Abdeslam BOULARIAS

Predictive Representations For Sequential Decision Making Under Uncertainty

Thèse présentée
à la Faculté des études supérieures de l'Université Laval
dans le cadre du programme de doctorat en informatique
pour l'obtention du grade de PhilosophiæDoctor (Ph.D.)

Département d'informatique et de génie logiciel
Faculté des sciences et de génie
UNIVERSITÉ LAVAL
QUÉBEC

2010

Résumé

La prise de décision est un problème omniprésent qui survient dés qu'on fait face à plusieurs choix possibles. Ce problème est d'autant plus complexe lorsque les décisions, ou les actions, doivent être prise d'une manière séquentielle. En effet, l'exécution d'une action à un moment donné entraîne un changement à l'environnement, ou au système qu'on veut contrôler, et un tel changement ne peut pas être prévu avec certitude. Le but d'un processus de prise de décision consiste alors à choisir des actions en vue de se comporter d'une manière optimale dans un environnement incertain. Afin d'y parvenir, l'environnement est souvent modélisé comme un système dynamique à plusieurs états, et les actions sont choisies d'une telle manière à ramener le système vers un état désirable.

Dans le cadre de cette thèse, nous avons proposé un ensemble de modèles stochastiques et d'algorithmes, afin d'améliorer la qualité du processus de prise de décision sous l'incertain. Les modèles développés sont une alternative aux Processus Décisionnels de Markov (MDPs), un cadre formel largement utilisé pour ce genre de problèmes.

En particulier, nous avons montré que l'état d'un système dynamique peut être représenté d'une manière plus concise lorsqu'il est décrit en termes de prédictions de certains évènements dans le futur. Nous avons aussi montré que le processus cognitif même du choix d'actions, appelé politique, peut être vu comme un système dynamique. Partant de cette observation, nous avons proposé une panoplie d'algorithmes, tous basés sur des représentations prédictives de politiques, pour résoudre différents problèmes de prise de décision, tels que la panification décentralisée, l'apprentissage par renforcement, ou bien encore l'apprentissage par imitation.

Nous avons montré analytiquement et empiriquement que les approches proposées mènent à des réductions de la complexité de calcul et à une amélioration de la qualité des solutions par rapport aux approches d'apprentissage et de planification standards.

Abstract

The problem of making decisions is ubiquitous in life. This problem becomes even more complex when the decisions should be made sequentially. In fact, the execution of an action at a given time leads to a change in the environment of the problem, and this change cannot be predicted with certainty. The aim of a decision-making process is to optimally select actions in an uncertain environment. To this end, the environment is often modeled as a dynamical system with multiple states, and the actions are executed so that the system evolves toward a desirable state.

In this thesis, we proposed a family of stochastic models and algorithms in order to improve the quality of of the decision-making process. The proposed models are alternative to Markov Decision Processes, a largely used framework for this type of problems.

In particular, we showed that the state of a dynamical system can be represented more compactly if it is described in terms of predictions of certain future events. We also showed that even the cognitive process of selecting actions, known as policy, can be seen as a dynamical system. Starting from this observation, we proposed a panoply of algorithms, all based on predictive policy representations, in order to solve different problems of decision-making, such as decentralized planning, reinforcement learning, or imitation learning.

We also analytically and empirically demonstrated that the proposed approaches lead to a decrease in the computational complexity and an increase in the quality of the decisions, compared to standard approaches for planning and learning under uncertainty.

Acknowledgements

I am grateful to my advisor, Brahim Chaib-draa, for his continuous support. Brahim was always there to listen and to give advice. A special thanks to Masoumeh Izadi, my collaborator from McGill University. I am indebted to my many lab mates, Alireza, Camille, Andriy, Hamid, Patrick, Guillaume, Pierre, Maxime, Jean-Samuel, Edi, Allain, Sébastien, Charles, Jilles, Olivier, Pierre-Luc, Julien, Pierrick, and Stéphane for providing a stimulating and fun environment. Lastly, and most importantly, I wish to thank my parents, Hamid and Sadia, my sisters, Houria and Sadjia, and my brothers, Samir and Sofiane. To them I dedicate this thesis.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The problem of making decisions is ubiquitous in life. Everybody is confronted every day to dozens of situations where a decision should be made, quickly and efficiently. Some of these situations are simple daily routines, such as choosing how to dress for the day, while others may require more mental efforts, such as driving a car to work. More importantly, one has to make a big life decision every once in while, such as choosing a major at the university, or a job after graduating. Therefore, we can certainly confirm that an important part of the cognitive process is dedicated to making decisions. A simple example of that is writing a text, such as this introduction, where the writer spends most of his time choosing lines of thought, examples, and words.

So, what makes some decisions harder than others? The answer to this question is not simple and has many facets, but it can be reduced to one principal cause: the uncertainty about the outcome of the decision. More precisely, it is the uncertainty about the *cost*, or the *reward*, that will be generated by a decision in a near or a far future.

This uncertainty is generally caused by a limited capacity of accurately modeling all the parameters and the variables related to a decision-making problem, or by an inherent indeterminacy within the problem itself, such as predicting the behavior of a human. This uncertainty can also be caused by a limited capacity of enumerating and processing all the possible outcomes of the decision, due to the high number of available choices, or to the duration of the decision's impact in the future.

In order to make a good decision, one should consider the outcome of the decision at any moment in the future, how this decision will change the context of the problem, and how it will interfere with other decisions that should be made in the future. This process, known as *Sequential Decision-Making Under Uncertainty*, has shown to play a key role in solving a large class of important problems, ranging from seemingly simple tasks, such as walking and path planning, to playing complex games, or conversing with others.

However, many real-world problems are so complex that they cannot be solved by humans. The aim of *Artificial Intelligence* [Russell and Norvig, 2009] is to ultimately automatize the process of solving these problems. In this context, the decision-maker is an autonomous entity called an *agent*, evolving in an *environment* defined by parts of the world that are related to the task of the agent. The decisions of the agent are called *actions*.

The computational approach to the decision-making problem is to represent the environment of an agent as a *dynamical system*. The definition of a dynamical system is intimately related to the definition of a *state*. The state of the system is all the information that is necessary for *predicting* the outcome of the agent's actions. A dynamical system is then a system in which the transition between different states can be formally described by a *transition function*. *Markov Decision Processes* (MDPs) are a popular formal framework for representing such systems. Partially Observable MDPs (POMDPs) are a variant of MDPs for representing systems where the agent cannot completely observe the state.

Given a dynamical system, a task is defined by associating a numerical reward, or cost, to every executed action. The goal of the decision-maker is to choose actions to change the state of the system so that it can maximize its total reward. The function that selects an action for each possible state is called a *policy*. Therefore, solving a sequential decision-making problem corresponds to finding a policy that maximizes the expected total reward, i.e., an *optimal policy*.

The problem of finding an optimal policy turns out to be untractable in systems with a large state space on a long planning horizon. The high dimensionality of the state space is partly due to the fact that the states are generally defined as a simple conjunction of all the variables describing the different aspects of the state. As an example of that, the state of a mobile robot can be defined by the current position of the robot, the position of its destination, its level of energy, the temperature of the weather, and so on.

In this thesis, we show that decision-making under uncertainty can be performed more efficiently when the state of a system is represented by directly *predicting* its future after a sequence of actions. Intuitively, a decision-maker should not be concerned by the complexity of its environment, but only about how the environment responds to its actions from a subjective point of view. For instance, if the task of a robot is to simply move forward while avoiding obstacles, then its state should indicate only the probability of hitting an obstacle after each move, and not the exact location of the robot. In general, a problem can be drastically simplified by changing its representation. To see that, one can try to multiply two large numbers using the Roman numerals, and then try to do the same operation using the Hindu-Arabic numerals.

In the next section, we present the different categories of the methods used for sequential decision-making under uncertainty. Then, we present the contributions of this thesis. We finish by giving a brief description of the content of this thesis.

1.1 A Categorization of Sequential Decision-Making Under Uncertainty Techniques

There are four principal types of approaches to decision-making under uncertainty, as depicted in Figure 1.1. Each one of these methods can or cannot be used depending on the available parameters defining the problem. The first one, known as *planning* or *optimal control*, consists in finding an optimal policy given a reward function (or a goal) and a model describing the dynamics of the system. If the dynamics model is unknown, than one can either learn it by observing the response of the system to different actions, or directly learn an optimal policy by trying different actions several times, and choosing the ones that lead to a higher overall reward. This latter type of approaches is known as *reinforcement learning*. Often, it would be better to show to the decision-maker examples of how to perform the task and let him find the general policy, rather than to let him execute costly actions. This type of learning is known as *direct imitation learning*, and it is generally used when no model of the dynamics is available. Even if a model is available, sometimes it is easier to specify the task by demonstrating it with examples and letting the decision-maker learn a reward function, than to handcraft a reward function. This method is called *inverse optimal control*. Finally, the decision-making process can involve several rational agents, forming a *multiagent system*.

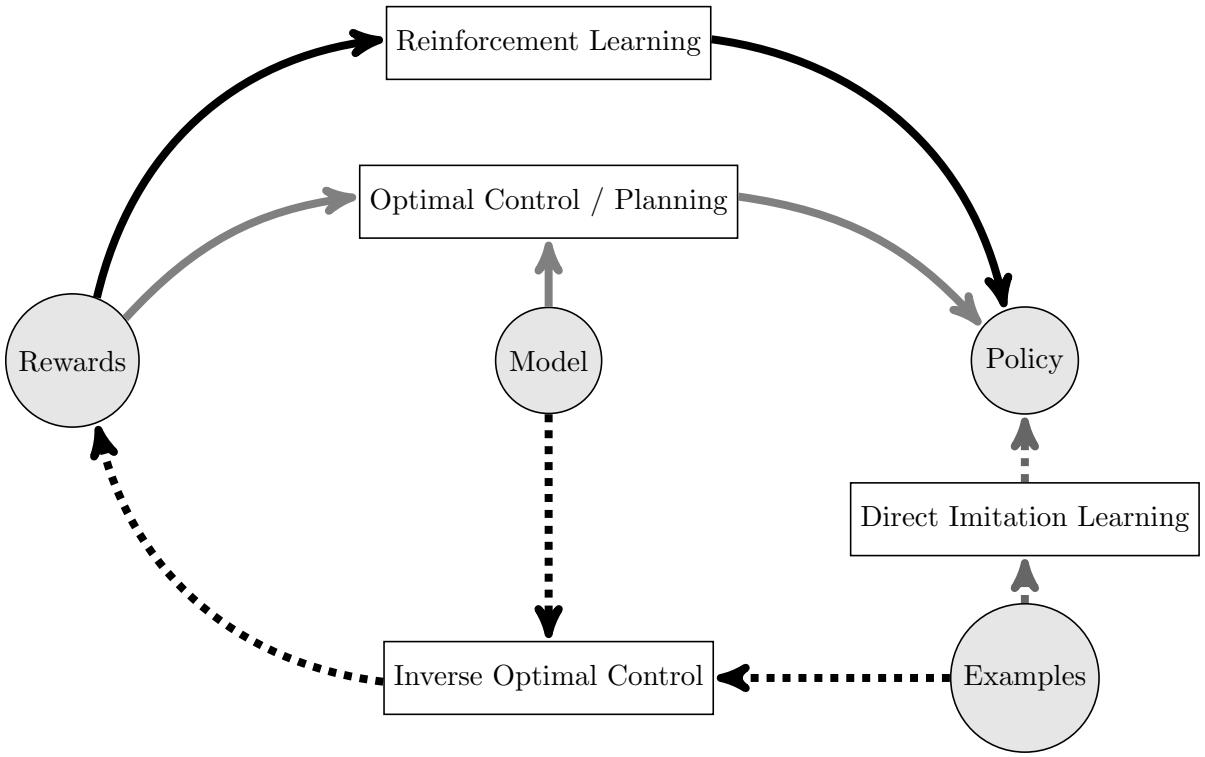


Figure 1.1: Different approaches to decision-making under uncertainty.

This thesis shows how efficient representations of states, rewards and policies can lead to an improvement of each one of these different approaches to decision-making under uncertainty.

1.2 Main Contributions

The key contributions of this thesis are:

A Predictive State Representation based on core histories (Chapter 2): In this thesis, we introduce a Predictive State Representations (PSR) based on special past events called *core histories*, that can be used for predicting the future events, and we provide a Bayes update rule for the *belief states* in this new model. Predictive State Representations (PSRs) are a family of predictive representations proposed by Littman et al. [2001]. PSRs are based on representing states using the probabilities of special future events called *core tests*. The probabilities of the core tests are a *sufficient information* for predicting the future of the system. Note that the existence of *core histories* has been already showed by James [2005], our contribution is the derivation of a Bayes rule for updating the belief state using core histories.

An algorithm for transforming Partially Observable MDPs to Predictive State Representations (Chapter 3): We propose an algorithm for finding an approximate and compact PSR model given a Partially Observable MDP (POMDP), based on the work of Poupart [2005] on compressing POMDPs. We formulate this problem as an optimization problem. Previously, Littman et al. [2001] proposed an algorithm for transforming a given POMDP into an equivalent PSR. However, the dimension of the equivalent PSR model is almost equal to the dimension of its original POMDP in most of standard benchmarks. Our algorithm minimizes the potential error caused by missing some core tests. In this context, we present an empirical evaluation on benchmark problems, illustrating the performance of this approach on prediction and planning tasks.

A Predictive Policy Representation (Chapter 4): We present a new model for representing stochastic policies in partially observable environments. This model, called *Predictive Policy Representation* (PPRs), can be seen as the dual of PSRs by switching the role of actions and perceptions. The principle of PPRs is to use PSRs for representing internal states of an agent instead of the states of the system. Note that the idea of using tests for representing policies was suggested by Wiewiora [2005], our contribution at this level is presenting a complete parametric model based on PPR and deriving a Bayes' update equation.

A point-based planning algorithm for Decentralized Partially Observable MDPs (Chapter 4): We present a new fast algorithm for planning in decentralized POMDPs, based on a simplified variant of the Predictive Policy Representations (PPRs). The major drawback of decentralized POMDPs is the large dimensionality of the internal state (the belief state) of a given agent. In this context, an internal state corresponds to a probability distribution on all the possible policies of all the agents in the system. Our algorithm is based on using the probabilities of specific events (tests) in order to predict the behavior of the agents in the future.

An optimal planning algorithm for Decentralized Partially Observable MDPs (Chapter 4): We propose a lossless policy space compression technique for planning with decentralized POMDPs. Our method is based on transforming high-dimensional distributions on policies to equivalent low-dimensional vectors. These vectors, called *reduced belief states*, indicate the probabilities of specific events using the predictive policy representations.

An algorithm for reinforcement learning in a partially observable environment (Chapter 5): We introduce a new model-free approach, based on PPRs, for reinforcement learning in partially observable environments. We also present a policy gradient algorithm for training PPRs. We empirically compare PPRs to Finite-State Machines, and show that better results can be achieved using PPRs.

An algorithm for learning by imitation in a partially observable environment (Chapter 6): We show how PPRs can be used for modeling the policy of an expert agent, and how this policy can be learned from demonstrations. We also compare PPRs to Finite-State Machines on tasks related to learning the behavior of a robotic wheelchair user.

Techniques for improving the performance of inverse optimal planning algorithms (Chapter 6): We propose to use local soft homomorphisms of Markov Decision Processes, introduced by Sorg and Singh [2009], in order to generalize the demonstration of an expert's policy into new unseen states. We also provide a technique for generalizing the expert's policy by using the learned reward function for generating new examples that cover the complete state space.

A Predictive Reward Representation (Chapter 6): We propose an alternative representation of the reward function in inverse optimal planning. In inverse optimal planning, a reward function is learned from a demonstration, assuming that the reward is a linear combination of basic features. In the predictive reward representation, the reward function is a linear combination of the predictions of the basic features after a sequence of actions. We show that these new features are more likely to be encountered in larger regions of the state space than the basic features. Consequently, the quality of the policies learned using this new representation of rewards can be improved.

1.3 Thesis Outline

This thesis is organized as follows. Chapter 2 presents a background on discrete-time dynamical systems, Markov Decision Processes (MDPs), Partially Observable Markov Decision Processes (POMDPs), Predictive State Representations (PSRs), and an overview of some other models, in addition to basic planning and learning algorithms using these models. Chapter 3 presents a new algorithm for compressing the state space in POMDP by transforming

it into a PSR with a reduced dimension. Chapter 4 introduces the idea of Predictive Policy Representations (PPRs). Novel approximate and exact algorithms for planning in Decentralized POMDPs are also presented in Chapter 4. Chapter 5 introduces a new policy gradient algorithm for reinforcement learning in partially observable environments. Chapter 6 covers different techniques, based on predictive representations, for imitation learning. At the end of each chapter, the proposed solutions are empirically compared to the alternative methods using several simulated domains. Finally, Chapter 7 concludes with a summary of our contribution, and a consideration of future research.

Publication notes

Some of the material from Chapter 2, related to predictive representations with core histories, appeared in [Boularias and Chaib-draa, 2009]. The key results from Chapter 3, related to state space compression, were reported in [Boularias et al., 2008]. Most of the work in Chapter 4, related to the policy space compression, was published in [Boularias and Chaib-draa, 2007], [Boularias and Chaib-draa, 2008a], and [Boularias and Chaib-draa, 2008b]. Most of the material in Chapter 5, related to reinforcement learning with PPRs, was published in [Boularias and Chaib-draa, 2009]. Part of Chapter 6, related to imitation learning in partially observable environments, was presented in [Boularias, 2008], the other part, related to homomorphism of MDPs, appeared in [Boularias and Chaib-draa, 2010].

Chapter 2

Discrete-Time Dynamical Systems

2.1 Introduction

Decision-making under uncertainty can generally be casted as the problem of controlling a *dynamical system*. The concept of dynamical systems is a mathematical formalization of systems that can be described at any given time by a variable called *state*, and the future values of this variable can be predicted by using a fixed evolution rule, called *the transition function*. Examples of dynamical systems range from the mathematical models that describe the working of mechanical devices, the movement of planets around the sun, to those describing the fluctuations of a stock market, or modeling the global climate of Earth.

Mathematical models of dynamical systems are used for two principal tasks. The first one is predicting the state of the system in the future, this prediction is then used for correcting the parameters of the model, or for making decisions related to the system. The second task is the control of the system by performing actions so that the future states will be close or equal to desirable states. Figure 2.1 shows a graphical view of the process of interaction between an *agent* and a dynamical system.

Unfortunately, real-world systems are highly complex and cannot be precisely described by using simple mathematical models. To deal with this complexity, one should make several assumptions about the state space and the transition function of dynamical systems. One of these assumptions is that the system is observed and controlled at discrete steps of time, and the intermediate state of the system between two steps is not relevant for determining future states. The actual time between two steps may be variable and unbounded. This assumption is not really a strong one in practice, since a continuous time-dependant transition function can be approximated by using a sufficiently small discretization of time. The second assumption that we will consider in modeling dynamical systems, though not always necessarily, is that the state space is discrete and finite. By considering this assumption, one can take advantage

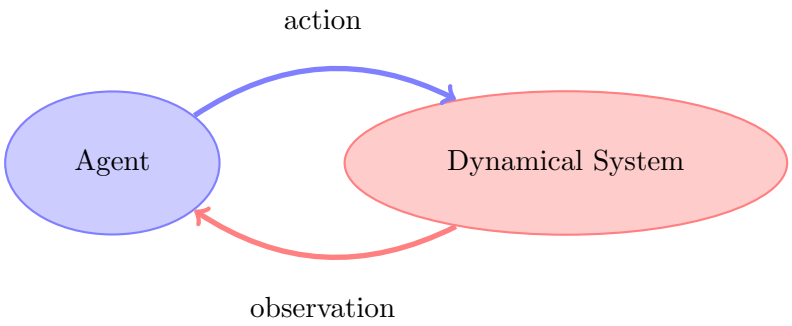


Figure 2.1: The process of interaction between an agent and a dynamical system.

of well-known optimization techniques, such as dynamic programming [Bellman, 1957], in order to find efficient and generic algorithms for controlling dynamical systems. Finally, the most important assumption is known as *Markov* property: the state of the system is a sufficient information for predicting its future states by using the transition function. In practice, the dynamics of a system depends on many variables that cannot be all included in a mathematical model, either for computational reasons or simply because they are unknown, a stochastic transition function is used to overcome the effect of missing information.

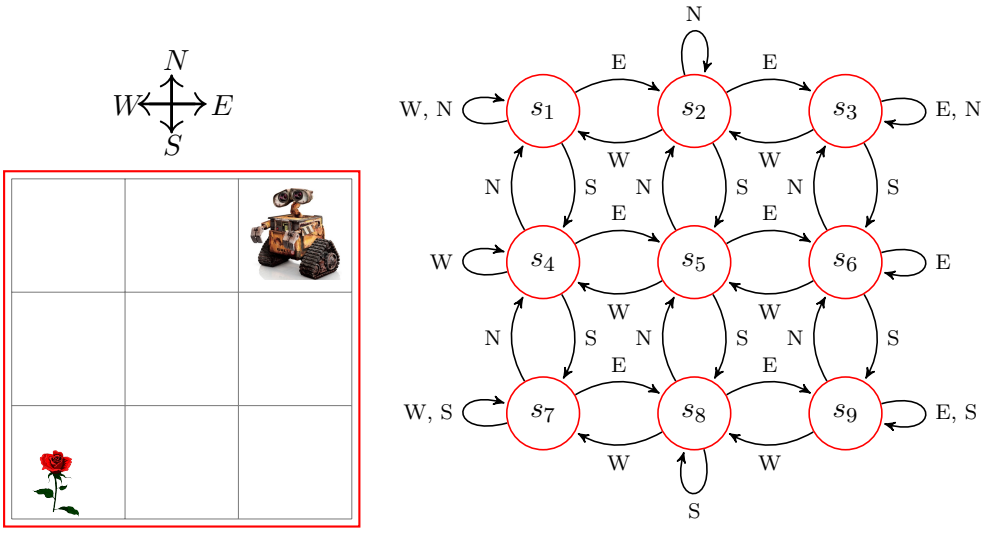
In this thesis, we mainly focus on making decisions with a particular model of dynamical systems known as (Partially Observable) Markov Decision Process (or (PO)MDP), as well as an alternative model known as Predictive State Representation (PSR). An overview of (PO)MDPs, PSRs, and some less known other models is given in this chapter.

2.2 Markov Decision Processes

The simplest model for representing discrete-time dynamical systems is known as Markov Decision Process (MDP) [Markov, 1913] (named after the brilliant Russian mathematician Andrey A. Markov (1856-1922)). Most of the research on MDPs was spawned by the publication of the book *Dynamic Programming and Markov Processes* by R. Howard [Howard, 1960]. This book presented a structured analysis of decision-making problems using the framework of MDPs, and several algorithms, based on Dynamic Programming, for solving these problems. MDPs provide a natural mathematical formalization of many real-world problems. Today, they are used in a variety of areas, including robotics, automated control, manufacturing, games, and economics. A survey of applications of MDPs can be found in [Powell, 2007].

2.2.1 Example

A frequent task of an autonomous mobile robot is to find optimal paths (or plans) for reaching goals while avoiding obstacles and minimizing the required time and the consumption of energy. To decide where to go, the robot should know only its current location, level of energy, and the location of the target. Figure 2.2 shows a simple navigation problem that can be represented as a Markov Decision Process. Of course, one can imagine more complicated tasks, suck as tracking a mobile target, collecting samples, or navigating in a hostile environment using defective actuators. Most of these problems can be easily casted as MDPs.



(a) A simple navigation problem

(b) MDP representation

Figure 2.2: A simple decision-making problem and its MDP formalization. The state of the system corresponds to the location of the robot, the actions correspond to movements between adjacent states, and the goal of the robot is to reach the flower.

2.2.2 Definition

Formally, an MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$. We now describe each one of these components.

State space $\mathcal{S}$: This thesis is intimately related to the notion of state. The definition of a state is recursive: A state is a representation of all the relevant information for predicting future states, in addition to all the information relevant for the related decision problem. In the example of figure 2.2, the state space $\mathcal{S} = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9\}$ corresponds to the set of the robot's locations on the grid. The state space may be finite, countably infinite, or continuous. We will focus on models with a finite set of states. In our example, the states correspond to different positions on a discretized grid.

Action space $\mathcal{A}$: The states of the system are modified by the actions executed by an agent. The goal of the agent is to choose actions that will influence the future of the system so that the more desirable states will occur more frequently. The actions space can be finite, infinite or continuous, but we will consider only the finite case. In our example, the actions of the robot might be move north, move south, move east, move west, or do not move, so $\mathcal{A} = \{N, S, E, W, nothing\}$. In the MDP framework, the state changes only after that an agent executes an action.

Transition function T: When an agent tries to execute an action in a given state, the action does not always lead to the same result, this is due to the fact that the information represented by the state is not sufficient for determining precisely the outcome of the actions. $T(s_t, a_t, s_{t+1})$ returns the probability of transitioning to state s_{t+1} after executing action a_t in state s_t. In our example, the actions can be either deterministic, or stochastic if the floor is slippery, and the robot might ends up in a different position while trying to move toward another one [1].

Reward function R: The preferences of the agent are defined by the reward function R. This function directs the agent towards desirable states and keeps it away from unwanted ones. $R(s_t, a_t)$ returns a reward (or a penalty) to the agent for executing action a_t in state s_t. The goal of the agent is then to choose actions that maximize its cumulated reward. The elegance of the MDP framework comes from the possibility of modeling complex concurrent tasks by simply assigning rewards to the states. In our example, one may consider a reward of $+10$ for reaching the goal state (the flower), a -2 for any movement (consumption of energy), and a -1 for not doing anything (waste of time).

2.2.3 Finite or Infinite Horizons, Discount Factors

Given a reward function, the goal of the agent is to maximize the expected cumulated reward over some number H of steps, called the *horizon*. The horizon can be either finite or infinite. If the horizon is finite, then the optimal actions of the agent will depend not only on the states, but also on the remaining number of steps until the end. In our example, if the agent is at the top right position on the grid, and only two steps are left before the end, then it would better to do nothing and receive an average reward of -1 per step, than to move and receive an average reward of -2 per step, since the goal cannot be reached within two steps. If the horizon is infinite, then the optimal actions depend only on the state. In our case, the optimal action at any step is to move toward the goal. A *discount factor* $\gamma \in [0, 1)$ is also used to indicate how the importance of the earned rewards decreases for every time-step delay. A reward that will be received k time-steps later is scaled down by a factor of γ^k. The discount factor can also be interpreted as the probability that the process continues after any step.

