditions of Theorem 3 are therefore all fulfilled, and we can apply it, thus proving the convergence.

However, our experimental analysis of MEDUSA shows that giving a significant value to ϵ_E has a very bad influence on the quality of the policies executed during learning and on the quickness of convergence to the optimal policy. So if we want to have this theoretical guarantee, we need to lower our performance.

5.2.3 Why our implemented version still works

In most of the POMDP environments (in every POMDP in (Cassandra, 2004)), the reward structure is such that for any history, for every action, there exists at least one model such that the action is optimal. Note that this implies that there is an infinite number of them, since the optimal action function is constant per interval. We call this property the **non-coercion** property.

We can argue that if it is not the case, an action that would given a certain history, be sub-optimal for every assignment of the parameters has associated parameters that would have no influence on the optimal value function and on the optimal policy. So we might actually consider not learning these parameters.

***Theorem 5**: If the reward structure has the non-coercion property, if (C1) and (C2) are fulfilled, if the query strategy is the same as in theorem 4, and if we use an infinite number of models, then our MEDUSA algorithm with the exploratory policy is guaranteed to converge to the true model.*

Proof: If the condition on the POMDP is fulfilled, then according to the analysis made in Section 5.1.2 every action will have a non-zero probability of being taken for all possible history. We can therefore apply Theorem 3, which proves the convergence.

Theorem 5 justifies why we did not use an exploration rate in our experiments and why our tests were successful. We precise that the property described in this section is actually verified in all of the POMDPs to which we applied our algorithm.

5.3 Conclusion

Throughout this chapter we made a theoretical study of the MEDUSA algorithm. We have proven that the learned policies converge to the optimal as the number of sampled models goes to infinity. We have proven that under certain conditions, and in particular under an infinite number of queries from the oracle, the agent was guaranteed to learn the exact model and to find an associated optimal policy. We have discussed that, though a strictly positive exploration rate would be required for convergence, in most cases the convergence can be obtained without it.

We have proven that MEDUSA is more than a heuristic algorithm. It is actually a theoretically sound method, and is in the limit as optimal as the Decision-Theoretic Planning setting. Its learning is guaranteed and is guaranteed to be optimal under certain conditions, which were described in this chapter.

Chapter 6

Experimental analysis of MEDUSA

We have justified the MEDUSA algorithm intuitively and theoretically. However, for the sake of completeness we need to proceed to an experimental validation. We test many different settings, in order to prove the robustness of MEDUSA when confronted to problems of various size or difficulty, and do a study of the influence of its parameters.

6.1 Experiments on the tiger POMDP

The first set of experiments were done on the Tiger POMDP, that we described in 2.1.1. This environment has the property of being quick to solve, and so we did many experiments in it. We postulated three different kinds of prior information one could have about the problem.

6.1.1 With a single unknown parameters

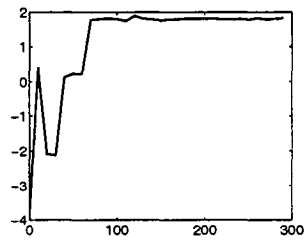
In this first experiment we know with full certainty every parameter except the probability of receiving the correct observation. We actually initialize it at 0.5, with a confidence of 1. We also know that this parameter applies in both states (with a symmetry).

The following experiment uses a model redraw at every step, and the graphs on Figure 6.1 are the means over 10 runs. We chose parameters that brought a reliable convergence with a minimum number of queries.

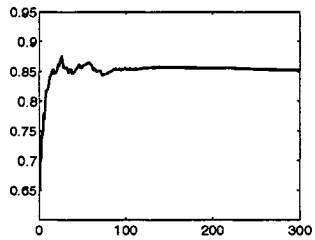
The mean discounted reward graph is built with the following experimental setting. At regular intervals, we simulate a pause in the learning, keep the current sampled models and the current uncertainty model. With these we apply the safe decision-making during a good number of runs (20,000). We compute the mean discounted reward associated to them, and then pursue learning. This setting allows to measure the reward returned by MEDUSA if we stopped learning at the current time step.

We first discover that very few time steps are actually needed to learn this parameter (300), and even fewer queries. The number of needed queries is usually around 33. Lots of queries can be spared because the transition when the Hear action is certain and it happens very often, We also see that the optimal reward is always reached quickly, and that the parameter converges to its true value (with a 0.5% precision). We see that on this problem, a 0.5% precision is sufficient to obtain the optimal policy.

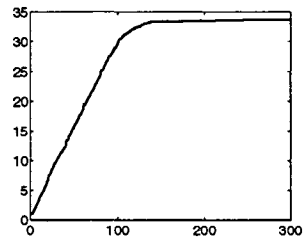
Let us comment on the evolution of the heuristics. As we described in Section 4.4.3, these are used to decide when to do queries. A good heuristic is one that is low when the currently executed policy is optimal. As we can see, Information



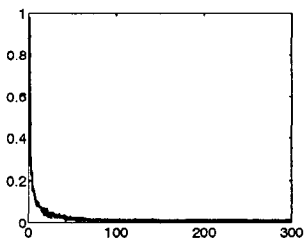
(a) Discounted Reward



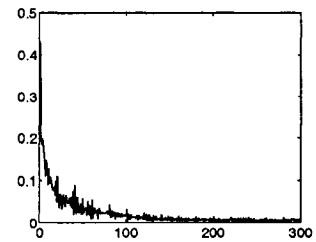
(b) Parameter Estimation



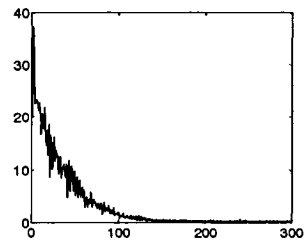
(c) Number of Queries



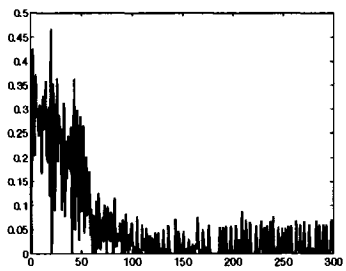
(d) Information Gain



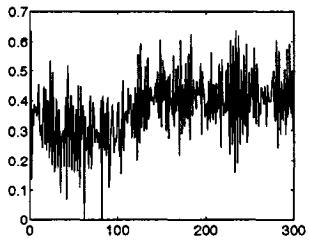
(e) Belief Variance



(f) Value Variance



(g) Policy Entropy



(h) Alternate State Entropy(1 run)

Figure 6.1: Mean results for MEDUSA on the Tiger experiment, where only one parameter is unknown.

Gain, Belief Variance, and Value Variance have very similar behavior: they decrease exponentially. This is a good behavior, since at the same time the reward increases in the same way.