[1] Throughout this thesis, we denote by a_t (respectively s_t) the current action (respectively state) at time step t, and by a^i (respectively s^i) a given action (respectively state) from the set $\mathcal{A}$ (respectively $\mathcal{S}$).

2.2.4 Policies and Value Functions

The agent selects its actions according to a special function called *policy*. A deterministic stationary policy π is a function that maps every state s into an action a. A stochastic stationary policy is a function that maps each state into a probability distribution over the different possible actions. The value function of a policy π is a function V^π that associates to each state the expected sum of rewards that the agent will receive if it starts executing policy π from that state. In other terms:

$$V^\pi(s) = \mathbb{E}_{s_t,a_t}\left[\sum_{t=0}^{H} \gamma^t R(s_t, a_t)|\pi, s_0 = s\right]$$

The value function of a stationary policy can also be recursively defined as:

$$\begin{aligned}
V^\pi(s) &= \mathbb{E}_{s_t,a_t}\left[\sum_{t=0}^{H} \gamma^t R(s_t, a_t)|\pi, s_0 = s\right] \\
&= \mathbb{E}_{s_t,a_t}\left[R(s_0, a_0) + \sum_{t=1}^{H} \gamma^t R(s_t, a_t)|\pi, s_0 = s\right] \\
&= R(s, \pi(s)) + \mathbb{E}_{s_t,a_t}\left[\sum_{t=1}^{H} \gamma^t R(s_t, a_t)|\pi, s_0 = s\right] \\
&= R(s, \pi(s)) + \gamma \mathbb{E}_{s_t,a_t}\left[\sum_{t=0}^{H} \gamma^t R(s_t, a_t)|\pi, s_0 \sim T(s, \pi(s), .)\right] \\
&= R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi(a), s') V^\pi(s')
\end{aligned}$$

where $\pi(s)$ is the action associated to state s.

This latter equation is well-known under the name of *Bellman equation*. An optimal policy π^* is one that satisfies:

$$\forall s \in \mathcal{S} : \pi^* \in \arg\max_\pi V^\pi(s)$$

The value function of an optimal policy is called *the optimal value function*, it is defined as:

$$V^*(s) = \max_{a \in \mathcal{A}}\left[R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') V^*(s')\right]$$

In his seminal work on Dynamic Programming, Bellman [1957] proved that a stationary deterministic optimal policy exists for any discounted infinite horizon MDP. In this thesis, we will mainly focus on finding deterministic optimal policies.

2.2.5 Planning with Markov Decision Processes

Planning in MDPs refers to the problem of finding an optimal (or a near optimal) policy π^* given the model parameters $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$. Before presenting some planning algorithms for MDPs, we will first need to introduce the *Q-value function*. A *Q-value* is the expected sum of rewards that the agent will receive if it executes an action a in a state s then follows a policy π for the remaining steps. Formally, a Q-value function is defined as:

$$Q^{\pi}(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V^{\pi}(s') \tag{2.1}$$

Similarly, the optimal Q-value function is defined as:

$$Q^{*}(s,a) = \max_{a \in \mathcal{A}} \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V^{*}(s') \right]$$

Most of planning algorithms for MDPs fall in one of the two categories: policy iteration and value iteration [Sutton and Barto, 1998]. These two categories are described in the remaining of this subsection.

2.2.5.1 Policy Iteration

This method is depicted in Algorithm 1. It consists in starting with a randomly chosen policy π_t and a random initialization of the corresponding value function V_k, for $k = 0$ and $t = 0$ (Steps 1 to 4), and iteratively repeating the *policy evaluation* and the *policy improvement* operations. Policy evaluation (Steps 6 to 10) consists in calculating the Q-values of policy π_{t+1} by solving the system of linear equations 2.1 for all the states $s \in \mathcal{S}$. An efficient iterative way to solve this equation is to initialize the value function of π_{k+1} with the value function V_k of the previous policy, and then repeat the operation:

$$\forall s \in \mathcal{S} : V_{k+1}(s) = R(s, \pi_t(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi_t(s), s') V_k(s')$$

until $\forall s \in \mathcal{S} : |V_k(s) - V_{k-1}(s)| < \epsilon$, for a predefined error threshold ϵ.
Policy improvement (Steps 11 to 13) consists in finding the *greedy* policy π_{t+1} given the value function V_k:

$$\forall s \in \mathcal{S} : \pi_{t+1}(s) = \arg\max_{a \in \mathcal{A}} \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V_k(s') \right]$$

This process stops when $\pi_t = \pi_{t-1}$, in which case π^t is an optimal policy, i.e. $\pi_t = \pi^*$.
The main drawback of policy iteration is that a complete policy evaluation is involved in each iteration. Value iteration consists in overlapping the evaluation and improvement steps.

Input: An MDP model $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$;
1 /* Initialization */ ;
2 $t = 0$, $k = 0$;
3 $\forall s \in \mathcal{S}$: Initialize $\pi_t(s)$ with an arbitrary action;
4 $\forall s \in \mathcal{S}$: Initialize $V_k(s)$ with an arbitrary value;
5 **repeat**
6 /* Policy evaluation */;
7 **repeat**
8 $\forall s \in \mathcal{S} : V_{k+1}(s) = R(s, \pi_t(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi_t(s), s') V_k(s')$;
9 $k \leftarrow k + 1$;
10 **until** $\forall s \in \mathcal{S} : |V_k(s) - V_{k-1}(s)| < \epsilon$;
11 /* Policy improvement */;
12 $\forall s \in \mathcal{S} : \pi_{t+1}(s) = \arg\max_{a \in \mathcal{A}} \left[R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') V_k(s') \right]$;
13 $t \leftarrow t + 1$;
14 **until** $\pi_t = \pi_{t-1}$;
15 $\pi^* = \pi_t$;
Output: An optimal policy π^*;

Algorithm 1: The policy iteration algorithm.

2.2.5.2 Value Iteration

Value iteration, described by Algorithm 2, can be written as a simple backup operation:

$$\forall s \in \mathcal{S} : V_{k+1}(s) = \max_{a \in \mathcal{A}} \left[R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') V_k(s') \right]$$

This operation is repeated until $\forall s \in \mathcal{S} : |V_k(s) - V_{k-1}(s)| < \epsilon$ (Steps 3 to 6), in which case the optimal policy is simply the greedy policy with respect to the value function V_k (Step 7).

Input: An MDP model $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$;
1 $k = 0$;
2 $\forall s \in \mathcal{S}$: Initialize $V_k(s)$ with an arbitrary value;
3 **repeat**
4 $\forall s \in \mathcal{S} : V_{k+1}(s) = \max_{a \in \mathcal{A}} \left[R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') V_k(s') \right]$;
5 $k \leftarrow k + 1$;
6 **until** $\forall s \in \mathcal{S} : |V_k(s) - V_{k-1}(s)| < \epsilon$;
7 $\forall s \in \mathcal{S} : \pi^*(s) = \arg\max_{a \in \mathcal{A}} \left[R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') V_k(s') \right]$;
Output: An optimal policy π^*;

Algorithm 2: The value iteration algorithm.

Both policy iteration and value iteration algorithms update the values of the different states in an arbitrary order, using the values of the previous iteration. Therefore, most of the planning time is wasted in propagating zero values if the reward function is sparse, which is the case in most systems with a final goal state, such as the example of Figure 2.2. An interesting solution to this problem is called *prioritized sweeping* [Wingate and Seppi, 2005; Moore and Atkeson, 1993], where the priority of updating the value of a state is related to the change that occurred in the values of its immediate neighbors.

2.2.6 Learning with Markov Decision Processes

Learning with MDPs generally refers to the problem of finding an optimal policy π^* for an MDP with unknown transition function T; the reward function R also can be unknown. This problem is better known under the term of *Reinforcement Learning* (RL), or also *Approximate Dynamic Programming* (ADP). RL algorithms can be regrouped in three main categories: model-based algorithms, bayesian RL algorithms, and model-free algorithms. In the first category, the agent learns the unknown parameters by interacting with the environment and perceiving the outcome of every executed action. The agent uses a planning algorithm after a period of time to find an optimal policy. Examples of model-based algorithms include E^3 [Kearns and Singh, 2002] and R-MAX [Brafman and Tennenholtz, 2003].

In the second category, bayesian RL algorithms, the uncertainty about the model parameters is itself regarded as a parameter of the model. At any moment, the state of the agent is a probability distribution on all the possible models, in addition to its state in the original model. The initial distribution over unknown parameters is called a *prior*, and it is sequentially updated by using Bayes' Rule, where the evidence correspond to the perceived transitions and rewards. Examples of Bayesian RL techniques can be found in [Ross and Pineau, 2008; Ross et al., 2008a; Ghavamzadeh and Engel, 2007].

In model-free RL algorithms, an optimal policy is directly learned by interacting with the environment, without attempting to learn the unknown parameters. A popular example of theses algorithms, described in Algorithm 3, is Q-learning [Watkins, 1989]. In its simplest form, Q-learning starts by initializing the Q-values with any randomly chosen values. After executing action a in state s, transiting to state s', and perceiving reward r, the Q-value $Q^{t+1}(s,a)$ is calculated by using the following Temporal-Difference (TD) rule [Sutton and Barto, 1998]:

$$Q^{t+1}(s,a) = (1-\alpha_t)Q^t(s,a) + \alpha_t\left[R(s,a) + \gamma \max_{a' \in \mathcal{A}} Q^t(s',a')\right]$$

where $\alpha_t \in [0,1]$, the other Q-values remain unchanged.

These Q-values converge to the optimal ones under the following conditions (known as

Robbins-Monro conditions):

$$\sum_{t=0}^{\infty} \alpha_t = \infty, \quad \sum_{t=0}^{\infty} \alpha_t^2 < \infty$$

A recent survey of reinforcement learning algorithms can be found in [Szepesvári, 2009].

Input: An MDP model $\langle \mathcal{S}, \mathcal{A}, R \rangle$ with unknown transition function;
$t = 0$, s_0 is an initial state;
$\forall s \in \mathcal{S}, \forall a \in \mathcal{A}$: Initialize $Q^t(s, a)$ with an arbitrary value;
repeat
  $\pi(s_t) = \arg\max_{a \in \mathcal{A}} Q^t(s_t, a)$;
  Choose action a_t corresponding to $\pi(s_t)$ with probability $1 - \epsilon_t$, and a random action a_t with probability ϵ_t;
  Execute action a_t and observe the received reward $R(s_t, a_t)$ and the next state s_{t+1};
  $Q^{t+1}(s_t, a_t) = (1 - \alpha_t) Q^t(s_t, a_t) + \alpha_t \left[R(s_t, a_t) + \gamma \max_{a' \in \mathcal{A}} Q^t(s_{t+1}, a') \right]$;
  $t \leftarrow t + 1$;
until *the end of learning* ;
Output: A learned policy π;

Algorithm 3: The Q-learning algorithm.

Learning with MDPs may also refer to the problem of recovering an optimal policy from examples of actions provided by an expert, this type of learning is known as *imitation learning*. In this context, one can generally distinguish between direct and undirect imitation approaches [Ratliff et al., 2009]. In direct methods, the agent learns a function that maps states into actions by using a supervised learning technique [Atkeson and Schaal, 1997]. The best known example of a system built on this paradigm is ALVINN [Pomerleau, 1989], where a neural network was trained to learn a mapping between a road image and a vehicle steering action. Despite the remarkable success of the ALVINN system and others, direct methods suffer from a serious drawback: they can learn only reactive policies, where the optimal action of a state depends only on its features, regardless of the future states of the system. To overcome this drawback, Ng and Russell [2000] introduced a new approach of undirect imitation learning known as Inverse Reinforcement Learning (inverse RL). The aim of inverse RL is to recover a reward function under which the expert's actions are optimal, rather than to directly mimic the actions of the expert. The learned reward is then used to find an optimal policy. Contrary to direct methods, inverse RL takes into account the fact that the different states of the system are interrelated by transition and value functions. Consequently, the expert's actions can be predicted in states that were not encountered during the demonstration. Imitation learning will be discussed with more details in Chapter 6.

A major restriction of finite-state MDPs is that the agent must at any moment know exactly the state of the system. Unfortunately, this assumption often does not hold in practice. In fact, the agent observes the world through limited and imperfect receptors, and the information contained in the perceptions is not sufficient for determining the state. In our previous

example depicted in Figure 2.2, the robot's sensors may be limited to only distinguishing between states where the robot faces a wall and the other states. *Partially Observable Markov Decision Processes* (POMDPs), introduced in the next section, provide a powerful tool for representing such systems.

2.3 Partially Observable Markov Decision Processes

2.3.1 Definition

Formally, a POMDP [Smallwood and Sondik, 1971] is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R \rangle$ where $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$ is an MDP, $\mathcal{O}$ is a set of observations (perceptions), and Z is an observation function: $Z(o, s, a)$ is the probability of observing $o \in \mathcal{O}$ if the state of the system is s and the action that led to this state is a.

The observations can be aliased (the same observation may be observed in different states) and stochastic (different observations may be observed in the same state). Consequently, the state of the system cannot be determined from the observations. Instead, an observation can be seen as an evidence about the state. The agent's belief about the hidden state is a probability distribution on all the possible states, called *the belief state* [2]:

$$b_t = [Pr(s_t = s^0), Pr(s_t = s^1), \ldots, Pr(s_t = s^{|\mathcal{S}|-1})]^T$$

The agent starts with an initial belief state b_0, and every time an action a_t is executed and an observation o_{t+1} is received, the belief state b_t is updated by using Bayes' Rule:

$$\begin{aligned} b_{t+1}(s) &= Pr(s_{t+1} = s | b_t, a_t, o_{t+1}) \\ &= \frac{Pr(s_{t+1} = s, o_{t+1} | b_t, a_t)}{Pr(o_{t+1} | b_t, a_t)} \\ &= \frac{\sum_{s' \in S} b_t(s') T(s', a_t, s) Z(o_{t+1}, s, a_t)}{\sum_{s' \in S} \sum_{s'' \in S} b_t(s') T(s', a_t, s'') Z(o_{t+1}, s'', a_t)} \end{aligned} \tag{2.2}$$

Therefore, the belief state b_{t+1} is a nonlinear function of the previous belief state b_t, action a_t and observation o_{t+1}:

$$b_{t+1} = \tau(b_t, a_t, o_{t+1})$$

The belief state b_t at time-step t allows us to calculate the probability of any observation o at time-step $t+1$ as:

$$Pr(o_{t+1} = o | b_t, a_t) = \sum_{s \in \mathcal{S}} \sum_{s' \in \mathcal{S}} b_t(s) T(s, a_t, s') Z(o, s', a_t)$$

[2] Through this thesis, a vector b corresponds to a column, and b^T corresponds to its transpose.

The expected reward of executing action a for a belief state b_t is given by:

$$R(a|b_t) = \sum_{s \in S} b_t(s) R(s, a)$$

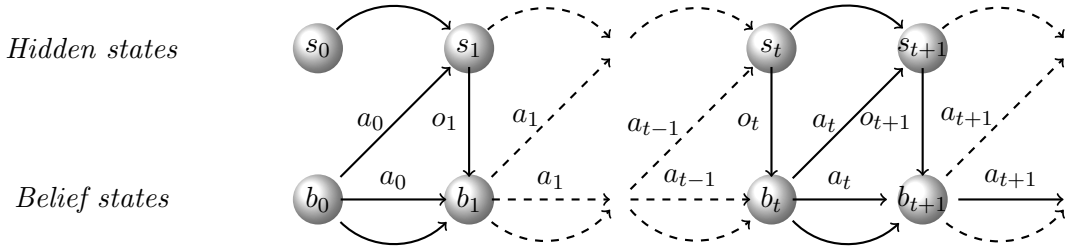


Figure 2.3: A Partially Observable Markov Decision Process, adapted from [Pineau, 2004].

Figure 2.3 illustrates the temporal series of belief states. The Markov property implies that the belief state and the full history of the system (sequence of actions and observations) contain exactly the same information about the current state, and consequently, about all the future events.

2.3.2 Planning with Partially Observable Markov Decision Processes

Planning with a POMDP model consists in finding for every possible belief state an optimal action, i.e. an action that maximizes the expected cumulated reward. This is a robust way of planning that takes into account the uncertainty about the hidden state. However, planning with POMDPs is known to be a very hard problem. In fact, finding an optimal policy for a finite horizon POMDP is proven to be a PSPACE-complete problem [Papadimitriou and Tsitsiklis, 1987], and verifying the existence of a policy with an expected value greater than a given threshold ϵ for an infinite horizon POMDP is undecidable [Madani et al., 1999]. Even finding a near-optimal policy (a policy with a bounded value loss compared to the optimal one) for a finite horizon POMDP is NP-hard [Lusena et al., 2001].

The main factor behind the poor scalability of POMDP planning algorithms is known as the *curse of dimensionality* [Bellman, 1957]. In fact, the number of possible belief states grows double-exponentially with respect to the number of observations and the length of the horizon. Numerous algorithms (both exact and approximate) have been proposed for planning in POMDPs. Examples include a variant of value iteration [Smallwood and Sondik, 1971], point-based approaches [Paquet et al., 2005; Pineau et al., 2003; Smith and Simmons, 2004], structure-based approaches [Bonet and Geffner, 2003; Dai and Goldsmith, 2007; Dibangoye et al., 2009b], compression-based approaches [Roy et al., 2005; Poupart and Boutilier, 2002; Li et al., 2007; Lee and Seung, 2000], and forward search approaches [Paquet, 2006; Ross

et al., 2008b]. The issue of planning with POMDPs will be discussed with more details in Chapter 3.

2.3.3 Learning with Partially Observable Markov Decision Processes

There are two major families of learning algorithms that can be used for finding the optimal parameters of a policy for a given POMDP. In the first family, a *value function* is used for estimating the expected long-term reward of each action for every belief state. The parameters of the policy are adjusted so that in each belief state, the actions with the highest estimated long-term reward will be executed more frequently. In the second family, the optimal policy is gradually learned by *searching* in the space of the policy parameters. These parameters are updated according to the immediate reward after executing each action, or according to the cumulated reward at the end of each trial. This family includes, for instance, most policy gradient algorithms, such as GAPS [Peshkin, 2001] or GPOMDP [Baxter and Bartlett, 2000]. Actor-critic algorithms [Sutton et al., 2000; Peters and Schaal, 2006; Aberdeen et al., 2007; Wierstra et al., 2009] combine the advantages of both value-function and policy search methods. The actor part maintains a parametric policy that is gradually improved according to the evaluation provided by the critic part. The critic learns a value function where the variables correspond to the policy parameters. The problem of reinforcement learning in POMDPs is discussed in chapter 5.

A special class of POMDPs, known as *Hidden Markov Model* (HMM) [Rabiner and Juang., 1986], is used for representing uncontrolled systems, with no actions or rewards. An HMM is a POMDP where the transition function depends only on the state of the system and not on the actions executed by the agent. Therefore, HMMs are used for recognition tasks rather than decision-making. HMMs have many successful applications in machine learning and temporal pattern recognition such as speech, handwriting, and gesture recognition [Yang et al., 1997].

There are two major issues related to POMDPs. First, finding optimal actions is computationally intractable. In fact, existing techniques for finding exact optimal policies for POMDPs typically cannot handle problems with more than a few thousands states [Paquet et al., 2005; Virin et al., 2007; Ross et al., 2008b; Amato et al., 2007a; Prentice and Roy, 2009]. This is principally due to the way the states are specified in POMDPs: a state is an instantiation of all the possible variables in the system, whereas only a small number of variables might be needed for predicting the future states of the system. The second issue is learning unknown parameters in POMDPs. The parameters of a POMDP model (or a POMDP policy) are a function of the hidden states, which cannot be directly observed in the training data, they can be efficiently estimated only when a prior knowledge about the structure of the system is available.

2.4 Predictive State Representations

In this section, we present an alternative model, known as *Predictive State Representations* (PSRs), where the parameters are a function of completely observable variables, and no hidden variables are used in the specification of the model.

2.4.1 Definition

Contrary to POMDPs, where the state is represented by hidden variables, PSRs [Littman et al., 2001; Singh et al., 2004] are based on testable experiences. The notion of *tests*, used in the definition of PSRs, carries the central idea of relating states to verifiable and observable quantities. A test q is an ordered finite sequence of action-observation pairs $a_0o_1...a_{k-1}o_k$. The *prediction* of a test q, is the probability of perceiving the sequence of observations $o_1, \ldots, o_k$ if the agent executes the sequence of actions $a_0, \ldots, a_{k-1}$. The probability of a test q starting after a history h_t is defined by:

$$Pr(q|h_t) \stackrel{def}{=} Pr(q^o|h_t, q^a) = Pr(o_{t+1} = o^1, \ldots, o_{t+k} = o^k|h_t, a_t = a^0, \ldots, a_{t+k-1} = a^{k-1})$$

where q^a denotes the sequence of actions in q and q^o denotes the sequence of observations. Through this thesis, when both superscripts and subscripts indices are used in the same equation, superscripts indices (such as a^k and o^k) are used for distinguishing between the elements of a given a set, and subscripts indices (such as a_t and o_t) are used for indicating time-steps.

Planning and learning in dynamical systems are basically problems of predicting the expected value of a random variable (such as a reward), or the probability of an event (such as an observation), after a given sequence of actions. In POMDPs for example, the belief state at a given moment is used only for predicting the future rewards and observations, and choosing actions accordingly. Rewards can be considered as part of the observations in a certain way. Therefore, the principal function of a POMDP model is predicting the probabilities of tests, the belief states can then be replaced by these probabilities. However, the length of tests grows infinitely, and their number quickly becomes higher than the number of hidden states. Fortunately, it has been proved by Littman et al. [2001] that the probability of any test depends only on the probabilities of a finite subset of tests called *core tests* and denoted by $Q = \{q_1, q_2, \ldots, q_k\}$ (Figure 2.5). The prediction of core tests, $Pr(Q|h) = [P(q_1|h), P(q_2|h), \ldots, P(q_k|h)]^T$, forms a sufficient statistic for the system after an arbitrary h, and can be used as the belief state. In other terms, the prediction for any test q after any history h can be calculated based on $Pr(Q|h)$:

$$Pr(q|h) = f_q(Pr(Q|h))$$

where $f_q : [0,1]^{|Q|} \rightarrow [0,1]$. It is important to note that f_q, which is called *projection function*, is independent of the history h.

Assume in our previous example of Figure 2.2 that the robot can only sense the presence of an obstacle when it tries to move beyond a wall. To simplify furthermore the example, assume that the actions and observations are deterministic. At a given time-step, the belief state of the robot using a POMDP model is a probability distribution over the nine hidden states (positions on the grid). Using a PSR model, the belief state becomes the probabilities of the following four core tests: {North Obstacle, South Obstacle, East Obstacle, West Obstacle} (the form of these tests is: *action observation*, the first part is a direction of a movement and the second part is a resulting observation). The probabilities of these tests can be used to calculate the underlying state distribution, and then to calculate the probability of any test. For instance, if $\{Pr(\text{North Obstacle}|h) = 0, Pr(\text{South Obstacle}|h) = 0, Pr(\text{East Obstacle}|h) = 0, Pr(\text{West Obstacle}|h) = 0\}$, then certainly $Pr(s_5|h) = 1$ (s_5 being the middle position). Also, if $\{Pr(\text{North Obstacle}|h) = 1, Pr(\text{South Obstacle}|h) = 0, Pr(\text{East Obstacle}|h) = 1, Pr(\text{West Obstacle}|h) = 0\}$, then certainly $Pr(s_3|h) = 1$ (s_3 is the upper right position).

After executing an action a and perceiving an observation o, the probability of a core test q is updated by using Bayes' Rule:

$$Pr(q|hao) = \frac{Pr(aoq|h)}{Pr(ao|h)} = \frac{f_{aoq}(Pr(Q|h))}{f_{ao}(Pr(Q|h))} \tag{2.3}$$

The parameters of the model correspond to the projection function of the set of *extension tests*: $\{aoq_i, ao|\forall q_i \in Q, a \in A, o \in O\}$. The number of extension tests, is proportional to the number of core tests, $|Q|$, which is called the *dimension* of the model. The PSR representation of a dynamical system has at most a number of core tests equal to the number of hidden states in the equivalent POMDP representation [Littman et al., 2001]. In fact, the PSR model is potentially more compact than the corresponding POMDP, as in the previous example.

Nearly all the PSR literature is devoted to a special class of PSRs where the projection function is linear. If we denote the projection function by a vector m_q, then for any test q :

$$Pr(q|h) = Pr(Q|h)^T m_q$$

A linear PSR model is a tuple $\langle \mathcal{A}, \mathcal{O}, Q, \{m_{ao}\}, \{m_{aoq_i}\}\rangle$, where $\mathcal{A}$ and $\mathcal{O}$ are action and observation sets, Q is a finite set of core tests $\{q_1, q_2, \ldots, q_k\}$, m_{ao} are weight vectors for one-step tests, defined for each action $a \in \mathcal{A}$ and observation $o \in \mathcal{O}$, and m_{aoq_i} are weight vectors for one-step extensions of core tests, defined for each action $a \in \mathcal{A}$, observation $o \in \mathcal{O}$ and core test $q_i \in Q$. In the PSR literature, the immediate reward is usually considered as part of the observation [James et al., 2004; Izadi, 2007; Rudary, 2008; Wingate, 2008; Wolfe, 2009].

The matrix containing the predictions of core tests given the hidden states is called the the outcome matrix U (a $|S| \times |Q|$ matrix, Figure 2.4), which is defined as $U(s, q) = Pr(q|s)$. This matrix can be found by searching for a maximum family of linearly independent tests. Littman et al. [2001] presented a polynomial time algorithm that incrementally finds all core tests in an iterative fashion, given a POMDP model. Belief states in PSRs, denoted by $\tilde{b}_t$, are

linked to the belief states b_t in POMDPs through the definition of the matrix U:

$$\tilde{b} \stackrel{def}{=} Pr(Q|h) = U^T b$$

$$\begin{array}{c} \\ s_1 \\ \vdots \\ s_i \\ \vdots \end{array} \begin{array}{c} \begin{array}{cccc} q_1 & \cdots & q_j & \cdots \end{array} \\ \begin{pmatrix} \cdot & \cdots & \cdot & \cdots \\ \vdots & \vdots & \vdots & \vdots \\ \cdot & \cdots & Pr(q_i|s_i) & \cdots \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix} \end{array}$$

Figure 2.4: The outcome matrix U.