We kept Information Gain because it is able to identify steps in which the queries are useless. However it is a bad indicator of how good our model already is. Even though Belief Variance and Value Variance have similar performance on this example, and both could be used, it is preferable to use Value Variance because the case might arise when the uncertainty over the belief state does not influence the value function, and in this case queries become useless. We can also see that Value Variance decreases more slowly than Belief Variance.

We also see that Policy Entropy is a bad heuristic to use, as its variance is very high. For instance we see that it continues to have positive values long after the optimal policy has been reached.

Plotting the mean of Alternate State Entropy over 10 runs would be pointless, since this heuristic is only used to decide when we can spare queries and is not supposed to be an indicator of convergence, so we plotted it over 1 run instead. It is cleared that it becomes non-zero as soon as a door is opened, and that it goes back to zero when a query is made. We can see that in the non-query learning part of the algorithm, it does not go back to zero anymore (since no query is made anymore).

We can see that when the query learning stops, around time step 120, the value of the parameter is still unprecise (0.87 in mean, instead of 0.85). During the steps of non-query learning it converges towards 0.85. Here the non-query learning does not fall into a local minimum and the parameter converges to the correct value. This is an indicator that non-query learning can work. It means that if we had an estimation of the parameters'd value, we could learn to correct it using non-query learning.

6.1.2 Justification of the query decision

We now compare the performance of different methods for the query decision. In Section 4.4.4, we have described the combination of heuristics we used for the query decision in MEDUSA. In this section we bring an experimental proof for the performance of the chosen query decision compared to a set of other query decisions. We show the performance of a set of query decisions, on the same Tiger problem with a single unknown parameter.

Since, as we showed on the previous Section, the policy entropy is a biased indicator, and since the variance on the belief state has a behavior similar to the variance of the value function, we selected only three heuristics. For each of these heuristics, Variance, InfoGain, and AltStateEntropy, we pick an optimal threshold and run experiments using each of these. We compare the resulting performances on Figure 6.2, where we plot the evolution of the mean discounted reward with the number of time steps for different possible query decisions, and on Figure 6.3, where we plot the evolution of the mean discounted reward with the number of queries.

The query decisions we study are the following. First, we use the **variance** on the value function. As we can see on the results, using only this heuristic may not be very efficient. We do not always converge to the optimal performance when the threshold is too high. The main advantage is that the number of queries is always bounded.

We also study a decision based on the **InfoGain** heuristic. From our experimental results, we see that it is more efficient than the Variance heuristic. It is actually the fastest heuristic to converge. But it takes more queries to reach the optimum than with the combination.

The third heuristic we study is the **Alternate State Entropy** heuristic. It

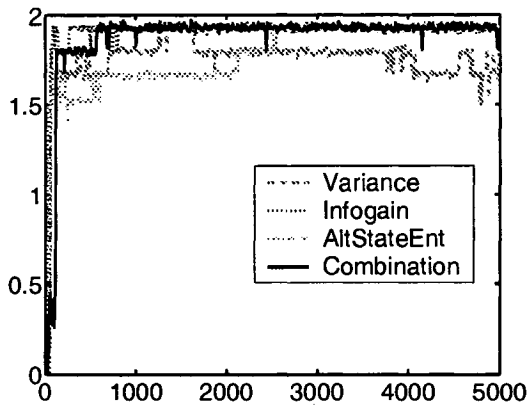


Figure 6.2: Evolution of the performance with the number of time steps for a query decision based on different heuristics.

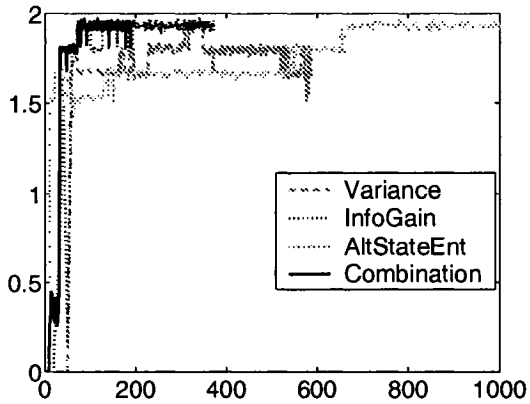
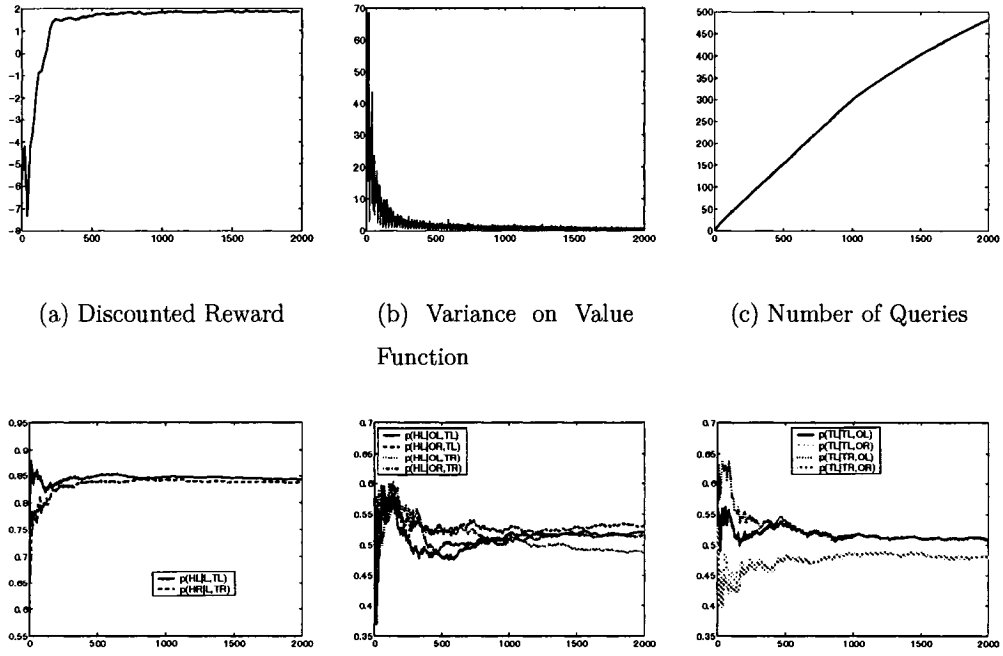


Figure 6.3: Evolution of the performance with the number of queries for a query decision based on different heuristics.

actually converges each time, it also does not stop doing queries. A lot of time step is also needed to reach convergence, and queries are made at nearly every time step.

The **combination** of the three heuristics that we used in final is better than each of them individually. This is particularly visible since the number of queries needed to reach the optimum is the lowest overall. However the convergence is slightly slower to reach in mean than when only Infogain is used. But the combination is better than infogain because less queries are made and because the process stops earlier.