Histories also can be used as a sufficient statistics for predicting future tests, as depicted in Figure 2.6. In fact, James [2005] showed that for any finite-state POMDPs with $|\mathcal{S}|$ states, there is a finite set $\mathcal{H}$ of *core histories* $\{h^1, h^2, \ldots h^{|\mathcal{H}|}\}$, such that:

$$\forall h, q \in \{\mathcal{A} \times \mathcal{O}\}^* : Pr(q|h_t) = \sum_{i=0}^{|\mathcal{H}|} \alpha_t^i Pr(q|h^i) \tag{2.4}$$

In other terms, the probability of any test q after a history h_t depends on the probabilities of the same test after the different core histories. Although the existence of a finite set of core histories has been noticed before [James, 2005], no PSR model based on core histories has been proposed before in the literature. The remaining of this section will be devoted to the description of PSRs based on core histories.

The PSR belief state[3] is a vector b_t, where $\tilde{b}_t(h)$ is the weight of the core history $h \in \mathcal{H}$ in the current history h_t. The probability of any test q after a history h_t is given by:

$$Pr(q^o|h_t, q^a) = \sum_{h \in \mathcal{H}} \tilde{b}_t(h) Pr(q^o|h, q^a) = \tilde{b}_t^T m_q \tag{2.5}$$

where $m^q(h) \stackrel{def}{=} Pr(q^o|h, q^a)$ is independent of h_t, and $\tilde{b}_t(h)$ is independent of q. The belief state $\tilde{b}_t \in \mathbb{R}^{\mathcal{H}}$ is a vector of real-valued weights and not probabilities.

After executing an action a and receiving an observation o, the PSR belief is updated by

[3]The same notation is used for different types of belief state; the interpretation depends on the model.

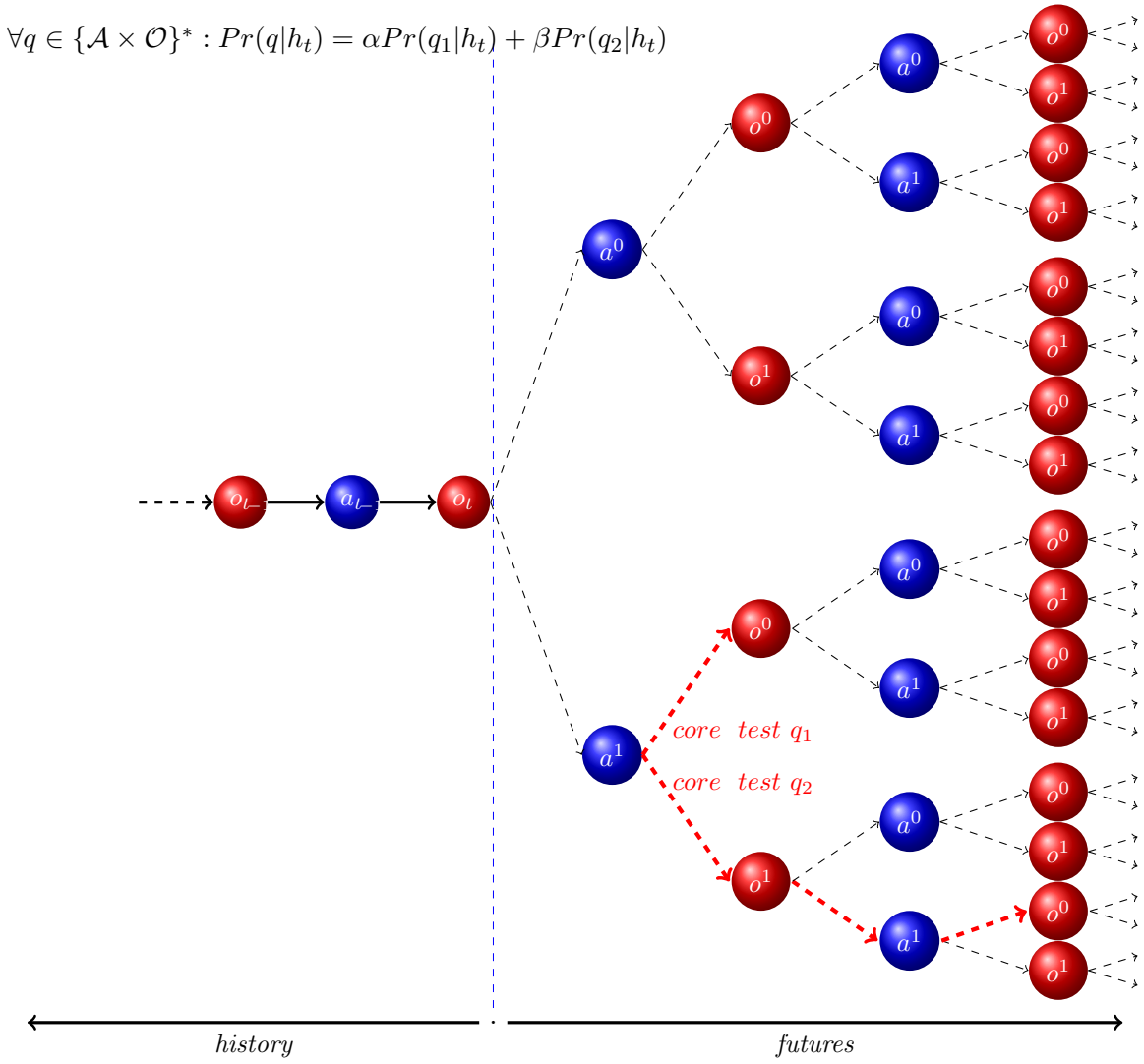


Figure 2.5: A Predictive State Representation with linear core tests

Bayes' Rule:

$$
\begin{aligned}
Pr(q^o|h_t, a, o, q^a) &= \frac{Pr(oq^o|h_t, a, q^a)}{Pr(o|h_t, a)} \\
&= \frac{\sum_{h \in \mathcal{H}} \tilde{b}_t(h) Pr(oq^o|h, a, q^a)}{\sum_{h \in \mathcal{H}} \tilde{b}_t(h) Pr(o|h, a)} \\
&= \frac{\sum_{h \in \mathcal{H}} \tilde{b}_t(h) Pr(o|h, a) Pr(q^o|h, a, o, q^a)}{\sum_{h \in \mathcal{H}} \tilde{b}_t(h) Pr(o|h, a)} \\
&= \frac{\sum_{h \in \mathcal{H}} \tilde{b}_t(h) m_{ao}(h) \sum_{h' \in \mathcal{H}} \tilde{b}_{hao}(h') m_q(h')}{\sum_{h \in \mathcal{H}} \tilde{b}_t(h) m_{ao}(h)} \\
&= \sum_{h \in \mathcal{H}} \tilde{b}_{t+1}(h) m_q(h) = \tilde{b}_{t+1}^T m_q
\end{aligned}
$$

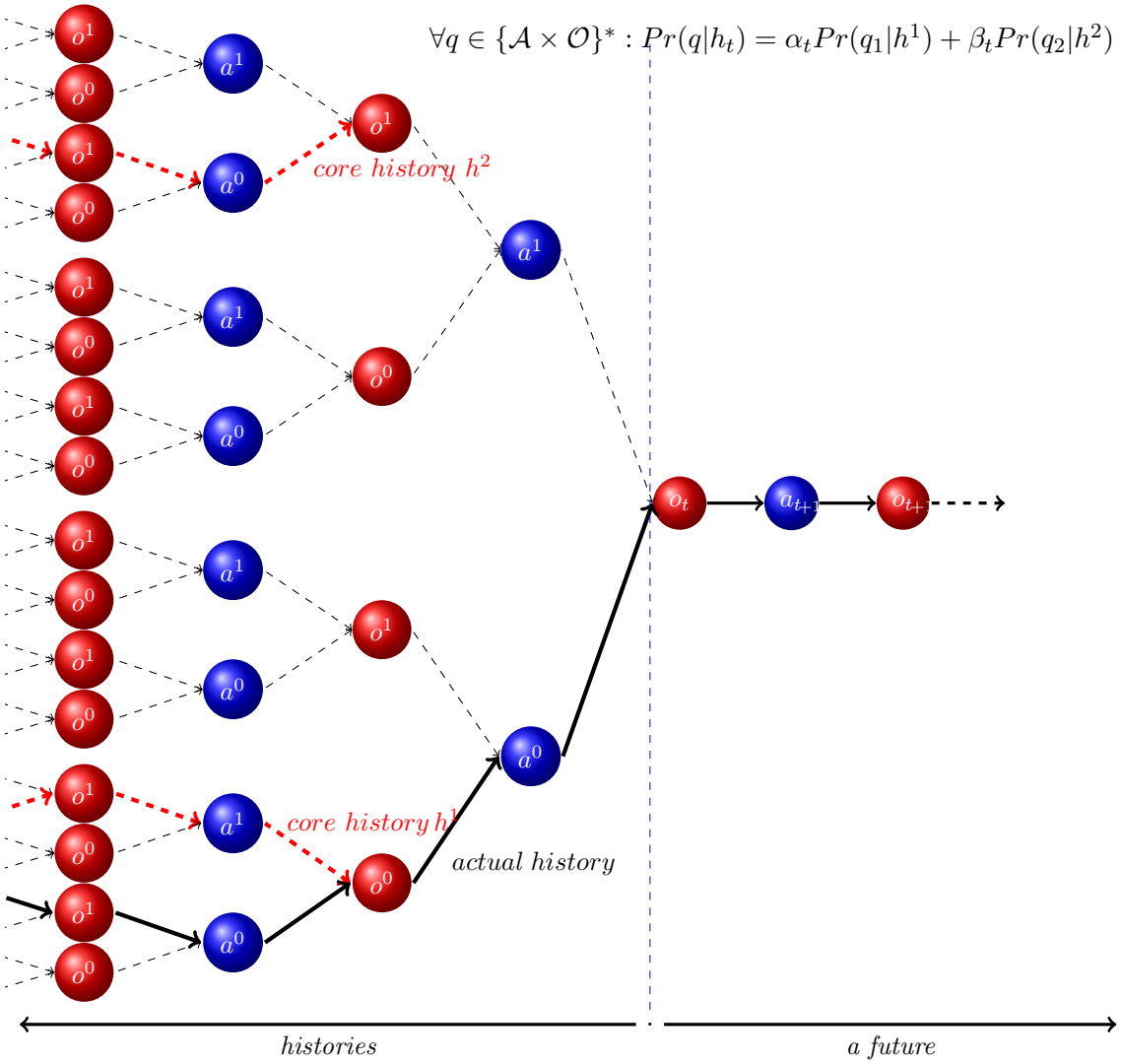


Figure 2.6: A Predictive State Representation with linear core histories

where

$$\tilde{b}_{t+1}(h) = \frac{\sum_{h' \in \mathcal{H}} \tilde{b}_t(h') \tilde{b}_{h'ao}(h) m_{ao}(h')}{\sum_{h' \in \mathcal{H}} \tilde{b}_t(h') m_{ao}(h')} \tag{2.6}$$

The parameters of a PSR based on core histories are then: $\mathcal{A}$, $\mathcal{O}$, the set of core histories $\mathcal{H}$, and the vectors m_{ao} and $\tilde{b}_{hao}$, $\forall a \in \mathcal{A}, o \in \mathcal{O}, h \in \mathcal{H}$. The vector $\tilde{b}_{hao}$ denotes the belief state at the history hao. Notice that the vectors m_{ao} define a probability distribution over the observations o for each action a and core history h, thus, they can be represented by a *softmax function*:

$$m_{ao}(h) = \frac{e^{T\theta^{ha}(o)}}{\sum_{o' \in \mathcal{O}} e^{T\theta^{ha}(o')}}$$

where θ^{ha} is any real-valued function defined on $\mathcal{O}$, and $T \in \mathbb{R}$.

Notice that this latter property (the fact that the parameters of a PSR with core histories are distributions of probabilities) is not satisfied in the parameters of PSRs with core tests. In fact, the weight vector m_{ao} with core tests can take any real value. On the other hand, the belief state with core histories is a vector of real values, whereas with core tests, the belief state is a vector of probabilities, although these probabilities do not necessarily sum to one. Besides this difference, core test representations are basically equivalent to core history ones.

2.4.2 Planning with Predictive State Representations

As for POMDPs, planning algorithms for PSRs aim to find a policy that maps every belief state into an action that maximizes the expected future rewards. Except for the definition and interpretation of belief states, planning with PSRs can be done in a very similar way to POMDPs, and most of planning algorithms for POMDPs can easily be transferred to PSRs after a little change. The first change that should be made is the inclusion of rewards in the observations, the extended set of observations becomes $\mathcal{O}' = \mathcal{O} \times \mathcal{R}$, where $\mathcal{R}$ is the set of all possible rewards [Izadi, 2007]. Since the rewards are real-valued, considering the set of all possible rewards may not be the optimal way of including rewards in the observations. This issue will be furthermore discussed in Chapter 3.

The first method for planning with PSRs was a policy iteration algorithm [Izadi and Precup, 2003]. The expected value of each action at the end of every possible test was evaluated, and the optimal actions selected accordingly. The main drawback of this algorithm is that the number of possible tests grows exponentially with respect to the number of actions and observations, and the core tests were not considered in the formulation. Shortly later, another exact algorithm was proposed by James et al. [2004]. This algorithm is basically an adaptation of another algorithm for POMDPs known as Incremental Pruning [Cassandra et al., 1997]. James et al. [2004] principally showed that, as for POMDPs, the value function of a policy is linear in the belief space (the probabilities of core tests).

More recently, a look-ahead planning method for PSRs was introduced by Izadi [2007], which is close to forward-search methods for POMDPs [Paquet et al., 2005]. Izadi [2007] also investigated the use of core beliefs to reduce the number of beliefs in Point-Based Value Iteration (PBVI) [Pineau et al., 2003]. A recent framework including an approach for learning approximate PSRs, and a point-based algorithm for planning with approximate PSRs was introduced by Boots et al. [2010]. This latter algorithm is basically an adaptation of PBVI [Pineau et al., 2003].

In general, we notice a lack of interest in planning with PSRs in the literature. This is due to the fact that, as mentioned above, planning with PSRs is virtually equivalent to planning with POMDPs, or any other model based on beliefs. The more important issue that should be considered is rather how to find a compact, possibly approximate, PSR model given

a POMDP. A theoretical analysis of the prediction error introduced by the use of compact PSRs was provided by Wolfe [2009] for the case of factored representations (where each state is a tuple of several variables). This issue, known as *dimensionality reduction*, will be addressed in Chapters 3 and 4.

2.4.3 Learning with Predictive State Representations

Most of the PSR literature is devoted to the problem of learning a model from a sequence of actions and observations. Learning a PSR model can be divided into two parts: *discovery* of a set of core tests, and *learning* the parameters corresponding to the core tests (the vectors m^{ao} and m^{aoq}). The first algorithm for learning PSRs focused only on learning the parameters, and assumed that the set of core tests was given [Singh et al., 2003]. The authors defined an error function corresponding to the squared difference between the probabilities of future tests according to the PSR model and the actual probabilities of the same tests. A stochastic gradient-descent was then used to update the parameters at each time-step and to gradually converge to a local optimum.

For their part, James and Singh [2004] proposed the Analytical Discovery and Learning (ADL) algorithm. It was the first algorithm that considered the problem of discovering a set of core tests. To address that, the authors constructed a large matrix D, known as the dynamics matrix, that captures all the dynamics of the system [Singh et al., 2004]. The rows of the dynamics matrix correspond to the histories and the columns correspond to the future tests, and the entries of the matrix are the conditional success probabilities of the tests after each possible history, i.e. $D(h, q) = Pr(q|h)$. This matrix can be directly estimated from the training data if the system can be reset to an initial distribution at the end of each trial. The core tests are the tests corresponding to linearly independent columns of the matrix D. Rosencrantz et al. [2004] proposed a learning algorithm close to ADL, where the dimensionality of the model was reduced even more by performing a Singular Value Decomposition (SVD) on the dynamics matrix. An analysis of ADL-style algorithms is given in [Wiewiora, 2005].

Resetting the system into an initial state distribution is not always possible in practice. To solve this problem, Wolfe [2009] proposed an algorithm for learning the parameters of a PSR model from a single trajectory of actions and observations. Another solution to the problem of learning PSRs in dynamical systems without reset was given by McCracken and Bowling [2006]. As in [Singh et al., 2003], a gradient-descent method was used to update the PSR parameters, while a dynamics matrix was learned and used for discovering core tests, as done in ADL [James and Singh, 2004]. Moreover, a set of constraints was used to restrict the entries of the dynamics matrix to be valid probabilities. A family of simple sufficient, but not necessary, constraints for defining valid vectors m^{ao} and m^{aoq} was given by Wolfe [2010b].

Finally, the problem of learning with *non-blind* (non-uniform) policies was considered

in [Bowling et al., 2006]. We notice that most of the recent advancements on learning PSRs have been focused on models with continuous states and observations, such as Linear Gaussian Models [Rudary, 2008; Wingate, 2008; Boots et al., 2010]. We will return to the problem of learning PSRs in Chapter 5.

2.4.3.1 Comparing PSRs to POMDPs

The main advantage of POMDPs over PSRs is the simplicity of the model description and the straightforward interpretation of the belief state. Another advantage is the use of only probability distributions in the parameters and the belief state of POMDPs, so that the belief remains a distribution after an arbitrary number of bayesian updates. In PSRs, the probabilities in the belief state are not necessarily a distribution, and the parameters used in the belief update are not even probabilities. Consequently, there are no known sufficient and necessary constraints ensuring that the probabilities in the belief state remain in $[0, 1]$ after an arbitrary number of updates [Wolfe, 2010b].

There are two *potential* advantages of PSRs over POMDPs: First, PSRs are event-directed models, so they provide a basic compression of POMDPs by regrouping together states which have the same outcome (*trace equivalence*), and states which have linearly dependent vectors (transition and reward vectors). Indeed, for every belief state in a POMDP, there is a unique corresponding belief state in the equivalent PSR, but the contrary is not always true, a PSR belief state may correspond to an infinite number of POMDP belief states. Second, PSRs are generally easier to learn than POMDPs (or HMMs) since their parameters are function of observable quantities [James and Singh, 2004]. In fact, to calculate the probability of a core test q at history h, we should only repeat h several times and count the number of times q occurs after h. But to calculate the probability of a state s at history h, even if we repeat h several times, we should use the parameters of the POMDP to estimate $Pr(s|h)$ since we cannot observe s. The POMDP parameters cannot be known during the learning process. Hence, POMDP learning algorithms start with random parameters, and then alternate between the phase of estimating the belief states b_i at each step i of the training sequence and the phase of optimizing the POMDP parameters [Koenig and Simmons, 1996]. Consequently, the result depends heavily on the initial parameters, and is subject to local optima.

2.5 Other Frameworks for Modeling Discrete-Time Dynamical Systems

In this section, we will briefly review some other models for representing discrete-time dynamical systems, and discuss their advantages and limitations.

2.5.1 History-based Models

Histories and belief states both carry the same information about predicting the future behavior of the system. A typical history-based model keeps in memory the full history of the interaction with the system, and uses it as the agent's internal state. This approach is very useful to overcome the problem of learning transition probabilities, since internal states will correspond to histories, and the transition between two histories is always deterministic. In that case, we should learn only the probabilities of immediate observations after each history, or the values of actions after each history if we rather want to directly learn an optimal control policy.

The principal drawback of history-based models is the huge number of histories that we have to keep in memory: $(|\mathcal{A}||\mathcal{O}|)^t$ different histories for t steps of interacting [Mccallum, 1996], where $|\mathcal{A}|$ and $|\mathcal{O}|$ are respectively the number of actions and observations. This problem has been partially addressed in Utile Distinction Memory (UDM) [McCallum, 1993] and Utile Suffix Memory (USM) [McCallum, 1995] models, where we keep in memory only the last sequence (suffix) of each history that is necessary to distinguish it from other histories. Two histories are considered as equivalent if all the actions have almost the same value at these histories. A statistical test (Kolmogorov-Smirnov for example) is used to verify if there is a significant divergence between two histories, in such case we consider longer suffixes for them, until an acceptable divergence is achieved.

The main problem of history-based models remains the fact that the agent often should remember the full history from the beginning in order to make accurate predictions about the future behavior (think of a system where the first executed action can determine the last observation after an arbitrary number of time-steps). This problem was a major factor that severely restrained the applicability of this type of models.

2.5.2 Diversity-based Machines

PSRs are based on the previous work of Rivest and Schapire [1994] on Diversity-based inference of deterministic Finite-State Automata (FSA). A deterministic FSA $M(S, O, \delta : S \rightarrow S, \gamma : S \times O \rightarrow \{true, false\})$ is defined by S: a finite set of states; O: a finite set of observations (called labels); δ: a deterministic transition function; γ: a deterministic observation function. A test t is defined as a non empty finite sequence of observations, i.e. $t = o_1 \dots o_n$. The set of all possible tests that can be executed on the automaton is indicated by T. Rivest and Schapire [1994] used the notation $\langle s|t\rangle \in \{true, false\}$ to indicate if test t will succeed when it is executed from s. Two tests t_1 and t_2 are equivalent if and only if: $\forall s \in S : \langle s|t_1\rangle = \langle s|t_2\rangle$, and two states s_1 and s_2 are equivalent if and only if: $\forall t \in T : \langle s_1|t\rangle = \langle s_2|t\rangle$. We indicate by $D_t(M)$ the test diversity of the automaton M, which is the number of equivalence classes of tests.

Rivest and Schapire [1994] proved that after eliminating duplicated states (states that are equivalent to others) the diversity of the automaton is bounded as:

$$\log(|S|) \leqslant D_t(M) \leqslant 2^{|S|}$$

Rivest and Schapire [1994] also showed that these bounds are the best possible. The Diversity-Based Machine that is equivalent to M is a deterministic machine where the states correspond to the tests equivalence classes, this machine can make predictions about any test, thus, it can be used as a generative model for predicting observations. Models based on deterministic automaton are relatively easy to understand and to interpret. However, diversity-based machines are limited to deterministic dynamical systems, PSRs are basically a generalization of these machines for handling stochastic dynamical systems.

2.5.3 Observable Operators Models

Predictive State Representations share many of their properties with another class of models known as Observable Operators Models (OOMs) [Jaeger, 2000]. As for PSRs, OOMs are motivated by the fact that HMMs are difficult to learn from a given stream of observations. There are two versions of OOMs: an interpretable version, in which the parameters correspond to probabilities and can be used for learning a dynamical system or predicting its future behavior, and an uninterpretable one, which generalizes the first version by relaxing the constraints on probability distributions.

OOMs were designed to model uncontrolled systems (without actions). The sufficient information in OOMs is given by the probabilities of sets of sequences, called *characteristic events*. A characteristic event is a set $E_i = \{q_1, q_2, \ldots, q_{n_i}\}$ where $q_1, q_2, \ldots, q_{n_i}$ are tests of the same length. A PSR core test is a special characteristic event containing only one sequence. However, OOMs constrain all the tests to have the same length, which is a restriction of PSR tests. OOMs make this restriction in order to force the tests contained in a characteristic event to be mutually exclusive, the probability of an event E_i is then given by the sum of the probabilities of its elements.

Singh et al. [2004] proved that OOMs and uncontrolled PSRs are equivalent, but there has been no empirical study comparing PSRs and OOMs learning algorithms, OOMs could be easier to learn given that the likelihood of observing a specific characteristic event (set of tests) in a sampled sequence is higher than the likelihood of observing a specific single test.

2.5.4 Temporal-Difference Networks

Temporal-Difference Networks [Sutton and Tanner, 2004, 2005; Makino and Takagi, 2008] are another predictive model very close to PSRs. The state of the system is represented by predictions about future events, as in the case of PSRs. However, the core tests are interconnected between them in a compositional and recursive way. For example, a test aq is connected to the test q with the action a. Thus, TD-learning methods [Sutton and Barto, 1998] can be used to learn the parameters related to a test by back-propagating the values of all its descendants. This is not trivial in PSRs where the core tests are treated independently, and the parameters are generally learned by Monte Carlo methods [Singh et al., 2003].

Although TD-Networks are very attractive for learning, they lack rigorous theoretical properties compared to PSRs. As an example of this, the update function defined in TD-Networks is a nonlinear operator inspired from neural networks and it is not clear how it is related to the Bayesian update, as in PSRs or POMDPs. Empirically, PSRs and TD-Networks learning algorithms exhibit similar performances on most benchmark problems [Sutton and Tanner, 2004], for this reason, we are more interested in PSRs since they can be used for learning, planning and even for reducing the state space dimensionality as we will show in the next chapter.

2.5.5 Duality-based Machines

Based on the work of Rivest and Schapire [1994] on diversity-based machines (described in subsection 2.5.2), Hundt et al. [2006] proposed a new model known as Duality-based machines by using the notion of equivalence of experiments. A test t is defined as a non empty finite sequence of observations followed by an action, i.e. $t = o_1 \dots o_n a$. An experiment $e \in E$ is a non empty sequence of tests $t_1 t_2 \dots t_m$. Two experiments e_1 and e_2 are equivalent if and only if $\forall s \in S : Pr(e_1|s) = Pr(e_2|s)$. The equivalence class of an experiment e is denoted by $[e]_{\mathcal{M}}$, where $\mathcal{M}$ is the POMDP model of the considered system. Two states s_1 and s_2 are equivalent iff $\forall e \in E : Pr(e|s_1) = Pr(e|s_2)$, the equivalence class of a state s is denoted by $[s]_{\mathcal{M}}$.

Given a POMDP model $\mathcal{M}$, a dual machine $\mathcal{M}'$ is generated by considering each equivalence class $[e]_{\mathcal{M}}$ as a state, i.e. $\mathcal{S}' = \{[e]_{\mathcal{M}}\}$. The transition function in $\mathcal{M}'$ is deterministic: $\delta'_a([e]_{\mathcal{M}}) = [ae]_{\mathcal{M}}$. A history h in $\mathcal{M}'$ is defined as a state followed by a non empty sequence of tests: $h = st_1 t_2 \dots t_m$. Two histories h_1 and h_2 are equivalent if and only if $\forall e \in E : Pr(e|h_1) = Pr(e|h_2)$, the equivalence class of a history h is denoted by $[h]_{\mathcal{M}'}$.

Given a dual machine $\mathcal{M}'$, a double-dual machine $\mathcal{M}''$ is generated by considering each equivalence class $[h]_{\mathcal{M}'}$ as a state, i.e. $\mathcal{S}'' = \{[h]_{\mathcal{M}'}\}$. Similarly, the transition function in $\mathcal{M}''$ is deterministic, the observation function returns the probabilities of all the experiments in a given history equivalence class $[h]_{\mathcal{M}'}$.