These experiments therefore show that combining the heuristics brings better results than using each heuristic separately.



(a) Discounted Reward

(b) Variance on Value Function

(c) Number of Queries

(d) Observation probabilities, Listen

(e) Observation probabilities, Open

(f) Transition probabilities, Open

Figure 6.4: Mean results for MEDUSA on the Tiger experiment, where many parameters are unknown.

6.1.3 With several unknown parameters

In our second experiment, the prior information is the following: we assume that we know only one thing about the Tiger problem, and it is the fact that the Hear action does not change the state. All the other parameters are initialized at 0.5 with a confidence of 1. In this experiment we know anything about symmetries in the environment (for instance, the parameter of having a correct observation when Listen is performed is supposed to be different in the two states): the number of uncertain parameters is 22 and so we have 22 Dirichlet distributions. Experimental results are shown on Figure 6.4.

These experimental results show the following: since the variance on the value function has a different evolution depending on the run, in some of the runs it doesn't get under the threshold before 2,000 time steps and the algorithm keeps making queries even though the optimal policy has already been reached. We should probably have set the parameters differently to fix this. However, this perfectly illustrates the difficulty of picking a correct threshold value to make the query decision.

The parameters that are learned the most precisely are the correct observation probabilities (precision is around 2% in mean): this is a consequence of the fact that they are the parameters which influence the optimal policy the most (it is very important that they are learned precisely). The other parameters are learned only approximately (up to 10% difference for some of them on some of the runs). However some of these parameters have almost no effect on the optimal policy, in particular the parameters that correspond to the transitions when the incorrect door is opened. Overall, 100 queries are needed to get a reasonable reward and 300 queries are needed to find the best policy.

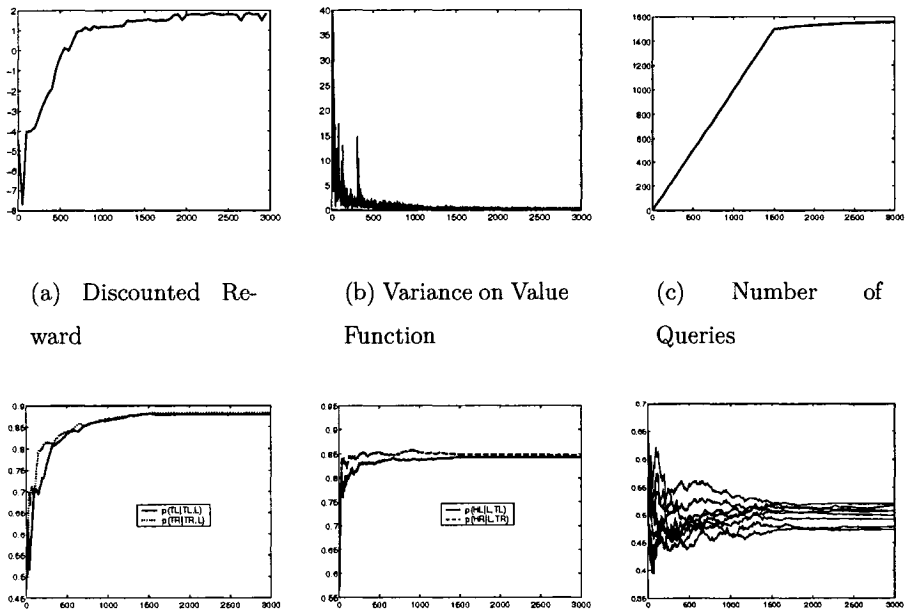
Furthermore, we see that non-query learning fails to correct the parameters. They fall to a local minimum. It does not work as well as in the previous experiment because here we have many uncertain parameters at the same time so there are many more local minima. To avoid them, we should probably use more query learning. However, even though the parameters does not converge to their exact correct values, the policy is still the optimal one. And this is what we wanted: we wanted to prioritize obtaining an optimal policy over finding exact values for the parameters. The ValueVariance heuristic helps ensure this actually happens.

6.1.4 With all parameters unknown

In this experiment, we have no knowledge at all about the POMDP model, and we learn everything from data All parameters are all initialized at 0.5 with a confidence of 1. This is a harder problem since the agent has to learn that the Listen action does not change the state. Note that this is the usual learning setting for the learning of POMDPs (Chrisman, 1992; Shatkay and Kaelbling, 1997; Kaebling *et al.*, 1998). Experience-based approaches manage to solve this problem so it is important to check if our algorithm is able to solve it.

As the value variance heuristic had too much variance to see when the model was good enough under this setting, it was necessary to use a coherent minimum of queries. We found out that in order to find out the optimal policy every time, it was necessary to impose at least 1,500 queries. As we can see on the experimental results shown on Figure 6.5, after 1,500 queries the parameters of the model are still not learned perfectly. And we can see that the model learned by non-query learning is not quite correct (the probabilities that the tiger moves when we listen that the algorithm finds is in mean of 0.9, whereas the true probability is 1.). However, one can see that the learned model has the same optimal policy as the real one *and* that the inexact learned model can very well explain some of the sequences of actions and observations experienced. We also observe that the probability of receiving the correct observation when Listen is performed is the only one that is estimated correctly, and it is because it is nearly the only parameter that has a relevant influence on the optimal policy.

This experiment shows that it is indeed possible to learn completely a model with queries. And it also shows that in order to find the best policy one need not learn the exact model. An approximate model can be sufficient to act optimally.



(a) Discounted Reward

(b) Variance on Value Function

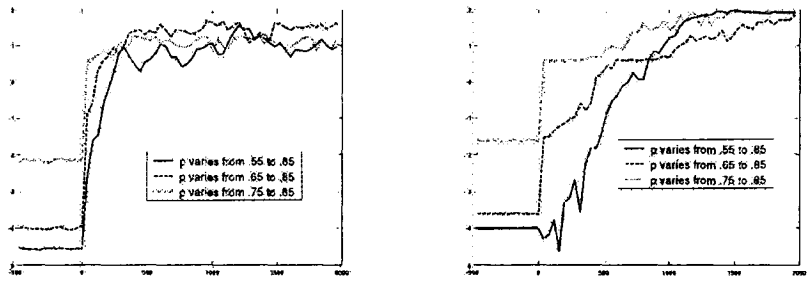
(c) Number of Queries

(d) Correct observation probabilities

(e) Probability of Tiger staying when Listen

(f) Other parameters

Figure 6.5: Mean results for MEDUSA on the Tiger experiment, where every parameter is unknown.



(a) Low equilibrium confidence (100)

(b) High equilibrium confidence (1000)

Figure 6.6: Evolution of the reward on the Tiger problem. There is a sudden change in the parameter p at time step 0.