The double-dual machine $\mathcal{M}''$ can be used for planning or predicting observations as with the original machine $\mathcal{M}$. Interestingly, the number of states in $\mathcal{M}''$ can be significantly smaller than the number of states in $\mathcal{M}$. However, one can find very simple examples where the number of states in $\mathcal{M}''$ is higher than the number of states in $\mathcal{M}$, or even infinite.

2.5.6 Causal-State machines

Causal-State machines (CSMs) [Shalizi and Shalizi, 2004] are similar to duality-based machines, each state of a CSM corresponds to a history equivalence class. CSMs are restrained to uncontrolled systems, but can be generalized to controlled systems.

Shalizi and Shalizi [2004] proposed an efficient algorithm for learning a CSM model from a given sequence of observations. This algorithm, called Causal-State Splitting Reconstruction (CSSR), is close to USM [McCallum, 1995] in the sense that it starts with the empty history class as the only state, and then uses a statistical test to check the accuracy of the model predictions. If the test fails with some probability α, then the corresponding state is split and longer histories are considered in the new states. This operation is repeated until no more states can be split or the maximum length of histories is reached.

Contrary to USM, the histories of the same equivalence class in CSMs do not necessarily share a common suffix. In fact, we can find two equivalent histories with completely different observations, which can be reflected by a smaller number of equivalence classes compared to USM. As for duality-based machines, the main drawback of CSM is the fact that many systems have an infinite number of history equivalence classes.

2.6 Summary

This chapter introduces the concept of discrete-time dynamical systems, and shows how it can be used as a mathematical framework for decision-making. A simple, yet powerful, model of dynamical systems known as Markov Decision Process (MDPs) is described, as well as some algorithms for planning and learning with this model. Partially Observable MDPs (POMDPs) are presented as a generalization of MDPs for handling noisy and aliasing input observations. Other models of partially observable dynamical systems are also discussed in this chapter.

One of the major issues of planning with POMDPs is the high dimensionality of the belief space. Predictive State Presentations (PSRs) have been introduced as an alternative model to POMDPs, with the hope of leading to more compact representations of the belief space. Despite their theoretical potential, PSRs had been found to have almost the same dimensionality as POMDPs in most of benchmark problems.

Instead of dealing with exact high-dimensional models, approximate compact models may be a better option for efficient planning. The next chapter presents a new algorithm for transforming a POMDP model into an approximate PSR with a customized compression ratio. Empirical results in simulated domains show how an online planning algorithm has been accelerated by using approximate PSRs, with no significant loss of performance.

Chapter 3

State Space Compression in POMDPs with Predictive Representations

High dimensionality of the belief space in POMDPs is one of the major causes that severely restricts the applicability of this model in planning and prediction tasks. Previous studies have demonstrated that the dimensionality of a POMDP can eventually be reduced by transforming it into an equivalent Predictive State Representation (PSR). In practice, the equivalent PSR will have a dimension close or equal to the one of the original POMDP. In this chapter, we address the problem of finding an approximate and compact PSR model corresponding to a given POMDP. We formulate this problem as an optimization problem. Our algorithm minimizes the potential error caused by missing some core tests. In this context, we present an empirical evaluation on benchmark problems, illustrating the performance of this approach on prediction and planning tasks.

3.1 Introduction

A number of representations for dynamical systems have been proposed during the past two decades beside Partially Observable Markov Decision Processes (POMDPs) (presented in Section 2.3 of Chapter 2). However, they either impose restrictions to the underlying environment, or they do not seem to provide any advantages over POMDPs. Among these representations, PSRs (presented in Section 2.4 of Chapter 2) seem appealing for several main reasons. First, PSRs are grounded in the sequence of actions and observations of the agent, and hence relate the state representation directly to the agent's experience. Another reason why the predictive approach is considered to be a good choice for state representation is related to the generalization problem. Indeed, the predictive representation does not rely on a specific physical layout

of an unobservable environment, so it has the potential of being useful for fast adaptation to a new similar environment. Finally, PSRs offer a representation for dynamical systems which is as general as POMDPs, and can be potentially more compact than POMDPs [Singh et al., 2004]. This is particularly useful for planning and predicting observations in large domains, where the dimensionality of the state space is a major drawback.

However, the equivalent PSR of a POMDP model has generally almost the same dimensionality as the original model [James et al., 2004; Izadi, 2007]. Consequently, it is natural to look for approximation algorithms that generate a subset of core tests more appropriate for planning purposes. We are also interested in reducing the uncertainty by having a problem formulation as detailed as possible. These two goals involve a trade-off between the dimensionality of the PSR model and the accuracy of the solution that can be obtained. The reduced PSR model, which contains only a subset of core tests, conveys this trade-off.

In this thesis, we address the issue of dynamically generating a subset of core tests for predictive representations in partially observable domains. We are interested in a lossy compressed PSR with a number of core tests fewer than the number of states in the corresponding POMDP. We formulate this problem as an optimization problem that minimizes the loss function related to prediction of observations and rewards. We also show how to use the compressed PSR for online planning. Most of the compression methods that we will review use matrix notations. To be conform with these methods, we will denote by $T^{a,o}$ the transition-observation matrix for action $a \in \mathcal{A}$ and observation $o \in \mathcal{O}$, defined as: $T^{a,o}(s, s') = T(s, a, s')Z(o, s, a), s, s' \in \mathcal{S}$, where T and Z are respectively the transition and observation functions defined in Section 2.3 of Chapter 2. For the same reason, we also make the weak assumption that the rewards depend only on states, and use the vector R to represent the reward function.

3.2 Exact Planning with POMDPs

Planning with POMDPs refers to the task of finding an optimal policy for a given horizon H. In this context, a policy is a function from belief states to actions, or to a distribution on actions:

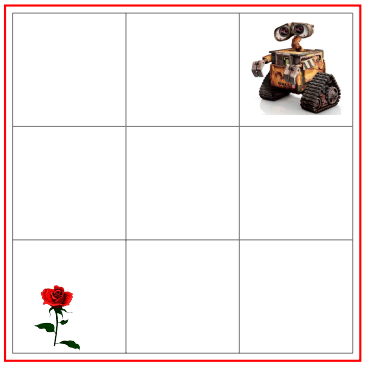
$$\pi : \Delta\mathcal{S} \rightarrow \Delta\mathcal{A}$$

where $\Delta\mathcal{S}$ is the set of belief states (an $|\mathcal{S}| - 1$ dimensional simplex), and $\Delta\mathcal{A}$ is the set of distributions on actions (an $|\mathcal{A}| - 1$ dimensional simplex). When a policy π is deterministic, $\pi(b)$ refers to the action selected by π in the belief state b. The value function of a deterministic policy π starting from the belief state b is given by:

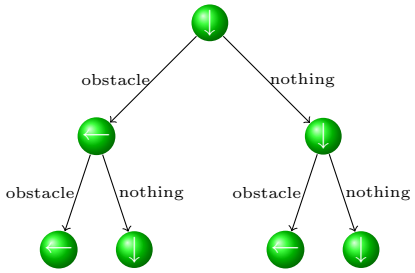
$$V^{\pi}(b) = \mathbb{E}_{b_t, a_t}\left[\sum_{t=0}^{H} \gamma^t b_t^T R | \pi, \{T^{a,o}\}, b_0 = b\right] \tag{3.1}$$

where b_t^T is the transpose of b_t, the belief state at time-step t, R is the vector of rewards, and $b_t^T R$ is the expected reward given the belief state b_t.

As the set of belief states is infinite, not every policy can be finitely represented. However, given an initial belief state b_0, the set of upcoming beliefs b_t for a finite horizon H is finite, and each one of these beliefs can be calculated by using b_0 and the sequence of actions and observations that lead to it. Therefore, a policy π can be represented by a decision tree, where each node correspond to a particular belief state and is labeled by an action, and each edge is labeled by an observation, as depicted in Figure 3.1. the first action (root) of policy π is denoted by $A(\pi)$, and the subtree of π under action a and observation o is denoted by $P(\pi, o)$.



(a) A simple POMDP problem.



(b) A policy represented as a decision tree.

Figure 3.1: Representing a POMDP policy as a decision tree. The robot can detect the presence or absence of an obstacle after moving into it, the actions correspond to movements to east, west, north, and south.

If policy π is a decision tree, then Equation 3.1 can be rewritten as a linear function of the belief state b. In fact, the value of policy π for $H = 0$ corresponds to the expected immediate reward given the belief b: $V^\pi(b) = b^T R$, which is a linear function of b. Therefore, the value function V^π can be represented by a vector α^π, called an *α-vector*. The value of policy π for a given horizon H can be written as:

$$
\begin{aligned}
V^\pi(b) &= \mathbb{E}_{b_t,a_t}\left[\sum_{t=0}^{H} \gamma^t b_t^T R | \pi, \{T^{a,o}\}, b_0 = b\right] \\
&= b^T R + \gamma \sum_{o \in \mathcal{O}} Pr(o|b, A(\pi)) \mathbb{E}_{b_t,a_t}\left[\sum_{t=0}^{H} \gamma^t b_t^T R | \pi, \{T^{a,o}\}, b_0 = \tau(b, A(\pi), o)\right] \\
&= b^T R + \gamma \sum_{o \in \mathcal{O}} Pr(o|b, A(\pi)) V^{P(\pi,o)}(\tau(b, A(\pi), o)) \\
&\qquad\qquad \text{(The belief update function } \tau \text{ is defined by Equation 2.2)} \\
&= b^T R + \gamma \sum_{o \in \mathcal{O}} Pr(o|b, A(\pi)) V^{P(\pi,o)}\left(\frac{b^T T^{A(\pi),o}}{Pr(o|b, A(\pi))}\right)
\end{aligned}
$$

$$\begin{aligned} V^{\pi}(b) &= b^T R + \gamma \sum_{o \in \mathcal{O}} Pr(o|b, A(\pi)) \frac{b^T T^{A(\pi),o}}{Pr(o|b, A(\pi))} \alpha^{P(\pi,o)} \\ &= b^T R + \gamma \sum_{o \in \mathcal{O}} b^T T^{A(\pi),o} \alpha^{P(\pi,o)} \\ &= b^T \alpha^{\pi} \end{aligned} \tag{3.2}$$

where

$$\alpha^{\pi} = R + \gamma \sum_{o \in \mathcal{O}} T^{A(\pi),o} \alpha^{P(\pi,o)} \tag{3.3}$$

One can generally distinguish between two types of planning: *online* and *offline*. Offline planning consists in finding a policy π before starting to interact with the environment or to execute any action. An advantage of offline approaches is that the running time of a planning algorithm is only limited by the computational resources that are available to the decision-maker, and not the response time of the system. However, the initial belief state is not always available when one is planning offline.

There have been many algorithms for offline planning in POMDPs that find an optimal policy covering the entire belief space [Cassandra, 1998; Zhang and Zhang, 2001; Feng and Zilberstein, 2005; Sondik, 1971]. Among these algorithms, Sondik's Enumeration algorithm is the most straightforward to describe [Sondik, 1971]. Let us denote by Π^H the set of all possible policies (decision trees) for horizon H. The value functions of these policies is a set of α-vectors $\alpha^{\pi}, \pi \in \Pi^H$ (from Equation 3.2). The optimal policy for a belief state b corresponds to $\arg\max_{\pi \in \Pi^H} b^T \alpha^{\pi}$. Therefore, the optimal value function for the entire belief space is a piecewise linear function corresponding to the α-vectors of non-dominated policies[1]. Sondik's algorithm consists in enumerating all the possible α-vectors for horizon H by performing a backup of the α-vectors for horizon $H-1$ using Equation 3.3. A linear program is then used to determine dominated policies and eliminate them.

Even when an initial belief is available, one should find an optimal policy that covers all the possible beliefs within a horizon H. Given an initial belief state b, and a finite horizon H, this problem can be solved by a *forward search* algorithm. Algorithm 4 describes the principal steps of a generic forward search. To accelerate the search, a heuristic lower bound $\underline{V}$ and a heuristic upper bound $\bar{V}$ on the optimal value function V^* are used for pruning sub-trees that are obviously not optimal (steps 1 to 5). A survey of such heuristics can be found in [Ross et al., 2008b]. The algorithm recursively calculates the values of all the other actions (steps 6 to 11) and returns the optimal one (steps 12 to 15).

Forward search is mostly used in online planning algorithms, where the decision-maker should find optimal actions while interacting with system. Since the time-window available

[1] A policy is said non-dominated if there is at least one belief state where the value of this policy is higher than the values of other policies.

for making a decision is generally very small, most forward search methods use a short planning horizon, along with several heuristics, in order to reduce their runtime [Paquet, 2006; Ross et al., 2008b].

Input: A POMDP model $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R \rangle$, an initial belief state b, and a horizon H ;
1 /* $\underline{V}$ and $\bar{V}$ are respectively lower and upper bounds on V^* */;
2 $V^*(b) \leftarrow \underline{V}$;
3 **foreach** $a \in \mathcal{A}$ **do**
4 $V(b) \leftarrow b^T R + \gamma \sum_{o \in \mathcal{O}} Pr(o|b,a) \bar{V}(\tau(b,a,o))$;
5 **if** $V(b) > V^*(b)$ **then**
6 $V(b) \leftarrow b^T R$;
7 **foreach** $o \in \mathcal{O}$ **do**
8 Let $V^*(\tau(b,a,o))$ be the optimal value for horizon $H-1$ returned by a forward search starting at the belief $\tau(b,a,o)$;
9 $V(b) \leftarrow V(b) + \gamma Pr(o|b,a) V^*(\tau(b,a,o))$;
10 **end**
11 **end**
12 **if** $V(b) > V^*(b)$ **then**
13 $V^*(b) \leftarrow V(b)$;
14 $\pi^*(b) \leftarrow a$;
15 **end**
16 **end**
Output: An optimal policy π^* and value $V^*(b)$ for horizon H;

Algorithm 4: A generic forward-search algorithm.

3.3 The Curse of Dimensionality

Planning with POMDPs in real-world domains is known to be a very hard problem. In fact, finding an optimal policy for a POMDP is proved to be a PSPACE-complete problem [Papadimitriou and Tsitsiklis, 1987], and verifying the existence of a policy with an expected value greater than a given threshold ϵ is undecidable [Madani et al., 1999]. Even finding a near-optimal policy for a POMDP is NP-hard [Lusena et al., 2001]. These complexity results remain valid for PSRs too, since each POMDP model can be transformed to a PSR in a polynomial time [Littman et al., 2001]. There are two main factors, known as the *curse of dimensionality* [Bellman, 1957], behind the poor scalability of a POMDP planning algorithm:

The dimensionality of the state space: A state is generally defined as a joint instantiation of many variables. Thus, the dimension of the state space grows exponentially with respect to the number of variables. The size of belief states, value vectors, and of many other structures used in POMDP algorithms corresponds to the number of states.

The dimensionality of the policy space (also knows as the curse of the history): The number of policies that should be evaluated at each step of the planing horizon is a double-exponential of the number of observations and the length of the horizon. Indeed, if we have $|\mathcal{A}|$ actions and $|\mathcal{O}|$ observations, then the total number of policies with horizon t is $|\mathcal{A}|^{\frac{|\mathcal{O}|^t-1}{|\mathcal{O}|-1}}$. The expected value of each policy π is given by a value vector α_π, defined on the state space, where $\alpha_\pi(s)$ is the expected value of the policy π, if we start executing this policy from state s (see the previous section). The number of basic operations required to calculate the value of a policy is $O(|\mathcal{S}|)$, and we have to repeat this computation for every policy, i.e. $O(|\mathcal{A}|^{\frac{|\mathcal{O}|^t-1}{|\mathcal{O}|-1}})$ times. Consequently, if we can reduce the number of states involved in calculating value vectors, then we will be able to significantly improve the computational efficiency of POMDP planning algorithms.

PSRs have been initially proposed as a tentative to address the problem of the dimensionality of the state space. By using only linearly independent tests to define the parameters, a PSR model has at most the same dimension as the equivalent POMDP. We will show how to furthermore reduce this dimension in Section 3.7. PSRs have also been used to address the curse of the history in POMDPs [Izadi, 2007], where policies were kept only in pertinent belief states, corresponding to core histories. In Chapter 4, we will show how to use PSRs to address the same issue in a more difficult problem related to *decentralized POMDPs*. But before that, we will first review in the next section the principal techniques used for dealing with the curse of dimensionality.

3.4 Approximate Planning with POMDPs

The techniques used to overcome the curses of dimensionality in POMDPs can be regrouped into the following categories: heuristic, grid-based, point-based, policy search, forward search, structure-based, and compression-based approaches.

3.4.1 Heuristic Approaches

Heuristics are often used to compute upper and lower bounds on the optimal value function, with a very low computational cost. These bounds can be directly used as an approximation of the optimal value function, or combined with other planning algorithms (such as forward-search, Algorithm 4) in order to orient the search in promising directions and to apply branch-and-bound pruning techniques [Ross et al., 2008b]. Most of heuristic approaches for POMDPs are based on using the underlying MDP by assuming that the states are complectly observable.

Among these heuristics, the simplest one is given by the value function of a *Blind policy* [Hauskrecht, 2000; Smith and Simmons, 2005]. A blind policy is a policy where the same action is always executed, regardless of the belief state. The value of a blind policy is obviously a lower bound on the optimal value function, as it corresponds to a specific policy. Given a belief state b, a lower bound on the optimal value function $V^*(b)$ is given by $V^{blind}(b) = \max_{a \in \mathcal{A}} b^T \alpha^a$, where

$$\alpha^a(s) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') \alpha^a(s')$$

Although this latter bound can be computed very quickly, by solving $|\mathcal{A}|$ systems of $|\mathcal{S}|$ linear equations with $|\mathcal{S}|$ variables, it is generally loose and not very informative. Another simple heuristic consists in executing the optimal action of the *Most Likely State* (MLS) [Nourbakhsh et al., 1995]. This heuristic is given by $V^{MLS}(b) = \max_{a \in \mathcal{A}} Q^*(\arg\max_{s \in \mathcal{S}} b(s), a)$, where $Q^*(s,a)$ is the Q-value of action a in state s, defined in Equation 2.2. This heuristic is generally tight only when there is one state with a probability that is significantly higher than the other states.

The MDP heuristic consists in approximating the optimal value function of a POMDP by using the optimal value function of the underlying MDP [Littman, 1996]. An upper bound on the optimal value function $V^*(b)$ for a belief b is given by $V^{MDP}(b) = b^T \alpha^*$, where

$$\alpha^*(s) = \max_{a \in \mathcal{A}} \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') \alpha^*(s') \right]$$

The *QMDP* heuristic is a slight improvement of the MDP heuristic, where the underlying current state is partially observable, but the future states are supposed to be completely observable. The QMDP heuristic is defined as $V^{MDP}(b) = \max_{a \in \mathcal{A}} \sum_{s \in \mathcal{S}} b(s) Q^*(s,a)$. Note that this bound is tighter than the MDP bound. An interesting generalization of this heuristic, known as *k-QMDP* [Besse and Chaib-draa, 2009], consists in taking into account the partial observability of the environment for k steps, and then using the policy of the underlying MDP for the remaining steps.

Finally, the *Fast Informed Bound* (FIB) [Hauskrecht, 2000] takes into account the probabilities of the observations for one time-step. This bound is computed as in QMDP, by using the following modified Q-values:

$$Q^{FIB}(s,a) = R(s,a) + \gamma \sum_{o \in \mathcal{O}} \max_{a' \in \mathcal{A}} \sum_{s' \in \mathcal{S}} Z(o,s',a) T(s,a,s') Q^*(s',a')$$

Despite the low computational cost associated to heuristic approaches, they are generally not used for planning because of their poor performance. They are more often used as subroutines for accelerating the search in other planning approaches.

3.4.2 Policy Search Approaches

Instead of computing the value function of a belief state, and then finding an optimal function accordingly, policy search methods directly optimize a parametric policy. This type of algorithms depends on the model and the parameters used for representing policies. A popular model for representing infinite-horizon policies is known as Finite-State Machines (FSMs). An FSM policy is defined by a finite set of internal states, where each internal state is labeled by an action to be executed in that state, and a transition function that maps each internal state and observation into a next internal state. Based on this model, Hansen [1998] proposed a policy iteration algorithm where internal states are iteratively added and removed.

Most of the other optimization techniques for FSMs are based on using parametric stochastic machines, where the parameters correspond to the probabilities of selecting actions and transitioning to next internal states. The gradient of the value function of the FSM is then computed and used for updating the parameters [Chrisman, 1992; Meuleau et al., 1999; Baxter and Bartlett, 2000].

More recently, Vlassis and Toussaint [2009] proposed a new policy approach for both MDPs and POMDPs, where the planning (and learning) problem was casted as the problem of maximizing the likelihood of a probabilistic mixture model via sampling. The principal idea behind this framework is to use rewards between 0 and 1 and to reinterpret the value function as the probability of succeeding a task. An Expectation Maximization (EM) algorithm is then used to find the parameters of the policy that maximizes the likelihood of succeeding the task.

Policy search techniques are well-suited for planning in problems where the value of a policy cannot be easily expressed as a function of the states, such as in problems with continuous state spaces for example. However, both policy gradient and likelihood maximization methods for searching policies are local optimization methods, and can be trapped in a poor locally optimal solution.

3.4.3 Grid-based Approaches

A straightforward approach for approximating the optimal value function, defined on the entire belief space, is to discretize the belief space by the means of a regular grid [Hauskrecht, 1997; Bonet, 2002; Zhou and Hansen, 2001]. In this context, the value function is approximated by extrapolating the value function at the beliefs corresponding to the nodes of the grid. These beliefs are selected in a grid pattern in order to ensure a maximum covering of the belief space. An extended overview of Grid-based approaches can be found in [Pineau, 2004]

Given a set of grid belief points B, the optimal value function for each belief $b \in B$ is approximated as:

$$V(b) = \max_{a \in \mathcal{A}} \left[b^T R + \gamma \sum_{o \in \mathcal{O}} Pr(o|b,a) V(\tau(b,a,o)) \right]$$

where $V(\tau(b,a,o))$ is the value of the next belief $\tau(b,a,o)$. If $\tau(b,a,o) \notin B$, then the following approximation is used: $V(\tau(b,a,o)) \approx \sum_{b_i \in B} \lambda_i(\tau(b,a,o)) V(b_i)$, where $\lambda_i(\tau(b,a,o)) \geqslant 0$ and $\sum_{b_i \in B} \lambda_i(\tau(b,a,o)) = 1$. The convex combination λ_i can be found by solving a linear program.

A major inconvenience of regular grid approximations lies in the fact that the distribution of reachable belief points is generally not uniform over the belief space. In practice, the belief points are concentrated in small dense regions. We also note that the interest in grid-based methods dropped down after the introduction of point-based methods, which are presented in the next section.

3.4.4 Point-based Approaches

Although the space of belief states is an $|\mathcal{S}|$-dimensional continuous simplex (the set of valid probability distributions over the states), only a few regions of it are reachable in practice. Thus, one should look only for the policies that are optimal in these regions. Moreover, one can choose a finite set of belief points that are very likely to occur during the control process, and evaluate the candidate policies only in these points. This type of approaches, known as point-based algorithms, have become very popular due to their ability to solve real-world problems with a large state space, where exact methods are almost useless [Pineau, 2004].

Point-Based Value Iteration (PBVI) [Pineau et al., 2003] is an offline algorithm that uses a limited list $B_t = \{b_t^1, b_t^2, \ldots, b_t^n\}$ of belief points that are reachable at time-step t, and keeps the best policy for every point in this list. More specifically, the value function of a policy is an α-vector (from Equation 3.2), and the optimal policy for a belief state b_t^i corresponds to $\arg\max_{\pi \in \Pi^t} \sum_{s \in \mathcal{S}} b_t^i(s) \alpha^\pi(s)$, where Π^t denotes the set of policies at step t. Finding the best policy for a given belief point can be performed in $O(|\Pi^t||\mathcal{A}|)$ operations, which is much faster than solving $O(|\Pi^t|)$ linear programs for finding the non-dominated policies over the whole belief space. At the end of each step, a new set of policies Π^{t-1} is created by extending the policies of the previous set Π^t. Regarding the set of reachable belief points, PBVI starts with a single belief point b_0, then it gradually adds more belief points based on the probability of reaching them during the execution phase, and their distance from the previous belief states. Given a set of beliefs B_t, PBVI generates for each belief $b_t \in B_t$ and action $a \in \mathcal{A}$ a new belief $\tau(b_t, a, o)$, where o is an observation randomly sampled according to b_t and a. Finally, only the farthest beliefs, using the L_1 norm, are kept and added to the list B_t to create B_{t+1}.

The Heuristical Search Value Iteration (HSVI) [Smith and Simmons, 2004] algorithm defines upper and lower bounds on the optimal expected value over the whole belief space.

Similarly to PBVI, HSVI keeps only the policies that are optimal in some belief points, these points are selected according to their reachability and the difference between their lower and upper bounds. This manner of collecting belief points leads to a gradual decrease in the difference between the upper bound and the true optimal value function over the whole belief space. As a consequence of that, less belief points are needed in HSVI than in PBVI, and a near-optimal policy can be found in a shorter time.

Other methods of selecting belief points include the Belief-Based Error Minimization Algorithm (PEMA) [Pineau and Gordon, 2005], where the beliefs are selected so that a theoretical bound on the error of the algorithm is minimized, Perseus [Spaan and Vlassis, 2005], where the beliefs are selected in a randomized greedy manner, and Forward Search Value Iteration (FSVI) [Virin et al., 2007; Shani et al., 2008a], where the value function of the underlying MDP is used to sort the belief points, and to perform prioritized backups according to this order.

Point-based approaches are very efficient in practice, however, they require that the initial belief state be known in advance, while in many problems, the initial belief state may change from an instance to another.

3.4.5 Forward Search Approaches

Offline planning methods should find a policy for each possible initial belief state. However, in many real-world problems, the initial belief state cannot be known in advance. Online planning methods, based on Forward Search, provide an efficient solution to this problem by focusing the search on the belief points that are reachable in a short-term horizon, and using heuristic bounds, calculated offline, for the remaining beliefs.

Real-Time Belief Space Search (RTBSS) [Paquet et al., 2005] was the first Forward Search algorithm for POMDPs used for solving problems with large state and action spaces. This algorithm starts with an initial belief point, and construct a decision tree of the optimal policy by using heuristics, such as QMDP and blind policy, for pruning dominated actions. This algorithm is very efficient for real-time applications with a long planning horizon and a large state space [Paquet, 2006; Wolf, 2009].

Anytime Error Minimization Search (AEMS) [Ross and Chaib-draa, 2007] is another online search algorithm. The search tree in AEMS is explored in an order that minimizes the expected error that can result from ignoring some observations in the backup operation, while focusing only on the most significant observations. This algorithm is well suited for real-world problems, characterized by a large number of observations.

A forward search algorithm was used by Doshi et al. [2008] in a framework for bayesian

active learning in POMDPs, where phases of forward search and learning were interleaved. In this context, the belief state is a probability distribution over states and POMDP models. Given the high number of possible POMDP models, one cannot sample all the reachable belief points during a forward search. The proposed solution consists in accelerating the search by assuming that the received observations will be used to update the beliefs about the state only, and not the model, which leads to a smaller search tree.

Another forward search algorithm was used in a bayesian framework by Martinez-Cantin et al. [2009]. The considered POMDP model was high-dimensional, continuous, nonlinear, and non-Gaussian, representing a challenging path planning problem for a mobile robot. To evaluate a parametric policy for a given belief state, Martinez-Cantin et al. [2009] proposed to sample several trajectories starting from that belief, and then use a direct policy search to estimate the stochastic gradient of the value function. The estimated gradient is then used for updating the parameters of the policy.

Finally, Roy and He [2009] proposed another efficient forward search planner for continuous domains, known as Posterior Belief Distribution (PBD) algorithm. Roy and He [2009] showed that for a special class of transition and observation functions, namely gaussian and exponential families, the belief states can be calculated in a closed-form, without enumerating the observations. This property enables one to plan in problems with large observation spaces, and also to search deeper by considering policies composed of multi-step action sequences.

Although they are very practical, forward search methods, as well as point-based approaches, tackle only the problem of the policy space dimensionality (the curse of the history), leaving the problem of the dimensionality of the state space untouched. The next two approaches that we will present try to solve this latter issue.

3.4.6 Structure-based Approaches

The planning process can be accelerated by exploiting the specific structure of the problem. This idea had been widely explored in Markov Decision Processes (MDPs) to solve problems with a large state space. Most of the work in this topic is based on decomposing the global problem into smaller sub-problems, and solving each sub-problem separately. There are two general approaches for decomposing the global problem: state space decomposition methods and policy space methods approaches.

In the state decomposition methods, it is assumed that the state space is composed of several loosely coupled sub-regions, with a weak mutual interaction between them. The transition between two regions should pass through an intermediate bottleneck state [Dean and Lin, 1995]. The optimal value of each couple (state, action) is calculated following a topological order of the state space, defined according to the location of the target states [Bonet and

Geffner, 2003; Dai and Goldsmith, 2007; Dibangoye et al., 2009b; Dibangoye, 2010].

In the policy decomposition approaches, also known as *abstraction approaches*, the global task is divided into smaller subtasks, each subtask can be solved by a set of sub-policies (or *options*), defined *a priori* by the designer. There are at least two planning levels: a lower level, where one should find the optimal sub-policy for each sub-task; and a higher level, where one should use these sub-policies as macro actions in order to find the optimal global policy.

There has been some interesting attempts to apply the decomposition techniques in partially observable domains. The proposed solutions generally make the strong assumption that the structure of the domain was previously decomposed. As an example, Hierarchical Q-learning algorithm [Wiering and Schmidhuber, 1997] finds optimal policies through a sequence of subgoals, assuming that the subgoal states are completely observable. Another approach, called HPOMDPs (Hierarchical POMDPs) [Theocharous et al., 2001, 2005; Foka and Trahanias, 2007; Toussaint et al., 2008] extends the Hierarchical Hidden Markov Model (HHMM) [Fine et al., 1998] to POMDPs. In the HPOMDP model, there can be different levels of abstraction (detail), each level is a POMDP model itself with a transition and observation function. A special transition function, called the vertical transition function, defines the probabilities of transitioning from a high level state to a state of a lower level. This model assumes the existence of fully observable terminating states that indicate the end of planning for a given level and the return to a higher level. Policy-Contingent Abstraction (PolCA+) [Pineau, 2004] is a hierarchical planning algorithm close to HPOMDP. However, the full observability of terminating states assumption was not necessary in this algorithm.

Factored POMDPs are a well known model that is used to overcome the problem of state dimensionality. Instead of decomposing the state space into weakly coupled components, Factored POMDPs [Boutilier and Poole, 1996; Boutilier et al., 1999; Shani et al., 2008b; Khan et al., 2009] decompose the feature space into independent sets. A state is usually defined as a joint instantiation of different variables (features), and often some of these variables are independent of others. Calculating value vectors on small subsets of variables leads to a significant gain in the computational complexity. One can also decompose the action space based on the independence between action variables and the *additive separability* of the reward function. Consequently, the global policy can be decomposed into partial policies, defined over subsets of action and state variables.

3.4.7 Compression-based Approaches

Belief compression refers to the methodology of projecting high-dimensional belief state points onto low-dimensional points in order to alleviate the problem of state space dimensionality. Contrary to structure-based approaches where the decomposition depends on the domain topology, compression techniques are based on analytical properties of the transition, obser-

vation, and reward matrices. Most of these properties rely on the regularity of belief points. In fact, while the full belief space is uncountable, the set of belief points that could be visited during the execution is countable (starting from a given initial belief point). Moreover, most of these points are related to each other, and centered around a few of them. If we can identify these points, then we will have a representation of the belief state in terms of a small set of basis variables, and finding value functions will generally be easier than in the original high-dimensional space.

Most of the compression techniques are based on finding a linear transformation function, characterized by a matrix F, that associates to every high-dimensional point b, a low-dimensional point $\tilde{b}$, such that $\tilde{b} = b^T F$. Roy et al. [2005] used Principal Components Analysis (PCA) to find the low-dimensional points. Given a set of sampled belief points $b_1, b_2, \ldots, b_n$, regrouped column by column in a matrix B, PCA technique returns low dimensional points $\tilde{b}_1, \tilde{b}_2, \ldots, \tilde{b}_n$, regrouped in a matrix $\tilde{B}$, such that the *reconstruction error* $\| B - G\tilde{B} \|_F^2$ is minimal[2], where G is a matrix corresponding to a linear function from $\tilde{B}$ to B.

Roy et al. [2005] noted that PCA performs poorly on probability distributions, and proposed a more general nonlinear compression technique based on exponential families of PCA (EPCA). In EPCA, the matrix of high-dimensional points B is given by $f(G\tilde{B})$, where f is an exponential family function and G is a function corresponding to a linear function. However, both of linear and exponential PCA cannot be used easily for planning in POMDPs, since the compression is performed only on the sampled belief points $b_1, b_2, \ldots, b_n$, and given a new belief point $\tilde{b}_i$, one cannot find the corresponding compressed belief $\tilde{b}_i = b_i^T F$, unless one can calculate $F = G^{-1}$. Nevertheless, the authors showed that one can construct an MDP where the states correspond to the reduced sampled belief points $\tilde{b}_1, \tilde{b}_2, \ldots, \tilde{b}_n$, and a Value Iteration or a Policy Iteration algorithm can be used for finding the optimal policy of this MDP. The main drawback of this approach lies in the processing time needed to transform every possible belief state into a reduced one. Note that there is a growing interest in applying spectral methods, such as PCA and EPCA, for handling problems with continuous state spaces [Huynh, 2008; Kurniawati et al., 2008; Bush and Pineau, 2009]

Based on the work of Pfeffer [2001] on *self-sufficient families*, Poupart and Boutilier [2002] proposed a compression algorithm for POMDPs where the values of policies are preserved in the compression. The algorithm, known as Value Directed Compression (VDC), uses a linear program to find a transformation matrix F, such that $\tilde{b} = b^T F$, while minimizing the errors $\|\tilde{R} - R^T F\|_\infty$ and $\|T^{ao} F - F\tilde{T}^{ao}\|_\infty$, where $\tilde{R}$ is the reward vector in the reduced belief space and $\tilde{T}^{ao}$ is the transition function in that space, as depicted in Figure 3.2. These constraints are nonlinear since the matrices F and $\tilde{T}^{a,o}$ and the vector $\tilde{R}$ are variables. This problem can be loosely reduced to a series of several linear programs (LPs): one can initialize the matrix F with random values, and alternate between optimizing $\tilde{T}^{a,o}$ and $\tilde{R}$ while F is constant, and optimizing F where $\tilde{T}^{a,o}$ and $\tilde{R}$ are constant. However, this iterative method can get stuck

[2] $\| A \|_F$ is the Frobenius norm, defined as $\| A \|_F = \sqrt{\sum_{i,j} A(i,j)^2}$

in a bad local optimum, and it is very time consuming, involving dozens calls of an LP solver for relatively small domains.

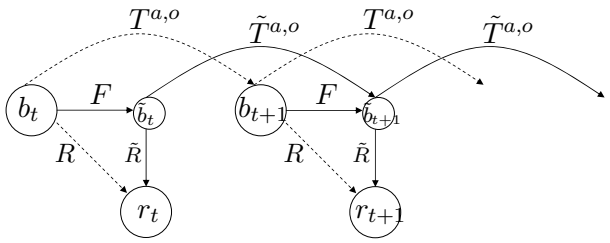


Figure 3.2: Value-directed compression [Poupart and Boutilier, 2002].

To deal with this latter problem, Poupart and Boutilier [2002] proposed to efficiently initialize F and then make only one call of a linear program solver to find the remaining parameters $\tilde{T}^{a,o}$ and $\tilde{R}$. In this context, the compression matrix F is initialized with the first vectors $R, T^{ao}R, \ldots, T^{ao}\ldots T^{ao}R$ that are linearly independent. This algorithm is value-directed because it preserves the prediction of the immediate rewards in the compressed space.

The low-dimensional belief states $\tilde{b}$ in VDC are vectors of expected rewards instead of probabilities. Given the current belief $\tilde{b}_t$, the transition function $\tilde{T}^{ao}$ can be used to calculate the next belief as $\tilde{b}_{t+1} = \tilde{b}_t\tilde{T}^{ao}$, but it cannot be used to calculate the probability of the next observation o alone. Thus, this method cannot be directly used for online planning algorithms, where belief states are updated after receiving each new observation o by calculating $Pr(o|b_t, a)$ and using Bayes' rule (Equation 2.2). We will return to this issue in Section 3.6, where we discuss the relation between PSRs and VDC. Finally, Poupart [2005] showed that this method is efficient for planning in large domains. Indeed, VDC was successfully used with the Perseus algorithm in a spoken dialogue system.

Another interesting compression algorithm, proposed by Li et al. [2007, 2009], makes use of a fast decomposition technique known as Non-Negative Matrix Factorization (NMF) [Lee and Seung, 2000]. As in [Roy et al., 2005], the NMF compression algorithm is sample-driven, i.e. given the sampled belief points $b_1, b_2, \ldots, b_n$, it finds a transformation matrix F such that $B = F^T\tilde{B}$. Moreover, this matrix satisfies the value preservation property as in [Poupart and Boutilier, 2002], i.e. one should find a matrix $\tilde{T}^{a,o}$ and a vector $\tilde{R}$ such that $F\tilde{T}^{a,o} = T^{a,o}F$ and $\tilde{R}^T = FR$. Finally, and more importantly, the function F should be orthogonal, i.e. $F^TF = FF^T = I$. This latter property allows us to pass from a high-dimensional space into a low-dimensional one. Therefore, we can use any planning algorithm to find optimal policies, yet we cannot directly use online algorithms, for the same raison as for VDC, i.e. there are no constraints used during the compression for preserving the prediction of observations.

In the remainder of this chapter, we will show that approximate PSRs provide an interesting linear compression of POMDPs, while preserving the predictions of both rewards and observations within the reduced state space. To do so, we will start by describing an efficient

way of handling rewards in PSRs.

3.5 Representing Rewards with PSRs

In order to predict observations with PSRs, one only need to know the set of core tests Q, the initial belief $\tilde{b}_0$ (defined by the probabilities of the core tests at the begining, i.e. $\tilde{b}_0(q_i) = Pr(q_i|h_0), q_i \in Q$), and the vectors m_{aoq} and m_{ao}, corresponding to the weights of the probabilities of the tests aoq and ao, i.e. $\forall a \in \mathcal{A}, \forall o \in \mathcal{O} : Pr(o|a, h_t) = \sum_{q_i \in Q} m_{ao}(q_i)\tilde{b}_0(q_i)$. The vectors m_{aoq} and m_{ao} are used to update the belief state by using Equation 2.3.

But, what about predicting rewards? In previous works on planning with PSRs [Izadi, 2007; James et al., 2004], the immediate reward was treated as a part of the observation. Consequently, the dimensionality of the observation space is multiplied by the number of different reward values, which can be as high as the dimension of the state space, since the rewards are real values. To solve this problem, we propose the following method for predicting rewards in PSRs.

We first define the expected final reward of a test q as:

$$r(q|h) = Pr(q|h) \sum_{s \in S} Pr(s|hq)R(s) \tag{3.4}$$

This value corresponds to the expected reward that will be received at the end of the test q, weighted by the probability of the test. One can show that the expected final reward of any test q depends linearly on the expected final rewards of a few tests, that we will refer to as *reward core tests*. In fact, if we denote by M the $|\mathcal{S}| \times \infty$ matrix defined as $M(s, q) = r(q|s), s \in \mathcal{S}, q \in \{\mathcal{A} \times \mathcal{O}\}^*$, then the rank of M is at most equal to the number of lines $|\mathcal{S}|$. Let us denote by $P = \{p_1, p_2, \ldots, p_n\}$ a maximal set of tests corresponding to linearly independent columns in M. These latter columns spans all the columns in M, therefore:

$$\begin{aligned} \forall q, h \in \{\mathcal{A} \times \mathcal{O}\}^* : r(q|h) &= \sum_{s \in S} Pr(s|h)r(q|s) \\ &= \sum_{s \in S} Pr(s|h) \sum_{i=0}^{n} w_q(p_i)r(p_i|s) \\ &= \sum_{i=0}^{n} w_q(p_i)r(p_i|h) \end{aligned}$$

where $w_q \in \mathbb{R}^{|P|}$ is a vector of weights. Therefore, the expected final rewards of the core tests p_i are sufficient for predicting the expected final rewards of other tests. Note that P is defined independently from Q, the set of core tests used for predicting observations, and P is generally different from Q.

The reduced reward belief is a $|P|$-dimensional vector $\tilde{r}_t$, defined as:

$$\tilde{r}_t(p_i) \stackrel{def}{=} r(p_i|h_t), p_i \in P \tag{3.5}$$

The expected final reward of any test q is given by:

$$r(q|h_t) = \tilde{r}_t^T w_q$$

where w_q is a $|P|$-dimensional weight vector. After executing action a and receiving observation o, the reduced reward belief is updated by using Bayes' Rule:

$$\begin{aligned} \forall p_i \in P : \tilde{r}(p_i|h_t ao) &= Pr(p_i|h_t ao) \sum_{s \in S} Pr(s|h_t ao p_i) R(s) \text{ (From Equation 3.4)} \\ &= \frac{Pr(aop_i|h_t) \sum_{s \in S} Pr(s|h_t aop_i) R(s)}{Pr(ao|h_t)} \text{ (Bayes' Rule)} \\ &= \frac{\tilde{r}(aop_i|h_t)}{Pr(ao|h_t)} \end{aligned} \tag{3.6}$$

So:

$$\tilde{r}_{t+1}(p_i) = \frac{\tilde{r}_t^T w_{aop_i}}{\tilde{b}_t^T m_{ao}} \tag{3.7}$$

Note that the probability $Pr(ao|h_t)$ is calculated by using $\tilde{b}_t$, the belief state of the PSR observation model. Similarly to the outcome matrix U containing the probabilities of tests (see Section 2.4), we define the $|\mathcal{S}| \times |P|$ *reward outcome matrix* F as (see Figure 3.3):

$$F(s, q) \stackrel{def}{=} r(q|s) \tag{3.8}$$

The matrix F can be found by searching for a maximum set of linearly independent columns in the matrix M. A simple iterative search algorithm for independent tests was given by [Littman et al., 2001]. We will return to this algorithm in Section 3.6.

$$\begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \\ s_8 \\ s_9 \end{array} \begin{array}{c} \begin{array}{cccc} q_1 & q_2 & q_3 & q_4 \end{array} \\ \begin{pmatrix} 0.2 & 0.5 & 1 & 0 \\ 0.5 & 0.8 & 0.3 & 0.7 \\ 0.5 & 0.9 & 0.1 & 0.3 \\ 1 & 0 & 1 & 0.2 \\ 0.5 & 0.3 & 0.8 & 0.9 \\ 0.9 & 0.6 & 0.7 & 0.3 \\ 0.9 & 0.9 & 0.5 & 0 \\ 0.9 & 0.5 & 0.3 & 0 \\ 0.7 & 0.1 & 0 & 0 \end{pmatrix} \\ U \end{array} \qquad \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \\ s_8 \\ s_9 \end{array} \begin{array}{c} \begin{array}{ccc} \emptyset & q'_1 & q'_2 \end{array} \\ \begin{pmatrix} -1 & 0.9 & 0 \\ -1 & -1 & 0.9 \\ -1 & 3.2 & -1 \\ -1 & 0 & 1.2 \\ -1 & -2 & 5.9 \\ -1 & 4 & 6.3 \\ -1 & 2 & -5 \\ -1 & -6.1 & 0 \\ 10 & 5 & 0 \end{pmatrix} \\ F \end{array}$$

Figure 3.3: An example of the outcome matrices U and F. The entries of U are the probabilities of core tests, whereas the entries of F are the expected reward of the core tests.

At a given time-step t, the immediate reward corresponds to $r(\emptyset|h_t)$, the expected final reward of the empty test $\emptyset$, so $F(.,\emptyset) = R$. We use w, the weight vector of the empty test, to find the expected immediate reward $F(\emptyset|h_t) = \tilde{r}_t^T w$ given a reduced reward belief $\tilde{r}_t$.

To conclude, both the PSR reward model $\langle \mathcal{A}, \mathcal{O}, P, w, \{w_{aop_i}\}\rangle$ and the PSR observation model $\langle \mathcal{A}, \mathcal{O}, Q, \{m_{ao}\}, \{m_{aoq_i}\}\rangle$ are needed for predicting rewards. The belief of the observation model is used to predict observations and to update the belief of the reward model (Equation 3.7). Before presenting the compression algorithm that transforms a POMDP model into a reduced PSR model, we show in the next section the equivalence between the PSR reward model and Value-directed Compression (VDC) [Poupart and Boutilier, 2002].

3.6 PSRs and Krylov subspaces

To show the relation between PSRs and VDC, we will first briefly review a linear algebraic concept called *Krylov subspace*, the interested reader can find an extended presentation of this concept in [Saad, 1996].

Let $\mathcal{V}$ be a vector space, v a n-dimensional vector contained in that space, and M a $n \times n$ matrix. A Krylov subspace $Kr(M, v)$ is the subspace of $\mathcal{V}$ that contains the vector v and that is invariant with respect to the matrix M, i.e.

$$\forall x \in Kr(M, v) : Mx \in Kr(M, v)$$

So

$$Kr(M, v) = \{v, Mv, MMv, MMMv, \dots\}$$

This definition can be generalized to subspaces with many matrices $M_1, M_2, \dots, M_k$. Similarly, a *Krylov* subspace $Kr(\{M_i\}, v)$ is the subspace of $\mathcal{V}$ that contains the vector v and that is invariant with respect to each matrix M_i:

$$\forall M \in \{M_i\}, \forall x \in Kr(\{M_i\}, v) : Mx \in Kr(\{M_i\}, v)$$

The krylov subspace $Kr(\{M_i\}, v)$ can be ordered by enumerating the vector v, then all the vectors $M_i v$, then all the vectors $M_i M_j v$, $M_i M_j M_k v$, ...etc. A basis of $Kr(M, v)$ can be constructed by keeping the first linearly independent vectors of the list $Kr(M, v)$. In fact, one can show that $\forall m > n$: the vector $\underbrace{M_i M_j \dots M_k}_{m \text{ matrices}} v$ can be written as a linear combination of the vectors $\underbrace{M_i M_j \dots M_k}_{l \text{ matrices}} v$, with $l \leqslant n$.

VDC finds a basis matrix F for the *Krylov* subspace $Kr(\{T^{a,o}\}, R)$ by iteratively enumerating linearly independent vectors of this space. The first column of F is R, then all

the vectors $T^{a^1o^1}R, \forall a^1 \in A, \forall o^1 \in O$ are generated, and the linearly independent vectors are added to F. At step k, we generate only the vectors $T^{a^ko^k}T^{a^{k-1}o^{k-1}} \dots T^{a^1o^1}R$ such that $T^{a^{k-1}o^{k-1}} \dots T^{a^1o^1}R$ are already in F. This process is repeated until no more independent vectors can be added to F. The basis matrix F is used as a transformation matrix for mapping belief states into reduced belief states and planning within the reduced space.

By noticing that for any test $q = a_1o_1 \dots a_ko_k$ we have $P(.,q) = T^{a_1o_1} \dots T^{a_ko_k}R$, one can see that F is the same outcome matrix defined in Section 3.5, hence, what the VDC algorithm finds is exactly the reward model of a PSR. As for the observation model, one can show that the PSR transformation matrix U (a $|S| \times |Q|$ matrix, Figure 2.4) is in fact a basis of the Krylov subspace $Kr(\{T^{a,o}\}, e_{|\mathcal{S}|})$, where $e^T_{|\mathcal{S}|} = (1, 1, \dots, 1)$. In fact, the vector containing the probabilities of the empty test starting from different states is given by $Pr(\emptyset) = U(., \emptyset) = e_{|\mathcal{S}|}$, and for any test q, $Pr(q) = U(., q) = T^{a_1o_1} \dots T^{a_ko_k}e_{|\mathcal{S}|}$.

VDC algorithm can make predictions about future rewards, but cannot update the reduced belief states after perceiving an observation o, thus it cannot be used for online planning for example. Poupart [2005] showed that this problem can be overcome by considering the Krylov subspace $Kr(\{T^{a,o}\}, R, e_{|\mathcal{S}|})$ instead of $Kr(\{T^{a,o}\}, R)$. In this case, some parts of the belief state will correspond to probabilities of tests, and other parts will correspond to expected rewards. Unless the rewards are normalized to values between 0 and 1, mixing rewards and probabilities can be problematic when it comes to define the loss function of the compression. In our approach, we keep the reward and observation models separated, and define a different loss function for each one of them, as we show in the next section.

3.7 Lossy compression of POMDPs with PSRs

Exact linear transformation is considered insufficient in practice. This is a motivation to further investigate a lossy transformation which scales better. Building on the lossy compression version of VDC, we develop an algorithm for finding compact PSRs. Given a POMDP model and the required reduced dimensions $|Q|$ and $|P|$, our algorithm finds the best parameters of an approximate PSR model that minimize the loss function, which measures the difference between the predictions of the accurate POMDP model and the predictions of the approximate PSR model. We use here the optimization Equations 3.9 and 3.10 to find the parameters of the approximate PSR.

$$\textit{minimize:} \qquad c_1 \sum_{\forall a \in A, \forall o \in \mathcal{O}} \epsilon_{ao} + c_2 \sum_{a \in \mathcal{A}, o \in \mathcal{O}, q_i \in Q} \epsilon_{aoq_i} \tag{3.9}$$

subject to:

$$\forall a \in A, \forall o \in \mathcal{O} : \|T^{ao}e_{|S|} - Um_{ao}\|_\infty \leqslant \epsilon_{ao}$$

$$\forall a \in A, \forall o \in \mathcal{O}, \forall q \in Q : \|T^{ao}U(., q) - Um_{aoq}\|_\infty \leqslant \epsilon_{aoq}$$

$$\begin{aligned} &\textit{minimize:} && c_3\epsilon_r + c_4 \sum_{a\in\mathcal{A},o\in\mathcal{O},p_i\in P} \epsilon_{aop_i} \\ &\textit{subject to:} \\ & && \|R - Fw\|_\infty \leqslant \epsilon_r \\ &\forall a \in \mathcal{A}, \forall o \in \mathcal{O}, \forall p_i \in P : && \|T^{ao}F(.,p_i) - Fw_{aop_i}\|_\infty \leqslant \epsilon_{aop_i} \end{aligned} \tag{3.10}$$

These latter equations are nonlinear, but can be solved with a linear program by using an iterative technique. To achieve that, we alternate between solving Equation 3.9, using a linear program solver, to find the parameters m_{ao} and m_{aoq_i} while keeping U fixed, and solving the same equation to find U while keeping the parameters m_{ao} and m_{aoq_i} fixed. We do the same thing for the reward model Equation 3.10. In our first algorithm, the matrices U and F are initialized with random values. The parameters c_1, c_2, c_3 and c_4 are the weights (the importance) given to the errors on m_{ao}, m_{aoq_i}, w and w_{aop_i} respectively. Convergence is guaranteed just as argued in [Poupart, 2005] since the objective function decreases at each iteration; however, the resulting fixed point may be local optimum.

Our first experiments with this algorithm showed that a large number of iterations are necessary for converging to an optimum, which is often a local optimum, and one should try several random initializations before getting the global optimum. To alleviate this problem, we propose an improved algorithm (Algorithm 5), where we initialize the matrix U with the first $|Q|$ linearly independent vectors of $Kr(\{T^{a,o}\}, e_{|\mathcal{S}|})$ (step 1). The matrix F is initialized with the first $|P|$ linearly independent vectors of $Kr(\{T^{a,o}\}, R)$ (step 2). These vectors can be found in polynomial time by a depth-first search in the spaces $Kr(\{T^{a,o}\}, e_{|\mathcal{S}|})$ and $Kr(\{T^{a,o}\}, R)$. The linear program solved in step 6 returns the weight vector m_{ao} that minimizes the difference between the probability of observation o after executing action a calculated by the POMDP model as $T^{ao}e$, and by the approximate PSR as Um_{ao}. Similar linear programs are solved in steps 6, 13 and 18 for finding the vectors m_{aoq}, used for predicting the probability of tests aoq, w_{aop_i}, used for predicting the expected final reward of tests aop_i, and w, used for predicting the immediate reward. Notice that if we set $|Q| = |P| = |\mathcal{S}|$ then Algorithm 5 will return an exact PSR model that is equivalent to the original POMDP.

3.8 Analysis of the approximation error

The value function V^π of a policy π is represented by an $|\mathcal{S}|$-dimensional vector α^π, such that $\alpha^\pi(s)$ is the expected cumulated reward if we start executing π from state s (see Section 3.2). Given a POMDP belief state b_t, the value of policy π is $b^T\alpha^\pi$. Similarly, a reduced value function $\tilde{\alpha}^\pi$ is a $|P|$-dimensional vector defining the value of π over the reduced belief space.