6.1.5 Changes in the environment

We consider the question of whether MEDUSA is robust to 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. the sensors are damaged and become less precise, or there was another agent in the environment that suddenly changed its behavior). Throughout these cases, as we showed in Section 4.5, MEDUSA is designed to adapt to the changes and learn the new correct model. The ideas is that if the change in parameters is small, the non-query learning is sufficient, however if there are large changes, it is more efficient to resort to queries.

We perform tests on the Tiger domain, where we assume that the probability p of correctly detecting the tiger when Listen is performed suddenly changes. We set the parameter ν described in Section 4.5, that characterize the decrease in confidence for our past estimates, to different values. Figure 6.6 summarizes the results. As expected, the speed at which MEDUSA learns the new model depends on the confidence (sum of Dirichlet parameters). In each case, the algorithm detects quickly

that the parameter as changed (the value of the variance heuristic goes quickly above the threshold), and switches to query learning so that it adapts to the new parameter. This would be an interesting feature to have on a robot. If it senses that is environment has changed, it calls for the operator asking for a new calibration phase.

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 the performance before and after the change is slightly sub-optimal since we are cautious about further parameter changes). When confidence is high (>1000), the agent takes many more steps to learn new parameter values (and when the change is small (from 0.75 to 0.85) the agent actually takes around 100 time steps to realize that the environment has changed and to call for queries). Note that the optimal reward is reached at the end of the experiment and was reached at the beginning of the experiment.

6.2 Experiments on classical POMDP problems

We now show that MEDUSA can work on many problems other than Tiger. In this section we apply it to classical POMDP problems with very different settings.

A set of POMDP problems is available for standard evaluation of algorithms in (Cassandra, 2004). For each of them, we build the Dirichlet structure using the following procedure: for each parameter that correspond to a deterministic transition or observation, we set the parameters as certain to 1. For each parameter that correspond to an impossible transition or observation we set the parameters as certain to 0. All the other parameters are completely unknown. We also impose constraints on the parameters to accelerate learning: for example if in the real model one of the distribution is [0.1 0.2 0.7] and another is [0.7 0.2 0.1], we detect it and impose that these two distributions use the same hyper-parameters. The correspondence is not the same in the second case. None of these constraints contradict the true model,

PDM-PO	—S—	—A—	—Z—	N_α	Req.Med.	N.Ex.PSR	Return	%
Paint	4	4	2	6	700	4000	3.27	100
Shuttle	8	3	5	10	0	1024000	32.80	100
Network	7	2	4	36	1300	2048000	244	95
1d maze	4	2	2	3	100	-	1.25	100
4x4 maze	16	4	2	15	200	-	3.73	100
4x3 maze	11	4	6	16	800	1024000	1.90	100
Cheese	11	4	7	10	100	32000	3.48	100
Mini-hall 2	13	3	9	12	75	-	2.71	100
Hallway	60	5	21	84	450	-	1.02	90

Table 6.1: Description of the POMDP problems of the repository.

and most of the time they are reasonable hypothesis that one can make given the problem, considering for instance state symmetries[1].

In Table 6.1, we give the total number of hyper-parameter corresponding to our system N_α; we also give, for each model. its number of states, of actions and of observations. "Return" is the optimal return for the true model. N.Ex.PSR indicates the number of samples needed to learn the model with an experience-based method (Wolfe *et al.*, 2005), without any prior: our results are also given in the table: % is the percentage of the optimal return that we manage to reach and "N.Req" is the number of requests that is needed to reach this value.

Note that all these experimental results were obtained with the same set of values for the parameters of the algorithm. The only parameter that we changed was the minimum number of queries needed: it can be 0 for the Shuttle problem and needs to be at least 1000 for the Network problem.

[1]Even when the exact same distribution was found for a transition probabilities set and for an observation probabilities set, we used a different set of hyper-parameters, since it is very unlikely that the two are actually related.

Figure 6.7 shows the return evolution for all these learning problems. Here are some comments:

In some cases the policy is very good at the start, even though we have not learned anything yet. It means that the parameters that are uncertain do not influence the optimal policy at all. In the Shuttle problem, the uncertain parameters do not influence the optimal policy at all so no learning is needed.

In some cases (like 4x3) the true distributions that are uncertain are actually uniform. Since we initialize our Dirichlet parameters in every case at a uniform distribution with low confidence, the policy is actually better at the beginning than what it gets after the learning has begun. This is because the first samples modify our uniform distribution so the policy becomes worse. After, the agent does realize that the distribution is uniform and the policy becomes optimal again.

In mostly every case the optimal policy is reached. In Hallway and Network the policy we find is slightly sub-optimal. However if we didn't use a Variance threshold and switched to non-query learning we would observe convergence.

Our learning 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.

6.3 Influence of the precision of the prior

We present experiments with MEDUSA on the Tiger-grid POMDP problem (Littman *et al.*, 1995). We have considered three kind of priors. The first one, simple, only supposes that the structure of the maze is known, and we do basic symmetry assumptions: the number of corresponding alpha-parameters is 164. The second one, more

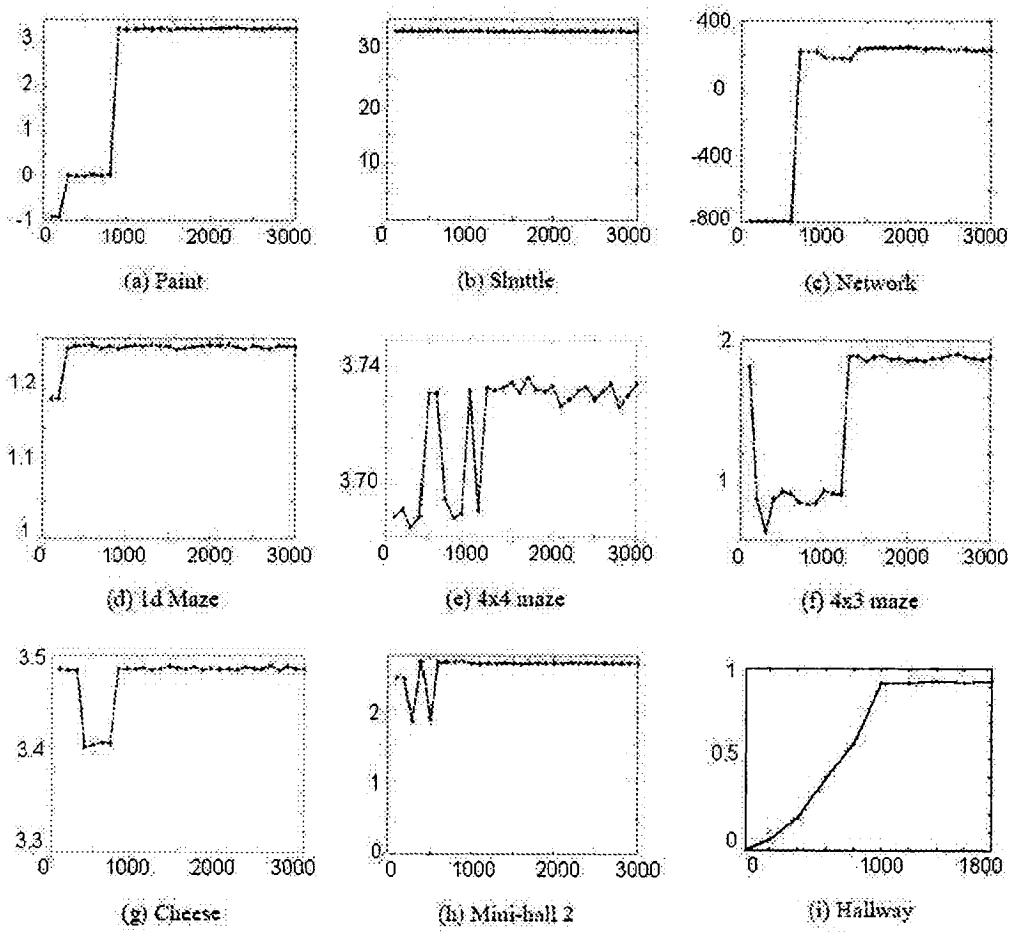


Figure 6.7: Evolution of the mean discounted reward on several classical POMDPs.

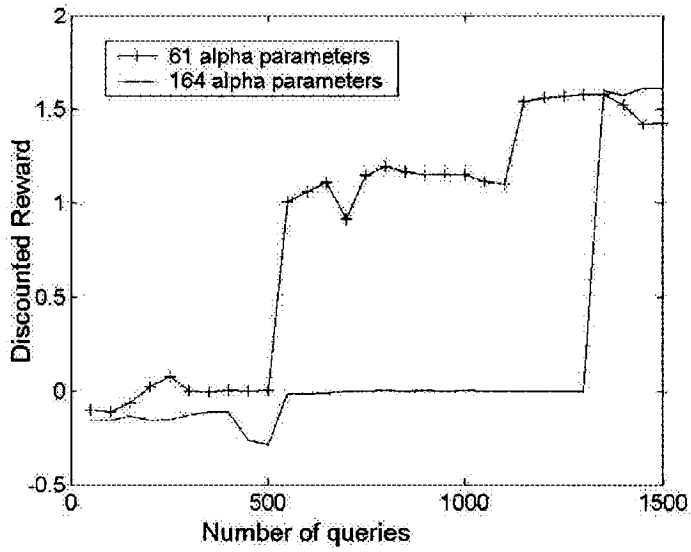


Figure 6.8: Experimental results on Tiger-Grid.

complex, supposes we know the relationship between the observation distributions: the corresponding number of alpha-parameters is 61. We also tried a third experiment, in which we used only 8 alpha-parameters. There was 4 distributions, 3 for each action and 1 for the observations. We learned only one probability, which was the probability of the action "succeeding", or the probability of the observation being "true". We supposed that all the other transition outcomes or wrong observations were equally likely.

The following figure shows the experimental results corresponding to the first and second setting. In the third case, the reward does not increase at all. The optimal policy remains "do nothing". Actually with the setting we had we could not capture the right set of parameters, and as we said earlier there is only a small set of parameters that leads to an interesting policy.

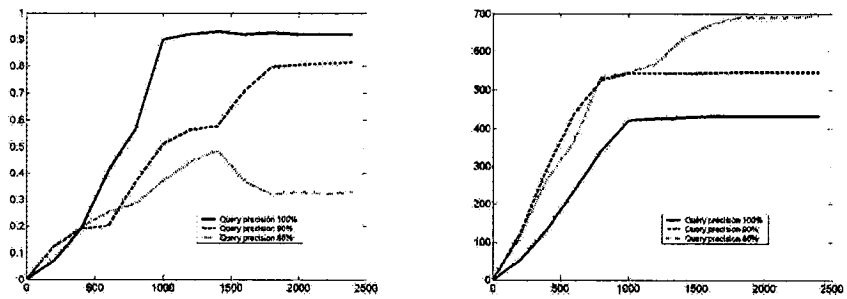
We have to tell, however, that this experiment needs huge amount of computation time. Furthermore, the parameters have to be tuned very precisely. And even then, the convergence sometimes do not always happen. This is known to be a difficult POMDP problem to solve, even when the parameters are perfectly known. Learning

it is an especially difficult task, because the range of parameters in which the policy is other than "do nothing" is very narrow. Figure 6.8 shows results with the 2 first settings. They, however, show only one run, in a case where the convergence did happen. Note that the optimal policy is not obtained, we only have approximately 60% of the optimal reward.

6.4 Experiments with a noisy oracle

We investigate MEDUSA's robustness to mistakes in the query responses. MEDUSA's query learning assumes that an oracle can provide exact state identification on demand. However it 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 as 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 have investigated the issue in the context of the standard Hallway domain (Littman *et al.*, 1995). It is a small maze, and its characteristics were given in Table 6.1. 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 6.9, 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 6.9 that the number of queries is still climbing. Only one run is shown, yet we have made several and discovered that the variance over the process is very high, especially in the 80% case. The mistakes made by the oracle give the learning process a very high variance.

It becomes unpredictable.



(a) (b)

Figure 6.9: 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.

6.5 A robotics application

We have investigated applications of MEDUSA on real robotics problems, as the algorithm was designed especially for this kind of applications. We used the Carmen robotics simulator (Thrun *et al.*, 2000). We allow MEDUSA to get data from the simulator (we get a map from it and discretize it) to produce a POMDP and to build a prior associated to it. We then execute MEDUSA to learn the resulting POMDP.

The exact POMDP that we investigated is a version of the HIDE problem (Pineau *et al.*, 2003). In this environment the agent has to reach a moving person. The position of the robot is supposed to be perfectly known, and the movements of the robot are perfectly known and deterministic. However, the position of the person at the start is unknown, as is the model of its movements. The person can move only to the neighboring cells, which it does, stochastically. Furthermore, in the observation model, when the robot is one square away from the person he has a non-zero probability of seeing it in the appropriate direction. When the robot is in the same cell as the person it is certain to see it. The prior knows all these facts: it knows that the

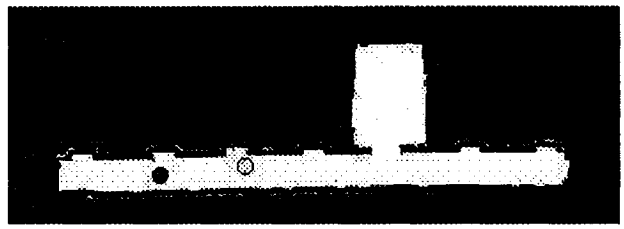


Figure 6.10: Map used for the Carmen experiment.