Input: A POMDP: $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R \rangle$. A number of core tests $|Q| \leqslant |\mathcal{S}|$. A number of reward core tests $|P| \leqslant |\mathcal{S}|$;

Output: The parameters U, F, m_{ao}, m_{aoq_i}, w, w_{aop_i} of the compressed model, $\forall a \in \mathcal{A}, \forall o \in \mathcal{O}, \forall q_i \in Q, \forall p_i \in P$;

Construct the matrix U using the $|Q|$ first linearly independent vectors of $Kr(\{T^{a,o}\}, e_{|\mathcal{S}|})$, and let Q be the set of tests corresponding to these vectors;
Construct the matrix F using the $|P|$ first linearly independent vectors of $Kr(\{T^{a,o}\}, R)$, and let P be the set of tests corresponding to these vectors;
for $a \in \mathcal{A}$ **do**
  for $o \in \mathcal{O}$ **do**
    Find m_{ao} by solving the following linear program:
    minimize ϵ_{ao} *s.t.* $\|T^{ao}e - Um_{ao}\|_\infty \leqslant \epsilon_{ao}$
    for $q_i \in Q$ **do**
      Find m_{aoq_i} by solving the following linear program:
      minimize ϵ_{aoq_i} *s.t.* $\|T^{ao}U(., q_i) - Um_{aoq_i}\|_\infty \leqslant \epsilon_{aoq_i}$
    end
    for $p_i \in P$ **do**
      Find w_{aop_i} by solving the following linear program:
      minimize ϵ_{aop_i} *s.t.* $\|T^{ao}F(., p_i) - Fw_{aop_i}\|_\infty \leqslant \epsilon_{aop_i}$
    end
  end
end
Find the vector w by solving the following linear program:
minimize ϵ_r *s.t.* $\|R - Fw\|_\infty \leqslant \epsilon_r$

Algorithm 5: Transforming a POMDP into an Approximate Compact PSR.

Given a PSR reward state $\tilde{r}_t$, the value of policy π can be written as:

$$\begin{aligned}
b^T\alpha^\pi &= \sum_{t=0}^{H}\gamma^t \sum_{q\in\{\mathcal{A}\times\mathcal{O}\}^t} Pr(q|h_t)\sum_{s\in\mathcal{S}} Pr(s|h_t q)R(s) \\
&= \sum_{t=0}^{H}\gamma^t \sum_{q\in\{\mathcal{A}\times\mathcal{O}\}^t} r(q|h_t) \text{ (From Equation 3.4)} \\
&= \sum_{t=0}^{H}\gamma^t \sum_{q\in\{\mathcal{A}\times\mathcal{O}\}^t}\sum_{p_i\in P} r(p_i|h_t)w_q(p_i) \\
&= \sum_{p_i\in P} r(p_i|h_t)\sum_{t=0}^{H}\gamma^t \sum_{q\in\{\mathcal{A}\times\mathcal{O}\}^t} w_q(p_i) \\
&= \tilde{r}^T\tilde{\alpha}^\pi
\end{aligned}$$

where $\tilde{\alpha}^\pi(p_i) \overset{def}{=} \sum_{t=0}^{T}\gamma^t \sum_{q\in\{\mathcal{A}\times\mathcal{O}\}^i} w_q(p_i)$, and H is a planning horizon, which can be infinite. Since we have $\forall b \in \Delta\mathcal{S} : b^T\alpha^\pi = \tilde{r}^T\tilde{\alpha}^\pi$ and $\tilde{r}^T = b^T F$, then $\alpha^\pi = F\tilde{\alpha}^\pi$ (see the definitions of the reward belief $\tilde{r}$ and the reward compression matrix F given by Equations 3.5 and 3.8).

We assumed in this latter derivation that the expected final reward of any test q is exactly given as a linear combination of the reward core tests. In fact, since the compression that we used is lossy, one cannot guarantee that the equality $\alpha^\pi = F\tilde{\alpha}^\pi$ will be preserved. In the remaining of this section, we provide a bound on the margin between the α-vector α^π of a policy π according to the POMDP model, and the estimated α-vector $F\tilde{\alpha}^\pi$ according to the approximate PSR model, as a function of the errors ϵ_{aop_i} and ϵ_r made by Algorithm 5 (the errors ϵ_{aoq_i} and ϵ_{ao} were not included in this bound). This bound is adapted from the proof of [Poupart, 2005].

Let us denote by Π_t the set of all policies that will be executed at time-step t. Given a policy $\pi \in \Pi_t$, we define ϵ_π as the difference between the value vector α^π of π, according to the POMDP model, and the value vector $F\tilde{\alpha}^\pi$ of the same policy, according to the approximate PSR reward model. Also, let ϵ_t^* be the worst such error over all the policies at time-step t, i.e.

$$\epsilon_t^* = \max_{\pi \in \Pi_t} \epsilon_\pi = \max_{\pi \in \Pi_t} \|\alpha^\pi - F\tilde{\alpha}^\pi\|_\infty$$

We define the reduced transition matrix W^{ao} as $W^{ao}(p_i, aop_j) = w_{aop_j}(p_i)$, for $p_i, p_j \in P$, $a \in \mathcal{A}, o \in \mathcal{O}$. We also use $A(\pi)$ to denote the first action of policy π, and $P(\pi, o)$ the remaining policy of π after executing its first action $A(\pi_t)$ and receiving the observation $o \in \mathcal{O}$. The error ϵ_t^* can be bounded as follows:

$$\begin{aligned}
\epsilon_t^* &= \max_{\pi \in \Pi_t} \|(R - Fw) + \gamma \sum_{o \in \mathcal{O}} (T^{A(\pi)o}\alpha^{P(\pi,o)} - FW^{A(\pi_t)o}\tilde{\alpha}^{P(\pi,o)})\|_\infty \text{ (From Equation 3.3)} \\
&\leqslant \max_{a \in \mathcal{A}} \|R - Fw\|_\infty + \gamma \sum_{o \in \mathcal{O}} \max_{\pi \in \Pi_{t+1}} \|T^{ao}\alpha^\pi - FW^{ao}\tilde{\alpha}^\pi\|_\infty \\
&\leqslant \epsilon_r + \gamma \| \sum_{o \in \mathcal{O}} \max_{\pi \in \Pi_{t+1}} (T^{ao}\alpha^\pi - T^{ao}F\tilde{\alpha}^\pi - FW^{ao}\tilde{\alpha}^\pi + T^{ao}F\tilde{\alpha}^\pi)\|_\infty \\
&\leqslant \epsilon_r + \gamma \| \sum_{o \in \mathcal{O}} \max_{\pi \in \Pi_{t+1}} (T^{ao}\alpha^\pi - T^{ao}F\tilde{\alpha}^\pi)\|_\infty + \gamma \sum_{o \in \mathcal{O}} \max_{\pi \in \Pi_{t+1}} \|T^{ao}F\tilde{\alpha}^\pi - FW^{ao}\tilde{\alpha}^\pi\|_\infty \\
&\leqslant \epsilon_r + \gamma \| \sum_{o \in \mathcal{O}} \max_{\pi \in \Pi_{t+1}} (T^{ao}\alpha^\pi - T^{ao}F\tilde{\alpha}^\pi)\|_\infty + \gamma \sum_{o \in \mathcal{O}} \max_{\pi \in \Pi_{t+1}} \|T^{ao}F - FW^{ao}\|_\infty \|\tilde{\alpha}^\pi\|_\infty \\
&\leqslant \epsilon_r + \gamma \epsilon_{t+1}^* \| \sum_{o \in \mathcal{O}} T^{ao}\|_\infty + \gamma \sum_{o \in \mathcal{O}} \max_{\pi \in \Pi_{t+1}} \epsilon_{aop} \|\tilde{\alpha}^\pi\|_\infty \\
&\leqslant \gamma \epsilon_{t+1}^* + \gamma |\mathcal{O}| \epsilon_{aop} \|\tilde{\alpha}^*\|_\infty + \epsilon_r \\
&\leqslant \frac{\epsilon_r + \gamma |\mathcal{O}| \epsilon_{aop} \|\tilde{\alpha}^*\|_\infty}{1 - \gamma}
\end{aligned}$$

where $\|\tilde{\alpha}^*\|_\infty = \max_{\pi \in \bigcup_{t=0}^{\infty} \Pi_t} \|\tilde{\alpha}_\pi\|_\infty$. In these substitutions, we used the following properties: $\|A + B\|_\infty \leqslant \|A\|_\infty + \|B\|_\infty$ and $\|AB\|_\infty \leqslant \|A\|_\infty \|B\|_\infty$.

3.9 Empirical evaluation

3.9.1 Online prediction

The main purpose of a dynamical system model is to predict the probabilities of different observations, and the expected immediate reward, given an arbitrary history h_t (see Section 2.1). After finding the compressed PSR model of a given problem by the means of Algorithm 5, we test its accuracy by simulating a sequence of H action-observations and compare the predictions of the reduced model to the predictions of the original POMDP. The initial belief state b_0 of the POMDP is randomly generated, and the initial PSR belief state is calculated by $\tilde{b}_0 = b_0^T U$ for the observation model, $\tilde{r}_0 = b_0^T F$ for the reward model. At each time-step t, we sample an action a_t according to a uniform distribution, and calculate the prediction $Pr(o|a_t, h_t)$ by using the exact model and the prediction $\tilde{Pr}(o|a_t, h_t)$ by using the approximate model, for each observation $o \in \mathcal{O}$. We also calculate the expected rewards $R(h_t) = b_t^T R$ and $\tilde{R}(h_t) = \tilde{r}_i^T w$. Then, we sample a next state and an observation o_t according to the accurate model. Action a_t and observation o_t are used by both models to update their internal beliefs, each one using its own parameters and bayesian update function.

The root mean squired error (RMSE) is used to measure the divergence between the predictions of the reduced model and the predictions of the accurate model. It is given by:

$$RMSE_{observation} = \sqrt{\frac{1}{H \times |\mathcal{O}|} \sum_{t=1}^{H} \sum_{o \in \mathcal{O}} (\tilde{Pr}(o|h_t, a_t) - Pr(o|h_t, a_t))^2} \tag{3.11}$$

$$RMSE_{reward} = \sqrt{\frac{1}{H} \sum_{t=1}^{H} (\tilde{R}(h_t) - R(h_t))^2} \tag{3.12}$$

In our experiments, we considered a horizon of $H = 100$, this was sufficient for testing the model since the system was reset to an initial state distribution often before reaching this horizon. The prediction error becomes more important as the horizon grows, this is due to the fact that the bayesian update function (Equations 2.3 and 3.6) of PSR belief states uses approximate probabilities $\tilde{Pr}(o|h, a)$ in the denominator of Bayes' Rules, so that, after a long sequence, the cumulated errors make the PSR belief state significantly inaccurate. As an example of that, if the true probability of an observation is $Pr(o|h, a) = 0.01$ (which can be the case when the number of observations is large) and the estimated probability of the same observation is $\tilde{Pr}(o|h, a) = 0.02$, then even if one uses the true probability $Pr(aoq|h)$ of an extended core test aoq, the estimated posterior probability $\tilde{Pr}(q|hao)$ of test q will be twice its true value, and could be higher than 1. However, the belief states of both POMDP and PSR models are always reset to an accurate value after executing a reset action, or perceiving a landmark (deterministic and non-aliasing) observation.

We tested Algorithm 5 on different standard benchmarks taken from the literature: 4×4 grid (16 states, 4 actions, 2 observations), Shuttle (8 states, 3 actions, 5 observations), Network (7 states, 4 actions, 2 observations), Hallway problem (60 states, 5 actions, 21 observations), Hallway2 problem (92 states, 5 actions, 17 observations) [Cassandra, 1998], Coffee domain (32 states, 2 actions, 3 observations) [Boutilier and Poole, 1996], and Spoken Dialogue Management (433 states, 37 actions, 16 observations) [Williams, 2006].

Table 3.1 presents the average error on predictions (Equations 3.11 and 3.12) as a function of the number of core tests used for each problem. The error on predicting rewards is contained in the interval $[0, 1]$ because it measures a difference between two probabilities. For the Coffee problem, we reported only the error on predicting rewards, the error on predicting observations becomes null by using only two core tests. In fact, most of the states in this problem have similar transition and observation dynamics, they are distinguished only by the reward function. For the other problems, we reported the error on predicting observations, because the error on rewards was almost null even when we use only one core test. In these specific domains, all the states have a null reward, except for the final state. With a random policy, we could rarely get to these final states, so the cumulated reward was null for both PSR and POMDP.

For small problems (Shuttle, Network and 4×4 grid), we used the first version of the compression algorithm (see Section 3.7), where all the parameters of the approximate PSR are found by solving the optimization equations 3.9 and 3.10. In larger domains (Coffee, Hallway, Hallway2 and Spoken Dialogue), this approach became untractable, so we set the columns of U and F to the linearly independent vectors of $Kr(\{T^{a,o}\}, e_{|\mathcal{S}|})$ and $Kr(\{T^{a,o}\}, R)$ and used Algorithm 5 to find the remaining parameters. The effective compression times for large problems are reported in Table 3.2, the experiments were performed using ILOG Cplex 10 solver on an AMD Athlon machine with a 1.80 GHZ processor and 1.5 GB memory.

In all these experiments, the compression of the POMDP was successful, because only a few core tests were necessary to construct an approximate PSR model with a low prediction error. For the 4×4 grid for example, we used only 6 core tests instead of 16 to make predictions on future observations with an average error of 0.10 over the 100 steps. A random prediction makes an average error of 0.55 in most of the problems. For Network problem, we can predict the future observations of the system by using only 3 tests, while the prediction error is nearly null (0.02). For the Coffee domain, the PSR model is able to predict the expected reward with an almost null error, by using 6 core tests. This gain is even more pronounced in larger domains, as like Hallways and Spoken Dialogue. For this latter domain, the prediction error was almost null with a very small number of core tests. This is due to the fact that the observations depend only on the initial belief state, which is a uniform distribution over states (In this problem, the final state is reached after only 3 actions in general). A Kullback-Leibler (KL) divergence would be a better measure of the error in this case, because the distribution on observations is always almost uniform, and the probabilities of the observations are too small (around 0.06 for each observation).

Domain	Original dimension	Reduced dimension	Compression Ratio	Root Mean Squared Error (RMSE)
Network ($RMSE_{observation}$)	7	1	15%	0.1575
		2	30%	0.1224
		3	45%	0.0159
		4	60%	0.0228
		5	75%	0.0029
		6	90%	0.0067
Shuttle ($RMSE_{observation}$)	8	1	12.5%	0.3867
		2	25%	0.3966
		3	37.5%	0.3920
		4	50%	0.3877
		5	62.5%	0.2851
		6	75%	0.1479
Grid ($RMSE_{observation}$)	16	1	6.25%	0.4595
		2	12.5%	0.3052
		3	18.75%	0.1155
		4	25%	0.1445
		5	31.25%	0.1349
		6	37.5%	0.1009
Coffee ($RMSE_{reward}$)	32	1	3%	1.3307
		2	6%	1.3316
		3	9%	0.3749
		4	12%	0.1065
		5	15%	0.1065
		6	18%	0.0027
Hallway ($RMSE_{observation}$)	60	5	8%	0.1234
		10	17%	0.0572
		15	25%	0.0585
		20	33%	0.0590
		25	42%	0.0590
Hallway2 ($RMSE_{observation}$)	92	5	6%	0.1272
		10	12%	0.1361
		15	17%	0.0947
		20	22%	0.0392
		25	27%	0.0369
Dialogue ($RMSE_{observation}$)	433	2	$< 1\%$	0.0279
		3	$< 1\%$	0.0255
		5	$\approx 1\%$	0.0255
		8	$\approx 2\%$	0.0000
		10	$\approx 2\%$	0.0000

Table 3.1: Average error of the approximate PSR on predicting observations, as function of the reduced dimension.

Domain	Original dimension	Reduced dimension	Compression ratio	Runtime in seconds
Hallway, observations	60	5	8%	20.59
		10	17%	35.42
		15	25%	60.87
		20	33%	74.47
		25	42%	94.93
		30	50%	114.9
Hallway2, observations	92	5	6%	20.46
		10	12%	41.92
		15	17%	69.46
		20	22%	107.28
		25	27%	157.73
		30	34%	184.23
Dialogue, observations	433	2	$< 1\%$	350.93
		3	$< 1\%$	557.09
		5	≈ 1 %	759.20
		8	≈ 2 %	1166.04
		10	≈ 2 %	1459.56
Coffee, observations	32	1	3%	0.14
		2	6 %	0.20
Coffee, reward	32	1	3%	0.22
		2	6%	0.36
		3	9%	0.48
		4	12%	0.78
		5	15%	0.95
		6	18%	0.98
Coffee, total (observations and reward)	32	1+2	9%	0.36
		2+2	12%	0.56
		3+2	15%	0.68
		4+2	18%	0.98
		5+2	21%	1.15
		6+2	24%	1.18

Table 3.2: Runtime of the compression algorithm in seconds, as function of the reduced dimension.

We also notice that in general, the average error decreases as the number of core tests increases, because with more variables, the parameters of the model can be better adjusted in order to minimize the loss function. The error becomes always null when $|Q| = |P| = |\mathcal{S}|$, since in this case, Algorithm 5 will be equivalent to the algorithm of Littman et al. [2001], which returns an exact PSR model.

3.9.2 Online planning

Decision-making is the other important purpose of using models of dynamical system. We tested our approach on the problem of online planning (see Section 2.1). To do so, we implemented RTBSS (Real-Time Belief Space Search) algorithm [Paquet et al., 2005], and adapted it for dealing with PSRs. RTBSS is based on a forward search (Algorithm 4) that is performed online each time the agent has to make a decision, i.e. after each observation. As an online method, RTBSS is particularly appealing for large real-time systems where offline methods are not applicable because of their computational complexity [Roy and He, 2009]. In our case, we set the search depth to 3, and used Blind Policy as lower bound heuristic and Q-MDP as upper bound (see Section 3.4.1). The adaptation of RTBSS for dealing with PSRs is almost straightforward, since we already provided a mechanism for updating reduced belief states, and calculating the expected reward corresponding to such belief states. However, Blind Policy and Q-MDP heuristics are functions of states and it is not obvious how to use these heuristics in PSRs, where the states are replaced by core tests. Nevertheless, the values of heuristics can be treated in the same way we treated immediate rewards, and preserved during the compression. Let $\bar{V}$ and $\underline{V}$ be two $|\mathcal{S}|$-dimensional vectors such that $\bar{V}(s)$ is an upper bound on the value of state s and $\underline{V}(s)$ is a lower bound on the value of state s. The bounds $\underline{V}$ and $\bar{V}$ are calculated offline by using the provided POMDP model. Let $\bar{v}$ and $\underline{v}$ be two $|P|$-dimensional vectors such that $\tilde{r}_t^T \bar{v}$ and $\tilde{r}_t^T \underline{v}$ are respectively an upper and a lower bounds on the value of the PSR reward belief state $\tilde{r}_t$. The vectors $\bar{v}$ and $\underline{v}$ are found by adding solving the following linear programs at the end of Algorithm 5, where F is the outcome matrix of the reward core tests (defined by Equation 3.8):

$$\begin{aligned} &\textit{minimize } \epsilon_{\underline{V}} \;\; \textit{s.t. } \|\underline{V} - F\underline{V}\|_\infty \leqslant \epsilon_{\underline{V}} \\ &\textit{minimize } \epsilon_{\bar{v}} \;\; \textit{s.t. } \|\bar{V} - F\bar{v}\|_\infty \leqslant \epsilon_{\bar{v}} \end{aligned}$$

Table 3.3 illustrates the reward and runtime per time-step of RTBSS using both POMDP and Approximate PSR models, as a function of the reduced dimension. These values are averaged over 100 time-steps. For the Coffee domain, we can see that we need at most two observation core tests and two reward core tests to get the same average reward as the exact POMDP model, while the runtime per step is significantly reduced. The results for Hallways are also very promising. In fact the runtime is always reduced and the average reward is close to the reward of POMDP. Note that for Hallway2, we apparently need higher dimensions in order to achieve the same results as with the exact POMDP.

	RTBSS with Compressed PSR				RTBSS with POMDP	
Domain	$\|Q\|$	$\|P\|$	Runtime per step	Average reward	Runtime per step	Average reward
Coffee	1	1	0.002	-1.99	0.35	-1.49
	2	2	0.007	-1.49		
Hallway	20	20	15.89	0.03	27.58	0.06
(60 states)	40	40	20.70	0.06		
Hallway2	20	20	1.56	0.00	80.20	0.02
(92 states)	40	40	5.70	0.01		

Table 3.3: Average reward and runtime in milliseconds of RTBSS using POMDP and Approximate PSR models, as function of the reduced dimension.

3.10 Conclusion

Current work on planning methods for POMDPs demonstrates that finding optimal policies for POMDPs is untractable in practice. In this thesis, we investigated the possibility of finding an approximate reduced PSR model from a given larger POMDP. We formulated this problem in linear programming framework. We illustrated a theoretical bound for value function error using the approximate PSR model. Using an approximate model seems to have a definite advantage in PSRs as confirmed by our experimental results. Our results show that the reduced model can predict the probabilities of observations and the rewards with a high precision compared to the exact model. The impact of this method is more pronounced for problems with special structure. However, there are more potential advantages in applying reduced PSRs instead of POMDPs for planning. Preliminary results on using approximate PSR in online planning show that we can always find a reduced dimension where planning with approximate PSR takes less time than with the original POMDP, without an important loss of performance. The immediate next step to this research is to perform more elaborated experiments on using the approximate PSR in planning, with larger domains, and to compare with other compression methods. The other extension is to study how to efficiently select the most promising independent vectors of Krylov subspaces instead of simply taking the first ones. Indeed, this problem, known as the *discovery problem*, remains one of the major unsolved problems in PSRs [Makino and Takagi, 2008].

Chapter 4

Policy Space Compression in Decentralized POMDPs with Predictive Representations

The rise of applications requiring cooperation between different agents, like robotic teams, distributed sensors and communication networks, has made coping with the presence of other decision makers a key challenge for building autonomous agents. For this purpose, a generalization of POMDPs to multi-agent systems, called DEC-POMDPs (decentralized POMDPs), was introduced by Bernstein et al. [2002], and since then, this framework has been receiving a growing amount of attention [Rabinovich et al., 2003; Becker et al., 2004; Emery-Montemerlo et al., 2004; Hansen et al., 2004; Szer et al., 2005; Szer and Charpillet, 2006; Seuken and Zilberstein, 2007a; Amato et al., 2007a; Dibangoye, 2010]. This research is basically motivated by the fact that many real world problems need to be casted as decentralized POMDPs, while finding an optimal solution for a decentralized POMDP is a NEXP-hard problem [Bernstein et al., 2002], and even finding ϵ-optimal solutions is NEXP-hard [Rabinovich et al., 2003]. Finding good solutions to decentralized POMDPs is so difficult because there is no optimality criterion for the policies of a single agent alone: whether a given policy is better or worse than another depends on the behavior of the remaining agents. Moreover, no agent is aware of the actual behavior of the remaining agents unless the agents can exchange their local information through communication, which is not always possible. Instead, each agent keeps a belief state on the policies of the other agents based on its local actions and observations received so far. In this chapter, we address the problem of finding compact representations of belief states on policies by using predictive representations of policies. To do so, we propose an adaptation of PSRs for dealing with uncertainty on policies. We also present an empirical evaluation on benchmark problems, illustrating the performance of this approach.

4.1 Introduction

Partially Observable Markov Decision Processes (POMDPs) provide a general framework for dealing with state uncertainty. However, when a system is controlled by more than one agent, Markov property becomes no longer true, unless the state of the system includes information about the policy (or behavior) of each agent involved in the process. Decentralized POMDPs have emerged in recent years as a natural extension of POMDPs for planning under uncertainty in multi-agent systems. In multi-agent systems, each agent faces different types of uncertainties:

Uncertainty with respect to the state of the system: As in POMDPs, the state of the system is partially observable, every agent receives a local observation at each time-step and uses it to update its local belief state.

Uncertainty with respect to the observations of other agents: The local observations received by different agents are not necessarily identical, each agent observes the state of the system through its own sensors. Therefore, in order to choose an optimal policy, each agent should take into account the probabilities of all the possible observations that may have been received by the other agent. In practice, this problem can be overcome by means of instantaneous cost-free and noise-free communication, in which case the problem reduces to a POMDP [Pynadath and Tambe, 2002]. Decentralized POMDPs deal with the more general case, where agents interact by means of actions and observations, sending and receiving messages can be considered as special actions and observations.

Uncertainty with respect to the actions of other agents: Even if an agent knows precisely the local observations of other agents, it cannot predict their actions by using only this information. In fact, the actions that will be executed by the other agents depend on many factors. The first factor is the rationality of the agents. A rational agent is one that acts optimally in order to maximize its payoff in the limit of its computational capacity. In Decentralized POMDPs, all the agents are assumed to be rational. The second factor is the payoff function minimized by each agent. Having different payoff functions may lead to adversary situations where agents can choose persuasive actions in order to mislead other agents. This type of problems constitutes the topic of Game Theory. In decentralized POMDPs, the agents are assumed to be cooperative and share the same payoff function. Finally, the actions of other agents may be unknown simply because their planning algorithms are unknown. In decentralized POMDPs, the same planning algorithm with similar parameters is executed by all the agents.

4.1.1 Example

Figure 4.1 represents a simple multi-agent system with two robots, *Walle* and *Eve*, navigating in a 3×3 grid world, and choosing their actions from the set: *Move right* ($\rightarrow$), *Move left* ($\leftarrow$), *Move up* ($\uparrow$), *Move down* ($\downarrow$), and receiving as observations *Obstacle: The robot hit a wall*, or *Nothing: No wall in sight*.

If the goal of the two robots is to meet as soon as possible anywhere on the grid, then each robot must know the policy of the other one in order to find the best policy leading to a meeting state. Since the policy of each robot depends on its local history of actions and observations, and none of them knows anything about the history of the other, each robot should use its local history to infer a probability distribution on the histories of the other one, and consequently, a distribution on the remaining policies of the other one. The planning problem is solved using this distribution, as we will show in Section 4.2.