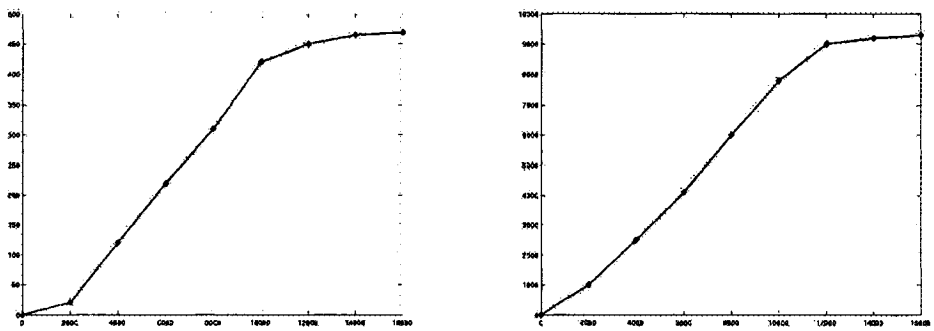
probabilities of the person moving two cells away is zero, and that the probabilities of seeing the person is zero except when near it, and it knows that the probability of seeing the person when in the same cell is 1. The behavior of the person is unknown, as is the precision of the sensors when the robot is near it.

When the robot is in the same cell as the person, it can use the catch action, and if the person really is here it gets a high reward and the problem ends. If the robot uses the catch action and the person isn't here it gets a bad reward.

We discretize the map into 19 cells. The associated POMDP has 362 states, 24 observations and 5 actions. MEDUSA is trying to learn 52 alpha parameters, which are the parameters that correspond to the person's behavior and the parameters that correspond to the robot sensors.

As we can see from Figure 6.11, MEDUSA converges within roughly 12,000 time steps, after having received answers to approximately 9,000 queries. 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. 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 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 6.11: 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.

6.6 Influence of the parameters

MEDUSA has many parameters (see Table 4.1) and all of them have an influence on the performance. As it would have been tedious to run every experiment on every set of parameters, we summarize here what we discovered experimentally when we changed the parameters.

On the hardest problems (Hide, Tiger-grid), it is difficult to find parameters such that convergence is obtained in a reasonable amount of computation time and time steps. The main issue is that when the quality of the computed policies is diminished, the exploratory behavior may not be very good; on the other hand, when the computed policies are of good quality, they take time to be computed. The right tradeoff needs to be made.

Learning rate λ: The learning rate can be equal to 1. A bigger learning rate can lead to a convergence to a local minimum. A lower learning rate increases the number of step needed to learn and can be used to compensate for a low re-sampling rate (in which case 0.2 can be used).

Non-Query learning rate λ': we usually took 0.01λ, but this might need to be lower. Actually, if we are willing to spend a lot of time steps doing non-query learning, it can be very low. It also should be low if the sampling frequency is low. When it is high, the convergence to a local minimum is accelerated, which we may not wish.

Resampling frequency : This definitely should be as high as possible. Drawing one model at every time step is the best possible setup (the experimental results with Tiger are made with this frequency). The minimum frequency should be around every 20 time steps. Having a low resampling diminishes the quality of the heuristics, of the non-query updates and of the policies executed. We recommend using a frequency as high as computationally possible.

Restarting frequency : Some POMDPs don't even need restarts to be learned whereas some need frequent ones. When one is setting this parameter, one should consider if the agent can become stuck in some subparts of the state space under some circumstances. If it is the case, restarts need to be done. Then they should not be too frequent. Note that in a real setting, with an operator, it would be easier to decide when to restart. A way to decide the restarting frequency would be to consider the smallest N such that $R_{\max}\gamma^N < \epsilon$.

Value Variance threshold : This parameter is tricky to set. If it is too low, unnecessary queries will be made. If it is too high, non-query learning will be done

too early and we will converge to a sub-optimal policy. If there is a lot of certain parameters it can be quite high, since non-query learning is more efficient. However, when the number of certain parameter is low it can be very low. Typical values range from 0.1 to 1.

Minimal Number of Queries : This parameter varies a great deal between domains. In some cases it can be equal to 0 (Tiger) and in others it needs to be 5,000. It should be high when the initial prior confidence is low and when the number of uncertain distributions is high. If one has no clue, the best is to set it to the amount of queries the oracle is willing to answer.

Value of information threshold : This parameter should be very low. For example, 10^{-5} is a good value. It depends on the precision of model we want to learn.

Alternate entropy threshold : This parameter should be at most 10^{-2}. Actually if we believe that some transitions may occur with very low probabilities, it should be decreased. If we do not know, it might be better to simply set it to 0.

Starting/Maximum number of models : The maximum number of models should be such that the memory needed by the program is not too high (on big problems 10 is a maximum). Furthermore, if the frequency of re-sampling is low we need to have a low maximum number of models, as we do not want inadequate models to live too long. The starting number should be low if the prior is poor and high if it is good.

Exploration rate : Empirically, 0 is the best value. Random actions usually lead to bad performance. However, on some complex problems in which a very good

knowledge of the environment is needed to find an approximatively good policy (for instance, Tiger-grid), or in cases where the prior information is weak, it might be useful to set it to a strictly positive value. But note that as soon as the exploration rate is set to a non-zero value lots of useless parameters are learned so the learning always takes more time.

6.6.1 Time and Memory Requirements

A good property of our setting is that the complexity is only linearly increased with respect to the POMDP solving complexity. However, MEDUSA uses lots of memory. The memory resources of the computer are mostly used to maintain the Dirichlet priors, the pool of models and their solutions. For the bigger problems (Hallway, Tiger-Grid), the memory used can be up to 200 megabytes of memory for 10 models (with 1000 grid points). Note that we can diminish the memory requirements by decreasing the number of models or decreasing the number of grid points per model: we can do a tradeoff between memory ressources and the quality of the policy executed. New point-based algorithms have been proposed, which are more efficient in their use of the belief points (Pineau, 2004; Pineau and Gordon, 2005)

The costly operation in time in MEDUSA is the redrawing/solving of a new model, which happens in our current setting at regular intervals. For the complex problems like Tiger-Grid, the redrawing/solving can take up to 60 minutes (with a horizon of 30 and 1000 grid points). Note that we can accelerate the solving routine by decreasing the horizon considered but this decreases the quality of the policy executed.

6.7 Conclusion

Throughout this chapter we have shown experimentally the performance of MEDUSA on many different problems. We have shown that with our setup and reasonable priors we were able to learn a near-optimal behavior in all the environments considered. The necessary number of queries and time steps was usually very small, even for large problems. The reliable behavior of the algorithm through different setups (non-stationarity, noisy queries, different levels of priors, different problem sizes) is a very good asset for applications on real problems. However, we do acknowledge that on the most complex learning problems, our algorithm can have a high variance and can also be parameter-sensitive.

Chapter 7

Conclusion

This thesis studies the problem of making optimal decisions in Markovian environments that are stochastic, partially observable and partially known. It is a problem in which the aim is to make good decisions despite uncertainty in the model, and by learning the most useful parameters of the model quickly so that the learning period is minimal.

We presented the POMDP framework, and explained how it allowed agents to make optimal sequential decisions in partially-observable stochastic environments. We described existing methods to learn parameters and explained how a more efficient approach was desirable in order to solve large realistic problems.

We described the active learning setting for partially observable environments. Using ideas from the Active Learning field (Cohn *et al.*, 1996; Anderson and Moore, 2005), we have made the oracle assumption, and built the theoretically optimal **Decision-Theoretic Planning** algorithm.

However Decision-Theoretic Planning is intractable to large environments, so we built the **MEDUSA** algorithm, which is an adaptation of Bayesian Learning ap-

proaches (Dearden *et al.*, 1999; Strens, 2000) to partially observable encironments. It also uses the oracle assumption. MEDUSA is a robust algorithm, able to include all kinds of prior information, to learn in the presence or absence of an oracle, to economize the requests to the oracle, and to whistand non-stationarity and noise in the oracle.

We have studied the theoretical properties of MEDUSA, and in particular its optimality when it used an infinite number of models and its convergence to the true model with an infinite number of oracle queries.

We have exposed experimental results of MEDUSA on several problems, of different sizes and settings, including problems that were inspired from robotics applications. We have showed that it was an efficient algorithm, able to learn an optimal policy quickly when a reasonable prior was available.

7.1 Contributions

We have studied a new problem. Even though learning in POMDPs had already been studied no one had studied how an oracle could be used to accelerate the learning and guarantee its convergence. Furthermore, previous approaches did not explicitly minimize the length of the learning period, and mostly used random actions through the environment in order to learn.

This problem is relevant in many areas and especially in robotics, in which the underlying state is something that can be measured by an external agent. The active learning phase is then a calibration phase, in which the external agent can be a human that supervises the learning agent.

Both approaches proposed in this thesis are model-based in the sense that we can

use all the prior knowledge we have about the environment to accelerate learning and in the sense that model parameters are estimated explicitly during learning. These parameter estimations can easily be used as priors for other tasks in a similar environment.

Our MEDUSA algorithm also brings an answer to previously unanswered problems: how can an agent act optimally from an exploitation point of view in the presence of partial observability and uncertainty in the parameters of the model, and how can the agent detect and follow changes in the parameters of the model when the model is non-stationary. The safe decision-making allows the first and the non-stationarity adjustments allow the second. It is very useful that the agent be able to detect unpredicted changes in the environment and request a new calibration phase to correct its estimations when necessary. This is a useful feature to have in a robot. Another important fact is that, while the oracle assumption is a strong one, we show results demonstrating that MEDUSA can be adapted to noise in the query answers.

If we compare our solutions to previously existing methods, we make the following remarks: decision-making in model-free POMDP approaches can learn and plan despite uncertainty in both state and model (Chrisman, 1992; McCallum, 1996; Shatkay and Kaelbling, 1997; Singh *et al.*, 2003; Shani and Brafman, 2004; Shani *et al.*, 2005b). However these approaches have not scaled to large-scale domains, due to their intensive data requirements. Our methods propose a more flexible trade-off between knowledge engineering and data requirements. Initial knowledge can be introduced through model priors. Our prior engineering and the queries allow the agent to learn very quickly nearly optimal policies even on very large domains. Our method is more flexible than the method based on USM (Shani *et al.*, 2005b) which requires a perfectly known model of the observations; we can use any kind of prior. Furthermore, the queries allow us not to fall into local minima as it happens with EM methods (Chrisman, 1992; Shatkay and Kaelbling, 1997).

7.2 Future work

About the Decision-Theoretic Planning approach that we studied in Section 3.2, we need to investigate if we can use a more appropriate solving method, that is more adapted to the structure of the resulting hyper-POMDP than classical solving methods. We also want to investigate if there can be a theoretical method to set the query cost so that the queries are used only to learn. For instance, we have considered having an additional state feature keeping track of the number of queries already made. The cost of the queries would increase with this new state feature.

We also want to improve MEDUSA in several ways. Its main drawback is currently the weak performance of the non-query learning. We can improve this: since the decision-making procedure is not directly influenced by the learning procedure, we can, before we do any non-query learning, wait for the next query (or wait for the next time the belief state becomes certain in one state). We can then execute the Baum-Welch algorithm (Baum, 1972) in order to find what was the most likely sequence of states we went through given our actions and observations, and do query updates for this sequence. This setting would allow the future to influence the learning of the current step, which is not the case right now. This would give a more efficient non-query learning and help diminish the number of needed queries. However we would then have to consider a different set of heuristics for the query decision.

We also want to work on how we can re-use the models that are already solved to solve similar ones. In our current setting we waste ressources re-solving parts of our POMDPs that do not change from one model to another. Furthermore we focus on parts of the model that are not used at all while the model is maintained. In order to correct this, we consider the possibility of solving the model on-line, by doing updates only on the belief state points that are being visited, in a setting similar to the RTBSS algorithm described in (Paquet *et al.*, 2005).

To apply MEDUSA to real robots, we intend to make the redrawing/solving routine in a thread separated from the decision making routine so that the time spent in the redrawing/solving routine does not penalize the rapidity of the robot. There would be a third thread applied to the querying/learning subroutine since these does not affect directly any of the other two processes except when we update the weights of the models. This third thread would allow some delay in a query answering, without affecting either of the two other processes.

The next versions of MEDUSA will certainly use this setting, with three parallel processes. Actually the idea is to use different processes for the "Decision-Making", "Learning", and "Re-Sampling" boxes in the MEDUSA chart of Figure 4.1. The algorithm could even be more efficient if a good number of processors could be used in parallel, since it would be possible to run several redrawing subroutines in parallel. Of course, our study will also focus on how to economize computational power, since we often solve many models which are nearly the same: we would like to use the information given by the solution of one model to help building the solution to another one.

With all these modifications MEDUSA should be well-adapted to complex robotics problem. The system could probably be adapted to a multi-agent setups when the behavior of the opponent or partner is anticipated and learned. The algorithm should also have a wide range of applications outside robotics.