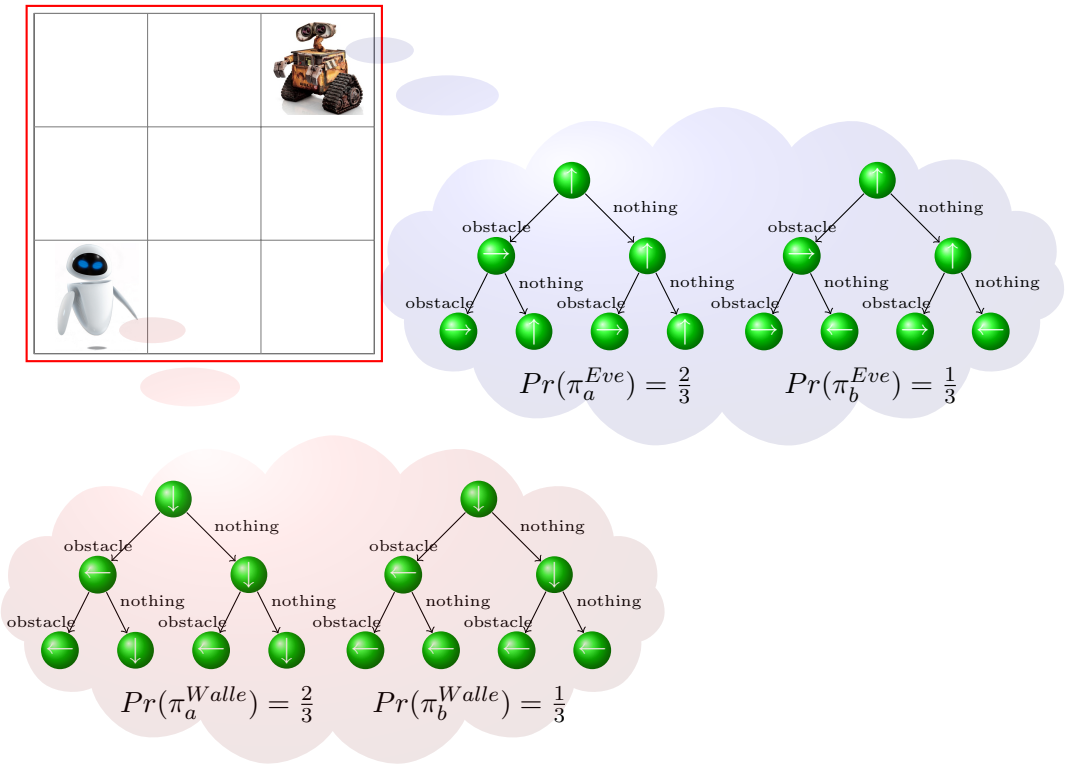


Figure 4.1: Meeting under uncertainty.

4.1.2 Definition of Decentralized POMDPs

Formally, a decentralized POMDP with n agents is a tuple $\langle I, \mathcal{S}, \{\mathcal{A}^i\}, \{\mathcal{O}^i\}, T, Z, R, \gamma, H\rangle$, where:

- I is a finite set of agents, indexed $1 \dots n$.
- $\mathcal{S}$ is a finite set of states.
- $\mathcal{A}^i$ is a finite set of individual actions for an agent i. $\vec{\mathcal{A}} = \otimes_{i\in I}\mathcal{A}^i$ is the set of joint actions, and $\vec{a} = \langle a^1, \dots, a^n\rangle$ denotes a joint action.
- T is a transition function. $T(s_t, \vec{a}_t, s_{t+1})$ returns the probability of transitioning to state s_{t+1} after the agents execute the joint action $\vec{a}_t$ in state s_t.
- $\mathcal{O}^i$ is a finite set of individual observations for an agent i. $\vec{\mathcal{O}} = \otimes_{i\in I}\mathcal{O}^i$ is the set of joint observations, and $\vec{o} = \langle o^1, \dots, o^n\rangle$ denotes a joint observation.
- Z is an observation function, $Z(\vec{o}|s_t, \vec{a})$ is the probability that the agents observe $\vec{o}$ when the current state is s_t and the joint action that led to this state was $\vec{a}$.
- R is a reward function, where $R(s, \vec{a})$ denotes the reward (or cost) that all the agents receive for executing joint action $\vec{a}$ in state s.
- H is the horizon of planning (the total number of time-steps).
- $\gamma \in (0, 1]$ is a discount factor.

4.1.3 Policies in decentralized POMDPs

Planning algorithms for decentralized POMDPs aim to find the best joint policy for a given horizon H, which is a collection of several local policies, one for each agent. A local policy for agent i, denoted by π^i, is a function mapping every history of local actions and observations $h^i = a^i_0 o^i_0 a^i_1 o^i_1 \dots a^i_t o^i_t$, for $k < H$, to an action in $\mathcal{A}^i$:

$$\begin{aligned} \pi^i : \quad \bigcup_{t=0}^{H-1} (\mathcal{A}^i \times \mathcal{O}^i)^t \quad &\rightarrow \quad \mathcal{A}^i \\ h^i \quad &\mapsto \quad a^i \end{aligned}$$

Usually, local policies are represented by decision trees, each node of the decision tree is labeled with an individual action a^i, and each arc is labeled with an individual observation o^i. For the sake of simplifying notations, we consider in the remainder of this chapter that we have only two agents, i and j, all the results can be easily extended to the general case. A joint policy for agents i and j is denoted by $\pi = \langle \pi^i, \pi^j\rangle$. We also use Π^i, Π^j to indicate the sets of local policies for agents i and j respectively, and Π for the set of joint policies. We also use the conventional notation $-i = j$ and $-j = i$.

4.1.4 Multiagent belief states

Since the states are partially observable, agents should choose their actions according to their belief about the current state of the system, as well as the actions (or policies) that the other agents are going to execute. Ideally, every agent should know exactly the actions of the other agents, but unfortunately, this cannot be achieved without the use of communication. In fact, the joint policy is provided to all the agents, and the first joint action can easily be predicted since it depends only on the initial belief state, which is the same for all the agents. But then, the next actions depend on the local observations perceived by each agent, and without communication, no agent knows what the other agents have received as observation, and therefore, no agent knows which policies the other agents will follow. Instead, each agent can calculate a probability distribution on the possible observations that might have been received by the others. Since the joint policy maps each sequence of observations into an action, the probability of executing a given action at a given time-step t is simply the sum of the probabilities of all the sequences of observations that are mapped to this action, i.e.

$$Pr(a^i_{t+1} = a^i | h^j) = \sum_{\substack{h^i \in (\mathcal{A}^i \times \mathcal{O}^i)^t \\ s.t.\ \pi^i(h^i) = a^i}} Pr(h^i | h^j)$$

This distribution on the policies (or the remaining subtrees) of the other agents is called a *multiagent belief state*. The belief state b^i for agent i contains a probability distribution on the states $\mathcal{S}$, and another probability distribution on the current policies Π_j of agent j. Formally, a multi-agent belief state for agent i is an $|\mathcal{S}| \times |\Pi^j|$ matrix, where $b^i(s, \pi^j)$ is the probability that the system is in state s and the current policy of agent j is π^j. In the example shown in Figure 4.1, the current belief state of Walle is a probability distribution on the 9×9 possible locations of him and Eve, as well as a distribution on two possible policies of Eve.

4.1.5 Multiagent value functions

This notion of multiagent belief state is used for defining the value function of a policy, and the concepts of *dominant* and *dominated* policies. The expected discounted reward of a joint policy π, started from state s, is given by Bellman value function:

$$V^\pi(s) = R(s, A(\pi)) + \gamma \sum_{s' \in \mathcal{S}} Pr(s' | s, A(\pi)) \sum_{\vec{o} \in \vec{\mathcal{O}}} Pr(\vec{o} | s', \vec{A}(\pi)) V^{P(\pi, \vec{o})}(s') \tag{4.1}$$

where $A(\pi)$ is the first joint action of policy π (the action given by the root node), and $P(\pi, \vec{o})$ is the sub-policy of π remaining after executing the first joint action $A(\pi)$ and receiving joint observation $\vec{o}$. The value of an individual policy π^i, according to a belief state b^i, is defined as:

$$V^{\pi^i}(b^i) = \sum_{s \in \mathcal{S}} \sum_{\pi^j \in \Pi^j} b^i(s, \pi^j) V^{\langle \pi^i, \pi^j \rangle}(s) \tag{4.2}$$

where $b^i(s, \pi^j)$ is the probability that the system is in state s and the current policy of agent j is π^j, and $\langle \pi^i, \pi^j \rangle$ denotes the joint policy made up of π^i and π^j.

A policy π^i is said to be *weakly dominant* if and only if there exists a belief state such that the value of policy π^i with that belief is higher than or equal to the value of any other policy:

$$\exists b^i \in \Delta(\mathcal{S} \times \Pi^j), \forall \pi'^i \in \Pi^i - \{\pi^i\} : V^{\pi^i}(b^i) \geqslant V^{\pi'^i}(b^i) \tag{4.3}$$

If this inequality is strict, then π^i is said to be *strictly dominant*. Similarly, a policy π^i is said to be *weakly dominated* (resp. *strictly dominated*) if it is not weakly dominant (resp. strongly dominant).

4.2 Planning with decentralized POMDPs

In this section, we provide an overview of planning algorithms for decentralized POMDPs. Most of these algorithms are not necessary for understanding the remaining of this chapter. Our work builds on two of planning algorithms, Dynamic Programming and Point Based Dynamic Programming, which are described in further details in the second and third subsections.

4.2.1 Literature review

A brute force solution to decentralized POMDPs consists in performing an exhaustive search in the space of all possible joint policies for a given horizon [Bernstein et al., 2002], but this approach is almost useless in practice, even for the smallest domains. In fact, the search should focus only on dominant policies. There are two main approaches for finding these policies: top-down heuristic search and bottom-up dynamic programming. Multiagent A* (MAA*) was the first algorithm to use a top-down heuristic [Szer and Charpillet, 2006]. It is an adaptation of the popular A* algorithm where the nodes contain the joint policies, and uses heuristic evaluation functions (based on the underlying MDP or POMDP) to prune some parts of the search space.

Generalized MAA* (GMAA*) is an extension of MAA* proposed by Oliehoek et al. [2008] where a multitude of heuristic functions are considered. More specifically, Oliehoek et al. [2008] introduced a new optimistic heuristic based on approximating the decentralized POMDP by a series of bayesian games. Point Based Incremental Pruning (PBIP) is a branch and bound search heuristic using the value function of the underlying POMDP as an upper bound [Dibangoye et al., 2009a]. The main difference between PBIP and MAA* is that PBIP improves the joint-policy one fringe at a time, whereas MAA* improves the joint-policy one time-step at a time. Although (G)MAA* and PBIP are optimal, an inherent disadvantage of all top-down

search approaches is that the starting point (the initial belief state) should be known in advance, otherwise, a different search should be performed for every possible initialization of the problem.

Along the same lines, Varakantham et al. [2007] used heuristic search where the nodes of the search tree correspond to different agents instead of histories. A graph of interactions between the agents is provided to the algorithm, and a near-optimal policy is found for each agent by iteratively considering the policies of its neighbors, in an order defined by the search tree. Despite the important scalability in the number of agents demonstrated by this algorithm, only a small class of decentralized POMDPs can be decomposed in such a way. In most cases, the degree of interaction between different agents is contextual, depending on the state of the system.

The Dynamic Programming (DP) algorithm, proposed by Hansen et al. [2004], consists in constructing the dominant policies from leaves to root by using iterated strategy elimination techniques taken from Game Theory. The DP algorithm can solve problems that are unfeasible with exhaustive search, but it keeps all the dominant policies for every point in the belief space, even those that will never be reached in practice. This problem has been efficiently addressed by the Point Based Dynamic Programming (PBDP) algorithm, proposed by Szer and Charpillet [2006]. PBDP makes use of top-down heuristic search to determine which belief points will be reached during the execution time, and constructs dominant policies from leaves to root with DP, by keeping only the policies that are dominant in the reachable belief states. Therefore, the runtime of this algorithm is significantly low compared to the original DP algorithm. An approximate version of this algorithm consists in considering only a small set of belief points that are reachable with high probability [Szer and Charpillet, 2006]. Memory Bounded Dynamic Programming (MBDP) is another approximation algorithm, close to PBDP, that bounds the maximum number of policies kept in memory after each iteration [Seuken and Zilberstein, 2007a,b]. However, both PBDP and MBDP work only when the initial belief state is known.

All the algorithms described above are based on reducing the number of policies to be evaluated or kept in memory. An interesting alternative is to preserve the original policy space, and to use a more compact representation of those policies. Finite-State Machines (FSM) are a well-known graphical model, used in POMDPs to represent infinite-horizon policies [Hansen, 1998]. An extension of FSMs to decentralized POMDPs has originally been proposed in [Bernstein et al., 2007], it makes use of a correlation device to synchronize the actions of different agents, and finds the optimal joint controller, for a given number of states, by iteratively improving local machines. However, this approach does not guarantee finding the global optimal joint machines, and can get stuck in a local optimum. Amato et al. [2007b] showed how to find an optimal joint machine without using a correlation device, but the proposed approach involves solving quadratic programs where the number of variables is a function of the number of states in the machine, which is computationally costly.

Decision tree policies are constructed by combining multiple sequences of actions and observations, and the same sequences can be replicated in different policies. The sequential representation [Koller et al., 1994; Aras et al., 2007] takes advantage of this characteristic: all the possible sequences of a given length are represented explicitly, and each policy is represented by a binary weight vector that indicates which sequences are contained in this policy. This method has been efficiently applied to decentralized POMDPs by Aras et al. [2007] in order to reduce the memory storage requirement. The proposed algorithm considers all the possible policies for a given horizon, and uses one Mixed Integer Linear Program (MILP) to find the non-dominated policies, where each variable corresponds to a sequence. However, there is no guarantee that the number of sequences will not exceed the number of candidate policies considered in dynamic programming, this can happen when we have a small number of dominant policies and a large planning horizon. Nevertheless, by noticing that a sequence of actions and observations is nothing else than a test, and that a policy can be seen as an internal state, one can take advantage of known properties of predictive representations, and perform a dimensionality reduction in the space of sequences, as we will show in Section 4.3.

4.2.2 Dynamic Programming

Dynamic Programming [Bellman, 1957] is by far the most used technique for solving multistage decision problems, where the non-dominated policies of time-step t are recursively constructed from the non-dominated sub-policies of time-step $t+1$. The main advantage of dynamic programming, compared to top-down methods, comes from the fact that the sub-policies of time-step $t+1$ are overlapping, i.e. they can be reused for several belief states at time-step t. This method has been widely used for finding finite-horizon policies since the introduction of the value iteration algorithm for POMDPs by Smallwood and Sondik [1971]. An interesting extension of the value iteration algorithm to decentralized POMDPs, called Dynamic Programming (DP) Operator for Decentralized POMDPs, was proposed by Hansen et al. [2004]. We review here the principal steps of this algorithm.

The Dynamic Programming Operator (Algorithm 6) is used to find the dominant policies of time-step t (horizon $H-t$), given the dominant policies of time-step $t+1$ (horizon $H-(t+1)$), for $t = 0, \ldots, H$. We denote a time-step t policy for agent i by π_t^i, and the set of time-step t policies by Π_t^i. The multiagent value function of a joint policy $\langle \pi_t^i, \pi_t^j \rangle$ can be represented by an $|\mathcal{S}|$-dimensional vector $V^{\langle \pi_t^i, \pi_t^j \rangle}$, we denote by V^{Π_t} the set of time-step t value vectors. First, the sets Π_t^i and Π_t^j are generated by exhaustively extending the policies of the sets Π_{t+1}^i and Π_{t+1}^j, i.e. $\Pi_t^i = \{\pi_t^i : (A(\pi_t^i) \in \mathcal{A}_i) \wedge (P(\pi_t^i, o^i) \in \Pi_{t+1}^i, o^i \in \mathcal{O}^i)\}$ (and the same thing for agent j), this operation is called *full backup*. The value vectors of the set V^{Π_t} are calculated by using the value vectors of $V^{\Pi_{t+1}}$ in Equation 4.1. Notice that for $t = H$, the sets Π_H^i and Π_H^j correspond to $\mathcal{A}^i$ and $\mathcal{A}^j$ respectively, and the value vectors of V^{Π_H} are initialized with the immediate rewards. The main loop of the algorithm consists in iteratively

pruning the weakly dominated policies of each agent. These later policies are found by solving Equation 4.3, using the following linear program [Hansen et al., 2004]:

$$
\begin{aligned}
&\textit{minimize:} \quad \epsilon \\
&\textit{subject to:} \\
&\qquad \sum_{s \in \mathcal{S}} \sum_{\pi^j \in \Pi_t^j} b_t^i(s, \pi^j) = 1 \\
&\qquad \forall s \in \mathcal{S}, \forall \pi^j \in \Pi_t^j : \\
&\qquad\qquad 0 \leqslant b_t^i(s, \pi^j) \leqslant 1 \\
&\qquad\qquad \forall \pi'^i \in \Pi_t^i - \{\pi^i\} : \\
&\sum_{s \in \mathcal{S}} \sum_{\pi^j \in \Pi_t^j} b_i(s, \pi^j)[V^{\langle \pi^i, \pi^j \rangle}(s) - V^{\langle \pi^i, \pi'^j \rangle}(s)] + \epsilon > 0
\end{aligned}
\tag{4.4}
$$

The pruning process stops when no more policies can be removed from Π_t^i or Π_t^j.

Input: Π_{t+1}^i, Π_{t+1}^j and $V^{\Pi_{t+1}}$;
1 $\Pi_t^i \leftarrow fullBackup(\Pi_{t+1}^i)$;
2 $\Pi_t^j \leftarrow fullBackup(\Pi_{t+1}^j)$;
3 Calculate the value vectors V^{Π_t} by using $V^{\Pi_{t+1}}$(Equation 4.1);
4 **repeat**
5 remove the policies of Π_t^i that are dominated (Equation 4.4);
6 remove the policies of Π_t^j that are dominated (Equation 4.4);
7 **until** *no more policies in* Π_t^i *or* Π_t^j *can be removed* ;
Output: Π_t^i,Π_t^j and V^{Π_t};

Algorithm 6: Dynamic Programming for Decentralized POMDPs [Hansen et al., 2004].

From Algorithm 6, we can see that the Dynamic Programming operator spends most of its time on looking for weakly dominated policies by checking Inequality 4.3 for every policy π_t^i. This is achieved by solving the linear program given by Equation 4.4. The objective function to be minimized is defined by ϵ, which is the greatest difference between the value of π^i and the value of any other policy π'^i over the belief space. If $\epsilon \geqslant 0$ then the policy π^i is weakly dominated and should be removed, and if $\epsilon < 0$, then there is some region in $\Delta(\mathcal{S} \times \Pi_t^j)$ where π^i is dominant. The variables are ϵ and the probabilities $b(.,.)$ of the multi-agent belief state, so there are $|\mathcal{S}||\Pi_j^t| + 1$ variables. The computational complexity of solving a linear program depends on the number of variables and constraints defined in the problem, so, it directly depends on the number of policies and the way the beliefs on those policies are represented.

4.2.3 Point Based Dynamic Programming

This latter issue, related to the computational complexity of solving linear programs, has been efficiently addressed with Point Based Dynamic Programming (PBDP) algorithm proposed by Szer and Charpillet [2006]. This algorithm (Algorithm 7) keeps only the policies that are dominant in the reachable belief points, denoted by the set $B(\Pi_t^{-k}, h_t^k, \Pi_{H-t})$, for $k \in \{i, j\}$. A necessary assumption for finding these belief points is that the initial belief state is known. The most important computational advantage of PBDP over exact DP comes from the fact that the dominance test is checked only for a finite set of reachable belief points. Therefore, the computational cost of this algorithm is significantly small compared to the original DP algorithm. An approximation version of PBDP consists in considering only a small set of belief points that are reachable with a high probability. Memory Bounded Dynamic Programming (MBDP) [Seuken and Zilberstein, 2007a] is another improved version of PBDP, based on bounding, by a constant, the maximum number of policies kept in memory after each iteration.

Input: Π_{t+1}^i, Π_{t+1}^j and $V^{\Pi_{t+1}}$;
1 $\Pi_t^i \leftarrow fullBackup(\Pi_{t+1}^i)$;
2 $\Pi_t^j \leftarrow fullBackup(\Pi_{t+1}^j)$;
3 Calculate the value vectors V^{Π_t} by using $V^{\Pi_{t+1}}$(Equation 4.1);
4 Generate the set of all possible joint policies Π_{H-t};
5 **repeat**
6 **forall** $k \in \{i, j\}$ **do**
7 **foreach** *history* h_t^k **do**
8 Generate the set of all possible belief states $B(\Pi_t^{-k}, h_t^k, \Pi_{H-t})$ for agent k;
9 Remove the policies of Π_t^i that are weakly dominated;
10 A policy π_t^k is weakly dominated if:
 $\forall b_t^k \in B(\Pi_t^{-k}, h_t^k, \Pi_{H-t}), \exists {\pi'}_t^k \in \Pi_t^k - \{\pi_t^k\} : V^{{\pi'}_t^k}(b_t^k) \geqslant V^{\pi_t^k}(b_t^k)$
11 **end**
12 **end**
13 **until** *no more policies in* Π_t^i *or* Π_t^j *can be removed* ;
Output: Π_t^i,Π_t^j and V^{Π_t};

Algorithm 7: Exact Point Based Dynamic Programming for Decentralized POMDPs [Szer and Charpillet, 2006].

Both PBDP and MBDP address the problem of the large (eventually infinite) number of histories (belief states) in which policies are evaluated. However, the main problem with decentralized POMDPs, compared to POMDPs, is the dimensionality of the policy space, rather then the number of histories. In fact, an important factor lying behind the poor scalability of planning algorithms for decentralized POMDPs is the computational cost of calculating the value of a policy in a given belief state. This latter calculation is repeated for

each policy and belief state, and any improvement at that level would lead to a significant improvement of the global performance of the planning algorithm. Moreover, the memory space needed to represent the value vectors at each time-step t corresponds to $|\mathcal{S}| \times |\Pi_t|$ real units, where the number of policies $|\Pi_t|$ grows double exponentially with respect to the horizon and the number of observations. Indeed, if we get $|\Pi^i_{t+1}|$ policies for agent i at step $t+1$, then $|\mathcal{A}^i||\Pi^i_t|^{|\mathcal{O}^i|}$ new policies will be created at step t by the full backup operation, and the number of joint policies will be $|\Pi_t| = |\Pi^i_t| \times |\Pi^j_t|$.

To solve this problem, we propose a new method for accelerating dynamic programming algorithms by compressing the policy belief space. Our approach is based on the following observation: given a set of policies, only a few sequences are necessary to represent all the policies. To do so, we will first show how to use Predictive State Representations for modeling multi-agent beliefs. In this model, policies are replaced by the sequences of actions and observations that have linearly independent probabilities. These independent sequences can be found quickly in polynomial time, and we can guarantee that the number of independent sequences will never exceed the number of policies.

4.3 Predictive Policy Representations (PPRs)

A policy is, in general, a function that maps each history into a distribution on actions. From this point of view, the stochastic process generating actions can be modeled as a discrete-time dynamical system, and any model used for representing discrete-time dynamical systems can be used for representing policies as well. In particular, Predictive State Representations (PSR) can be used to represent policies by switching the roles of actions and observations.

Wiewiora [2005] already showed that any function, mapping histories into a distribution on actions, can be represented by a PSR, the dimensionality of this representation was used for analyzing learning algorithms in partially observable environments [Wiewiora, 2007]. We refer to this model, where policies are given by PSRs, as Predictive Policy Representations (PPRs). Along this section, we show how to represent a policy of a given agent as a PPR, and compare this representation to other models. We denote the actions of the agent by $\mathcal{A}$, and its observations by $\mathcal{O}$. The upper script will be used to indicate the index of an action or an observation, and the lower scripts to indicate time-steps.

We redefine the notion of a test q as an ordered sequence of (*observation, action*) couples, i.e. $q = o^1a^1 \dots o^k a^k$. The prediction of a test q, is the probability that the agent will execute the sequence of actions $a_1, \dots, a_k$ if it perceives the sequence of observations $o_1, \dots, o_k$. The probability of a test q starting after a history h_t is defined by:

$$Pr(q|h_t) \stackrel{def}{=} Pr(q^a|h_t, q^o) = Pr(a_{t+1} = a^1, \dots, a_{t+k} = a^k | h_t, o_{t+1} = o^1, \dots, o_{t+k} = o^k) \quad (4.5)$$

where q^a denotes the sequence of actions in q and q^o denotes the sequence of observations.

The history h_t ends with an action and not an observation as in PSRs, because tests in PPRs start with an observation. In fact, a time-step t is the moment after executing a_t and before perceiving o_t. We also consider that all the histories start with a fictive observation o^* which has probability 1 (the default observation).

A test in PPRs can be seen as a question regarding what the agent will do when it perceives a specified sequence of observations.

PPR is different from state-based representations, where the actions are directly related to physical states (or belief states), and from Finite-State Machines [Hansen, 1998], where internal states are defined, and the actions are chosen according to these states.

We define the *policy matrix* D as an infinite dimensional matrix, where the columns correspond to all possible tests and the rows to all possible histories. An entry $D(h_t, q)$ is given by $Pr(q|h_t)$, the probability that the actions indicated in the test q will be executed by the agent, such that the current history of the system is h_t and the future observations will be the observations indicated in q. As in PSRs (Section 2.4), the tests corresponding to linearly independent rows in the policy matrix D are called core tests.

We will show that for a large class of policies, a finite set of core tests always exists. We denote the set of core tests by $Q = \{q_1, q_2, \ldots, q_k\}$. The vector of core tests probabilities, $Pr(Q|h) = [P(q_1|h), \ldots, P(q_n|h)]^T$, forms a sufficient statistic for the system after any history h, and can be used as a belief about the internal state of the agent. As for PSRs, we will focus on a special class of PPRs where the projection function, mapping beliefs into predictions, is linear. If we denote this projection function by a vector m_q, then for any test q :

$$Pr(q|h) = Pr(Q|h)^T m_q \tag{4.6}$$

After perceiving an observation o and executing an action a, the probability of a core test q is updated by using Bayes' Rule:

$$Pr(q|hoa) = \frac{Pr(oaq|h)}{Pr(oa|h)} = \frac{Pr(Q|h)^T m_{oaq}}{Pr(Q|h)^T m_{oa}} \tag{4.7}$$

A linear PPR model is then a tuple $\langle \mathcal{A}, \mathcal{O}, Q, \{m_{oa}\}, \{m_{oaq_i}\} \rangle$, where $\mathcal{A}$ and $\mathcal{O}$ are action and observation sets, Q is a finite set of core tests $\{q_1, q_2, \ldots, q_k\}$, m_{oa} are weight vectors for one-step tests, defined for observation $o \in \mathcal{O}$ and action $a \in \mathcal{A}$, and m_{oaq_i} are weight vectors for one-step extensions of core tests, defined for each observation $o \in \mathcal{O}$, action $a \in \mathcal{A}$ and core test $q_i \in Q$.