7.3 Conclusion

We presented a new way to look at sequential decision-making in stochastic, partially-observable and partially known environments. Our approach gathers ideas from POMDPs, Active Learning, and Bayesian Learning. Our algorithms, Decision-Theoretic Planning and MEDUSA, offer a way to do an efficient model-based active

learning. Experimental results show that the setting can be applied successfully to large domains, and that applications on real robots are feasible. We strongly believe that the methods presented here will allow to build reliable Artificial Intelligence agents that are adapted to real-life problems.

Bibliography

A. CASSANDRA, A., KAELBLING, L., AND KURIEN, J. (1996). Acting under uncertainty: Discrete bayesian models for mobile-robot navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems.*

ANDERSON, B. AND MOORE, A. (2005). Active learning for hidden markov models: Objective functions and algorithms. In *ICML.*

BAGNELL, J. AND SCHNEIDER, J. (2001). Autonomous helicopter control using reinforcement learning policy search methods. In *ICRA.*

BAUM, L. E. (1972). An inequality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. *Inequalities, 3:1-8.*

BRAFMAN, R. I. (1997). A heuristic variable grid solution method for POMDPs. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97)*, pages 76–81, Providence, Rhode Island. AAAI Press / MIT Press.

CASSANDRA, A. (1998). *Exact and approximate algorithms for partially observable Markov decision processes.* Ph.D. thesis, U. Brown.

CASSANDRA, A. (2004). www.pomdp.org: Cassandra's pomdp page.

CHRISMAN, L. (1992). Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *AAAI*, pages 183–188.

COHN, D. A., GHAHRAMANI, Z., AND JORDAN, M. I. (1996). Active learning with statistical models. *J. Artif. Intell. Res. (JAIR)*, 4:129–145.

CRITES, R. (1996). *Large-scale Dynamic Optimization Using Teams of Reinforcement Learning Agents.* Ph.D. thesis, University of Massachusetts.

DARRELL, T. AND PENTLAND, A. (1996). Active gesture recognition using partially observable markov decision processes. In *IEEE International Conference on Pattern Recognition (IPCR'96).*

DEARDEN, R., FRIEDMAN, N., AND ANDRE, D. (1999). Model based bayesian exploration. In *UAI*, pages 150–159.

DEVROYE, L. (1986). Non-uniform random variate generation.

HAUSKRECHT, M. (1997). *Planning and Control in Stochastic Domains with Imperfect Information.* Ph.D. thesis, Massachusetts Institute of Technology.

KAEBLING, L. P., LITTMAN, M. L., AND CASSANDRA, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134.

KAPLAN, R. (1969). Optimal investigation strategies with imperfect information. *Journal of Accounting Research.*

LITTMAN, M., CASSANDRA, A., AND L.KAELBLING (1995). Learning policies for partially observable environments: Scaling up. *Technical Report. Brown University.*

MCCALLUM, A. K. (1996). *Reinforcement learning with selective perception and hidden state.* Ph.D. thesis, University of Rochester. Dept. of Computer Science.

MCCALLUM, R. (1995). Instance-based utile distinctions for reinforcement learning with hidden state. In *ICML.*

NG, A. Y. AND JORDAN, M. (2000). Pegasus: A policy search method for large mdps and pomdps. In *UAI.*

NOURBAKHSH, I., POWERS, R., AND BIRCHFIELD, S. (1995). Dervish: An office-navigating robot. *AI Magazine.*

PAQUET, S., TOBIN, L., AND CHAIB-DRAA, B. (2005). An online pomdp algorithm for complex multiagent environments. In *AAMAS.*

PINEAU, J. (2004). *Tractable Planning Under Uncertainty: Exploiting Structure.* Ph.D. thesis, Carnegie Mellon University.

PINEAU, J. AND GORDON, G. (2005). Pomdp planning for robust robot control. In *ISRR.*

PINEAU, J., GORDON, G. J., AND THRUN, S. (2003). Point-based value iteration: An anytime algorithm for POMDPs. In *IJCAI*, pages 1025–1032.

ROY, N., PINEAU, J., AND THRUN, S. (2000). Spoken dialog management using probabilistic reasoning. In *Association for Computational Linguistics (ACL).*

ROY, N. AND THRUN, S. (1999). Coastal navigation for mobile robots. In *NIPS.*

SHANI, G., BRAFMAN, R., AND SHIMONY, S. (2005a). Adaptation for changing stochastic environments through online pomdp policy learning. In *Workshop on Reinforcement Learning in non-stationary environments, ECML.*

SHANI, G., BRAFMAN, R., AND SHIMONY, S. (2005b). Model-based online learning of pomdps. In *ECML*.

SHANI, G. AND BRAFMAN, R. I. (2004). Resolving perceptual aliasing in the presence of noisy sensors. In *NIPS*, pages 1249–1256.

SHATKAY, H. AND KAELBLING, L. P. (1997). Learning topological maps with weak local odometric information. In *IJCAI*, pages 920–929.

SIMMONS, R. AND KOENIG, S. (1995). Probabilistic navigation in partially observable environments. In *IJCAI*.

SINGH, S. P., LITTMAN, M. L., JONG, N. K., PARDOE, D., AND STONE, P. (2003). Learning predictive state representations. In *ICML*, pages 712–719.

SMALLWOOD, R., SONDIK, E., AND OFFENSEND, F. (1971). Toward and integrated methodology for the analysis of health-care systems. *Operations Research 19.*

SMITH, T. AND SIMMONS, R. (2004). Heuristic search value iteration for pomdps. In *UAI*.

SONDIK, E. J. (1971). *The Optimal Control of Partially Observable Markov Decision Processes.* Ph.D. thesis, Stanford University, Stanford, California.

SPAAN, M. AND N.VLASSIS (2005). Perseus: Randomized point-based value iteration for pomdps. *AI Research 24 (195-220).*

STRENS, M. J. A. (2000). A bayesian framework for reinforcement learning. In *ICML*.

SUTTON, R. S. AND BARTO, A. G. (1998). Reinforcement learning: An introduction.

THIEBEAUX, S., CORDIER, M.-O., O.JEHL, AND KRIVINE, J.-P. (1996). Supply restoration in power distribution systems — a case study in integrating model-based diagnosis and repair planning. In *UAI*.

THRUN, S., FOX, D., BURGARD, W., AND DELLAERT, F. (2000). Robust monte carlo localization for mobile robots. *Artificial Intelligence, volume 128.*

WOLFE, B., JAMES, M. R., AND SINGH, S. (2005). Learning predictive state representations in dynamical systems without reset. In *ICML*.

ZHANG, N. AND W.LIU (1996). Planning in stochastic domains: problem characteristics and approximation, technical report hkust-cs96-31.