The theorem below (Theorem 1) makes a comparison between PPR and Stochastic Finite-State Machines (FSMs) [Hansen, 1998]. FSMs are a popular model used for representing infinite-horizon policies. FSMs are to POMDPs exactly what PPRs are to PSRs. Many policy representations can be seen as a special case of FSMs, such as decision trees or environment

state based policies (as in MDPs). The main result of this theorem is that PPRs offer a representation that uses at most the same number of parameters used in the equivalent FSM. However, there no general conditions under which PPRs are more compact than FSMs, though we will give an example where this is the case. The number of parameters considered here is the number of core policy tests for the PPR, and the number of states for the FSM.

Finite-State Machines are a graphical model that can represent stochastic policies without memorizing all the histories [Meuleau et al., 1999]. An FSM is defined by: a finite set of internal states (I-states) $\mathcal{G}$; a set of action-selection functions $\mu^{o,a}$, where $\mu^{o,a}(g,\theta)$ is the probability that the agent will execute action a if its I-state is g, its last observation was o, and the parameters of the policy are θ; a set of transition functions ω^o, where $\omega^o(g,g',\theta)$ is the probability that the agent will select the I-state g if the current I-state is g and the perceived observation is o.

Figure 4.2 shows a simple example of deterministic Finite-State Machines, which are a special class of stochastic machines. Every I-state is labeled by the action to be executed in that state. This machine contains four different I-states, but can be represented with only two core policy tests: $q_1 = o_1a_1$ and $q_2 = o_2a_2$. The answers to these two tests are sufficient to determine the state of the machine. For example, if we have $Pr(o_1a_1|h_t) = 1$ and $Pr(o_2a_2|h_t) = 0$, then we conclude that $Pr(o_2a_4|h_t) = 1, Pr(o_2a_4o_2a_2|h_t) = 1$, and so on (Implicitly, we are in the I-state of action a_2). In fact one can verify that:

$$\begin{aligned} Pr(o_2a_4|h_t) &= Pr(q_1|h_t)(1 - Pr(q_2|h_t)) \\ &\quad +(1 - Pr(q_1|h_t))(1 - Pr(q_2|h_t)) \\ &= 1 - Pr(q_2|h_t) \end{aligned}$$

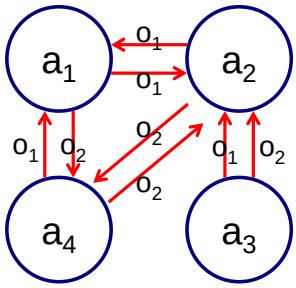


Figure 4.2: A deterministic FSM that can be represented with only two tests.

Similar nonlinear rules can be derived for every sequence.

Theorem 1. *Every stochastic Finite-State Machine can be transformed to an equivalent linear PPR with a number of core tests no more than the number of the internal states of the Machine.*

Proof. This proof borrows an idea introduced in [Singh et al., 2004]. Let D be the policy matrix corresponding to a given FSM. Since $D(h_t,q) = Pr(q|h_t) = \sum_{g\in\mathcal{G}} Pr(g|h_t)Pr(q|g)$, then we can decompose the matrix D into $D = FU$, where F is a $\infty \times |\mathcal{G}|$ matrix, defined by $F(h_t,g) = Pr(g|h_t)$, and U is a $|\mathcal{G}| \times \infty$ matrix, defined by $U(g,q) = Pr(q|g)$. F and U can be constructed by using the transition and action-selection functions of the machine. Since $rank(F) \leqslant |\mathcal{G}|$ and $rank(U) \leqslant |\mathcal{G}|$ (the rank of a matrix is upper bounded by both the number of its rows and the number of its columns), then $rank(D) = rank(FU) \leqslant |\mathcal{G}|$. The number of core tests needed in the linear PPR model is at most equal to $|\mathcal{G}|$, since the core tests correspond to linearly independent vectors in the matrix D. □

In the remaining of this chapter, we will see how one can use PPRs to represent policies in decentralized POMDPs. However, adapting PPRs to dynamic programming algorithms requires a complicated construction of the value vectors. For this reason, we will first consider only a simplified version of linear PPRs, where each test, denoted by $ao\pi$, is a pair of action a and observation o followed by a full decision tree policy π, instead of one sequence of actions and observations. We also treat all the tests as core tests, consequently, every test $ao\pi$ is explicitly represented, and we completely ignore the linear dependencies between tests. In fact, we found from our experiments that this simple semi-sequential representation of policies is sufficient for accelerating planning algorithms, even without performing a further dimensionally reduction.

4.4 Point Based Dynamic Programming with PPRs

PPRs can be used with PBDP as well as with MBDP (see Section 4.2.3) since the main difference between these two algorithms is in the number of policies considered and not the method used to represent these policies.

4.4.1 Reduced multiagent belief states

We propose to use Predictive Policy Representations in order to reduce the dimensionality of belief points in the PBDP algorithm, presented in Section 4.2.3. To do so, we need to redefine the belief state and the value vectors with PPRs. A belief state $\tilde{b}_t(s, ao\pi)$ for agent i is the probability that the system is in state s, and agent j will execute the action a, and if the observation of j will be o, then the next policy of j will be π (π is a decision tree).

$$\tilde{b}_t(s, ao\pi) = Pr(s_t = s|h_t)Pr(a_t = a, \pi_{t+1} = \pi|h_t, o_t = a) \tag{4.8}$$

The only difference between this definition and the usual definition of multi-agent belief states is that each tree π is factored at the first level and separated into several tests $ao\pi'$, $a = A(\pi)$ is the first action (root) of the policy π, $\pi' = P(\pi, o)$ is the subtree of π under action a and observation o. So, for each policy π for agent j, there are $|\mathcal{O}^j||\Pi^j_{t+1}|$ corresponding tests. Generally, every test appears in several policies, and the number of tests is significantly smaller than the number of policies. This is particularly true right after the exhaustive backup, and before pruning the dominated policies, since $|\Pi^j_t| = |\mathcal{A}^j||\Pi^j_{t+1}|^{|\mathcal{O}^j|}$ new policies are generated, while we need only $|\mathcal{A}^j||\mathcal{O}^j||\Pi^j_{t+1}|$ tests to represent all these policies. Consequently, the size of a belief state defined on tests can be exponentially smaller than the size of a belief state defined on policies.

A belief on states and policies can always be written as a function of a belief on states and tests:

$$
\begin{aligned}
b_t(s,\pi) &\stackrel{def}{=} Pr(s,\pi|h_t) \\
&= Pr(s|h_t)Pr(\pi|h_t) \text{ (the two probabilities are independent given the history } h_t) \\
&= Pr(s|h_t)Pr(A(\pi)|h_t)\prod_{o\in\mathcal{O}^j} Pr(P(\pi,o)|h_t,A(\pi),o) \\
&\quad (\ A(\pi) \text{ is the first action of } \pi \text{ and } P(\pi,o) \text{ is the remaining of } \pi \text{ after } A(\pi) \text{ and } o) \\
&= Pr(s|h_t)Pr(A(\pi)|h_t)\prod_{o\in\mathcal{O}^j}\frac{Pr(A(\pi)oP(\pi,o)|h_t)}{Pr(A(\pi)|h_t)} \text{ (Bayes' Rule)} \\
&= Pr(s|h_t)\frac{\prod_{o\in\mathcal{O}^j} Pr(A(\pi)oP(\pi,o)|h_t)}{[\sum_{\pi'\in\Pi^j_{t+1}} Pr(A(\pi)o^0\pi'|h_t)]^{|\mathcal{O}^j|-1}} \\
&= \underbrace{\left[\sum_{a\in\mathcal{A}^j}\sum_{\pi'\in\Pi^j_{t+1}}\tilde{b}_t(s,ao^0\pi')\right]}_{Pr(s|h_t)}\underbrace{\left[\frac{\prod_{o\in\mathcal{O}^j}\sum_{s'\in\mathcal{S}}\tilde{b}_t(s',A(\pi)oP(\pi,o))}{[\sum_{s'\in\mathcal{S}}\sum_{\pi'\in\Pi^j_{t+1}}\tilde{b}_t(s',A(\pi)o^0\pi')]^{|\mathcal{O}^j|-1}}\right]}_{Pr(\pi|h_t)} \\
&\quad \text{(From Equation 4.8)}
\end{aligned}
$$

where o^0 is any observation in the set $\mathcal{O}^j$, since the actions are selected independently of the upcoming observations. In fact, one can easily verify that:

$$
\begin{aligned}
\forall a\in\mathcal{A}^j,\forall o,o'\in\mathcal{O}^j:\sum_{\pi\in\Pi^j_{t+1}} Pr(a_t=a,\pi_{t+1}=\pi|o_t=o) &= \sum_{\pi\in\Pi^j_{t+1}} Pr(a_t=a,\pi_{t+1}=\pi|o_t=o') \\
&= Pr(a_t=a)
\end{aligned}
$$

In order that a randomly generated belief on $\mathcal{S}$ and Π^j_t will be an accurate belief, we only need to guarantee that:

$$
\begin{cases}
\forall s\in\mathcal{S},\forall\pi\in\Pi^j_t : b_t(s,\pi)\geqslant 0 \\
\sum_{s\in\mathcal{S}}\sum_{\pi\in\Pi^j_t} b_t(s,\pi)=1
\end{cases}
$$

However, to guarantee that a randomly generated belief on $\mathcal{S}$ and $\mathcal{A}^j\times\mathcal{O}^j\times\Pi^j_{t+1}$ (the tests used to represent the policies of the set Π^j_t) will be an accurate belief (i.e. defines a distribution on states and policies), one should verify:

$$
\begin{cases}
\forall s\in\mathcal{S},a\in\mathcal{A}^j,o\in\mathcal{O}^j,\forall\pi\in\Pi^j_{t+1} : \tilde{b}_t(s,ao\pi)\geqslant 0 \\
\forall o\in\mathcal{O}^j : \sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{A}^j}\sum_{\pi\in\Pi^j_{t+1}}\tilde{b}_t(s,ao\pi)=1 \\
\forall a\in\mathcal{A}^j,\forall o,o'\in\mathcal{O}^j : \sum_{s\in\mathcal{S}}\sum_{\pi\in\Pi^j_{t+1}}\tilde{b}_t(s,ao\pi)=\sum_{s\in\mathcal{S}}\sum_{\pi\in\Pi^j_{t+1}}\tilde{b}_t(s,ao'\pi)
\end{cases}
\tag{4.9}
$$

In fact, the sum $\sum_{s\in\mathcal{S}}\sum_{\pi\in\Pi^j_{t+1}}\tilde{b}_t(s,ao\pi)$ is the probability that agent j will execute action a at time-step t, and it must be the same for any next observation o. Contrary to decision trees, which are always mutually exclusive, the tests starting with the same action and followed by different observations are not mutually exclusive.

4.4.2 Reduced value vectors

In order to use the new policy representation in a dynamic programming algorithm, we need to define the value of any policy as a function of the reduced belief states. To do so, we first define the value of a joint test $\langle a^i o^i \pi^i_{t+1}, a^j o^j \pi^j_{t+1} \rangle$ starting from state s as:

$$\begin{aligned} V^{\langle a^i o^i \pi^i, a^j o^j \pi^j \rangle}(s) &= \sum_{s' \in \mathcal{S}} Pr(s'|s, \langle a^i, a^j \rangle) Pr(\langle o^i, o^j \rangle | s', \langle a^i, a^j \rangle) R(s, \langle a^i, a^j \rangle) \\ &\quad + \gamma \sum_{s' \in \mathcal{S}} Pr(s'|s, \langle a^i, a^j \rangle) Pr(\langle o^i, o^j \rangle | s', \langle a^i, a^j \rangle) V^{\langle \pi^i, \pi^j \rangle}(s') \end{aligned}$$

The value of an individual policy π^i, according to a belief state $\tilde{b}^i_t$, is given by:

$$V^{\pi^i}(\tilde{b}^i_t) = \sum_{s \in \mathcal{S}} \sum_{a^j \in \mathcal{A}^j} \sum_{\substack{o^i \in \mathcal{O}^i \\ o^j \in \mathcal{O}^j}} \sum_{\pi'^j \in \Pi^j_{t+1}} \tilde{b}^i_t(s, a^j o^j \pi'^i) V^{\langle A(\pi^i) o^i P(\pi^i, o^i), a^j o^j \pi'^j \rangle}(s)$$

We can now verify that the value of any policy according to a belief on states and policies is equal to its value according to the corresponding belief on states and tests. Indeed:

$$\begin{aligned} V^{\pi^i}(\tilde{b}^i_t) &= \sum_{s \in \mathcal{S}} \sum_{a^j \in \mathcal{A}^j} \sum_{\substack{o^i \in \mathcal{O}^i \\ o^j \in \mathcal{O}^j}} \sum_{\pi'^j \in \Pi^j_{t+1}} \tilde{b}^i_t(s, a^j o^j \pi'^j) V^{\langle A(\pi^i) o^i P(\pi^i, o^i), a^j o^j \pi'^j \rangle}(s) \\ &= \sum_{s \in \mathcal{S}} Pr(s) \sum_{a^j \in \mathcal{A}^j} \sum_{\substack{o^i \in \mathcal{O}^i \\ o^j \in \mathcal{O}^j}} \sum_{\pi'^j \in \Pi^j_{t+1}} Pr(a^j o^j \pi'^j) V^{\langle A(\pi^i) o^i P(\pi^i, o^i), a^j o^j \pi'^j \rangle}(s) \\ &= \sum_{s \in \mathcal{S}} Pr(s) \sum_{a^j \in \mathcal{A}^j} \sum_{\substack{o^i \in \mathcal{O}^i \\ o^j \in \mathcal{O}^j}} \sum_{\pi'^j \in \Pi^j_{t+1}} Pr(a^j o^j \pi'^j) \Big[\sum_{s' \in \mathcal{S}} \big[Pr(s'|s, \langle a^i, a^j \rangle) Pr(\langle o^i, o^j \rangle | s', \langle a^i, a^j \rangle) \\ &\quad R(s, \langle a^i, a^j \rangle) \big] + \gamma \sum_{s' \in \mathcal{S}} Pr(s'|s, \langle a^i, a^j \rangle) Pr(\langle o^i, o^j \rangle | s', \langle a^i, a^j \rangle) V^{\langle \pi^i, \pi^j \rangle}(s') \Big] \\ &= \sum_{s \in \mathcal{S}} Pr(s) \sum_{a^j \in \mathcal{A}^j} Pr(a^j) R(s, \langle a^i, a^j \rangle) \\ &\quad + \gamma \sum_{s \in \mathcal{S}} Pr(s) \sum_{a^j \in \mathcal{A}^j} \sum_{\substack{o^i \in \mathcal{O}^i \\ o^j \in \mathcal{O}^j}} \sum_{\pi'^j \in \Pi^j_{t+1}} Pr(a^j o^j \pi'^j) \sum_{s' \in \mathcal{S}} [Pr(s'|s, \langle a^i, a^j \rangle) Pr(\langle o^i, o^j \rangle | s', \langle a^i, a^j \rangle) \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad V^{\langle \pi^i, \pi^j \rangle}(s')] \\ &= \sum_{s \in \mathcal{S}} \sum_{\pi^j \in \Pi^j_t} b^i_t(s, \pi^j) V^{\langle \pi^i, \pi^j \rangle}(s) = V^{\pi^i}(b^i_t) \end{aligned}$$

Therefore, instead of using the high dimensional beliefs b^i_t for evaluating a policy π, one can use the equivalent reduced belief $\tilde{b}^i_t$ and obtain exactly the same result. Note that the cost of calculating $V^{\pi^i}(b^i_t)$ is $O(|\mathcal{S}||\Pi^j_t|)$ while the cost of calculating $V^{\pi^i}(\tilde{b}^i_t)$ is $O(|\mathcal{S}||\mathcal{A}^j||O^i||O^j||\Pi^j_{t+1}|)$, which is significantly smaller given that $|\Pi^j_t| = |\mathcal{A}^j||O^j|^{|\Pi^j_{t+1}|}$.

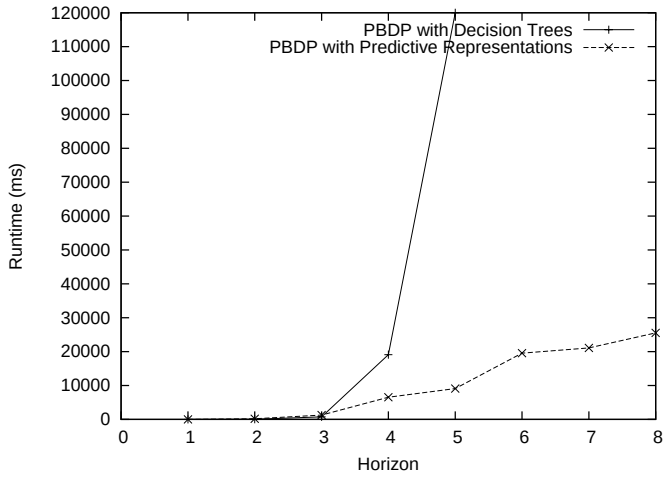
4.4.3 Empirical Results

For our first experiments, we implemented the PBDP algorithm using a randomly generated set of accurate belief points for the dominance test (Inequality 4.3), without considering if these points are reachable or not. In fact, for the small problems used in the decentralized POMDPs literature, we found that there is no significant improvement when we consider only the reachable belief points. Instead of spending a large proportion of the runtime looking for reachable belief states, we consider a set of random belief states, generated at the beginning of the algorithm, and every optimal policy will be likely dominant in at least one of these points. However, this heuristic does not guarantee that all the optimal policies will be found.

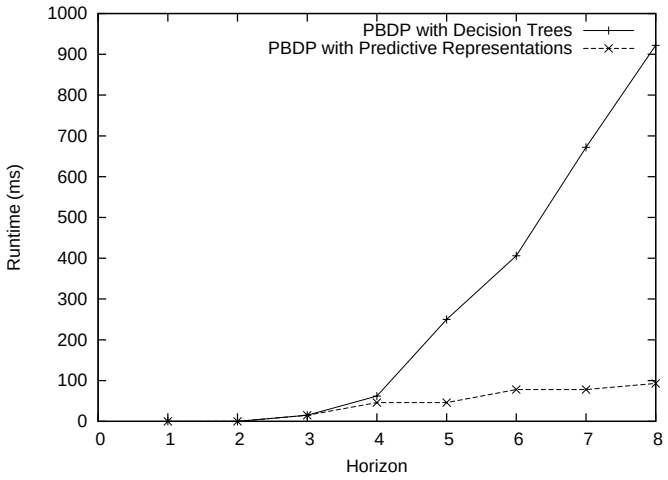
We implemented the PBDP algorithm with both full decision trees and our modified version of Predictive Policy Representations, and we compared the performance of theses two approaches on three standard problems taken from DEC-POMDPs literature: Multiagent Tiger (MA-Tiger), Multiagent Broadcasting Channel (MABC), and The Meeting problem (with a 2×2 grid) [Seuken and Zilberstein, 2007a]. The code of the implementation is written in C++, and the experiments were performed on a 1.73 GHz Pentium M processor, with a RAM of 512 Mo. In our implementation, we relaxed the constraints 4.9 on the belief states in PPRs except for the constraint that all the probabilities are between 0 and 1, so some belief states may be inaccurate (useless), but they can be quickly generated. We used 100 random belief points for MA-Tiger, 25 random belief points for MABC, and 30 random belief points for The Meeting problem. The same number of belief points was used for both PBDP with decision trees and PBDP with PPRs.

Figures 4.3 show the runtime of the PBDP for different horizons. As expected, the runtime of PBDP is significantly reduced when we use a predictive representation of the policies. In the multiagent tiger (Figure 4.3 (a)) and the meeting (Figure 4.3 (c)) problems, PBDP runs out of time and was stopped at horizons 5 and 4 respectively. This is explained by the fact that the belief points defined on tests have a smaller dimensionality than the belief points defined on decision trees. Consequently, the value of a given policy can be calculated with fewer operations, and the dominance verification (Inequality 4.3) is performed in a shorter time. We also noticed that most of this computational gain is made right after the full backup, where the belief points in the decision trees approach contain a number of policies that is exponential with respect to the number of observations.

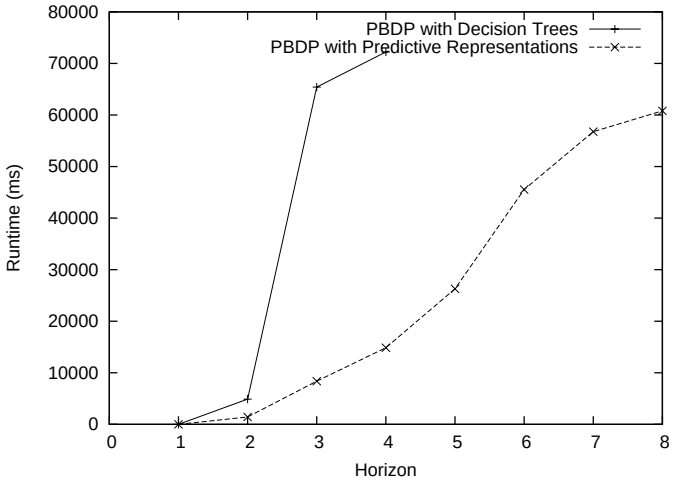
Table 4.1 shows that the values of the policies returned by PBDP are almost the same for the two types of representations. These values were estimated using Bellman Equation 4.1. The values with the PPR approach are slightly suboptimal because the belief points were underconstrained, and since we used a limited set of belief points, some optimal policies were weakly dominated in all these points, and therefore wrongfully eliminated.



(a) Multiagent Tiger problem



(b) Multiagent Broadcasting Channel problem



(c) The meeting problem

Figure 4.3: The effective runtime of the PBDP algorithm as a function of the horizon, with different problems.

Meeting problem	H=2	H=3	H=4	H=5	H=6	H=7	H=8
PBDP with Decision Trees	0	0.81	1.90	n.a.	n.a.	n.a.	n.a.
PPRs	0.81	1.79	2.78	3.78	4.78	5.78	6.78
MA-Tiger problem	H=2	H=3	H=4	H=5	H=6	H=7	H=8
PBDP with Decision Trees	-4	5.19	4.80	n.a.	n.a.	n.a.	n.a.
PPRs	-4	5.19	4.39	4.21	2.27	0.41	-1.5
MABC problem	H=2	H=3	H=4	H=5	H=6	H=7	H=8
PBDP with Decision Trees	2	2.90	3.89	4.79	5.69	6.59	7.49
PBDP with PPRs	2	2.99	3.80	4.79	5.60	6.50	7.49

Table 4.1: The values of the optimal policies returned by PBDP using decision trees and PPRs to represent policies, as function of the planning horizon H.

4.5 Exact Dynamic Programming with PPRs

4.5.1 Motivation

The exact dynamic programming operator (Algorithm 6) spends most of its runtime searching for weakly dominated policies by checking Inequality 4.3 for every policy π^i. The usual approach for performing this test is to use the linear program of Table 4.4. The computational complexity of solving this linear program is, on average, polynomial on the size of the multiagent belief states, which is $|\mathcal{S}| \times |\Pi_t^i|$. The number of policies $|\Pi_t^i|$ is exponential with respect to the number of observations and double exponential with respect to the planning horizon. However, the main problem of Dynamic Programming is the size of the memory space required to represent the value vectors for the joint policy. The memory space required to represent these vectors at time-step t is $|\Pi_t^i||\Pi_t^j||\mathcal{S}|$ floats. Indeed, this algorithm runs out of memory several iterations before running out of time.

To alleviate these problems, we need to use a more compact technique for representing the belief of agent i on the policies of agent j, instead of the naive probability distribution on all the policies Π_t^j. To do so, we can exploit the structure of the the set Π_t^j, and find a set of features $Q = \{q_1, q_2, \ldots, q_k\}$ that constitute a sufficient information, so that each point in the space of beliefs on policies will correspond to a point in the space of beliefs on these features, and vice versa. We should also guarantee that $|Q|$ is significantly smaller that $|\Pi_t^j|$ for most of the time.

A policy, specified by a decision tree, is a finite collection of tests. A test q^i for agent i at time-step t is an ordered list of $H-t$ individual actions and $H-t-1$ individual observations: $q^i = a_t^i, o_t^i, \ldots, o_{H-1}^i, a_H^i$. A joint test $\vec{q}$ is a couple $\langle q^i, q^j \rangle$ where q^i is an individual test for agent i and q^j is an individual test for agent j, a joint test can also be seen as one ordered list of joint actions and observations.

The set of policies Π_t^i can be completely replaced, without any loss of information, by an outcome matrix U_t^i, where each row corresponds to a policy $\pi^i \in \Pi_t^i$ and each column corresponds to a test q^i. An entry $U_t^i(\pi^i, q^i)$ is defined as the probability that agent i will execute the actions of the test q^i if the observations of q^i occur, when the current policy of agent i is π^i. Since the policies are deterministic, we have $U_t^i(\pi^i, q^i) = 1$ if the test q^i appears in the policy π^i, otherwise $U_t^i(\pi^i, q^i) = 0$. Figure 4.4 shows a set Π_t^i containing 4 individual policies for agent i: q_a, q_b, q_c and q_d. There are 8 different tests in these policies, the matrix U^i has then 4 rows and 8 columns.

If $b^j(s, .)$ is a multiagent belief state, i.e. a probability distribution on the policies of Π_t^i for some state $s \in \mathcal{S}$, then the product $b^j(s, .)U_t^i$ is a vector containing the probability of every test of Π_t^i. To reduce the dimensionality of $b^j(s, .)$ from $|\Pi_t^i|$ to N, we should find a *compression function* f defined by:

$$\begin{aligned} f : \Delta\Pi_t^i &\rightarrow [0,1]^N \\ b^j(s, .) &\mapsto \tilde{b}^j(s, .) \end{aligned}$$

We refer to $\tilde{b}^j(s, .)$ as *the reduced multiagent belief state*, it corresponds to the belief about the tests, whereas $b^j(s, .)$ is the belief about the policies. In order that f be an accurate compression function, we should be able to make predictions about any test q^i by using only the belief $\tilde{b}^j(s, .)$.

In the next section, we show how to find the compression function f in the case of linear predictive representations.

4.5.2 Linear reduction of policy space dimensionality

The outcome matrix U_t^i, representing the policies of agent i at time-step t, can be factorized as follows:

$$U_t^i = F_t^i \tilde{U}_t^i$$

In this case, the compression function is simply the matrix F_t^i. The reduced belief state $\tilde{b}_t^j$ can be generated from b_t^j by:

$$\tilde{b}_t^j(s, .) = [b_t^j(s, .)]^T F_t^j \tag{4.10}$$

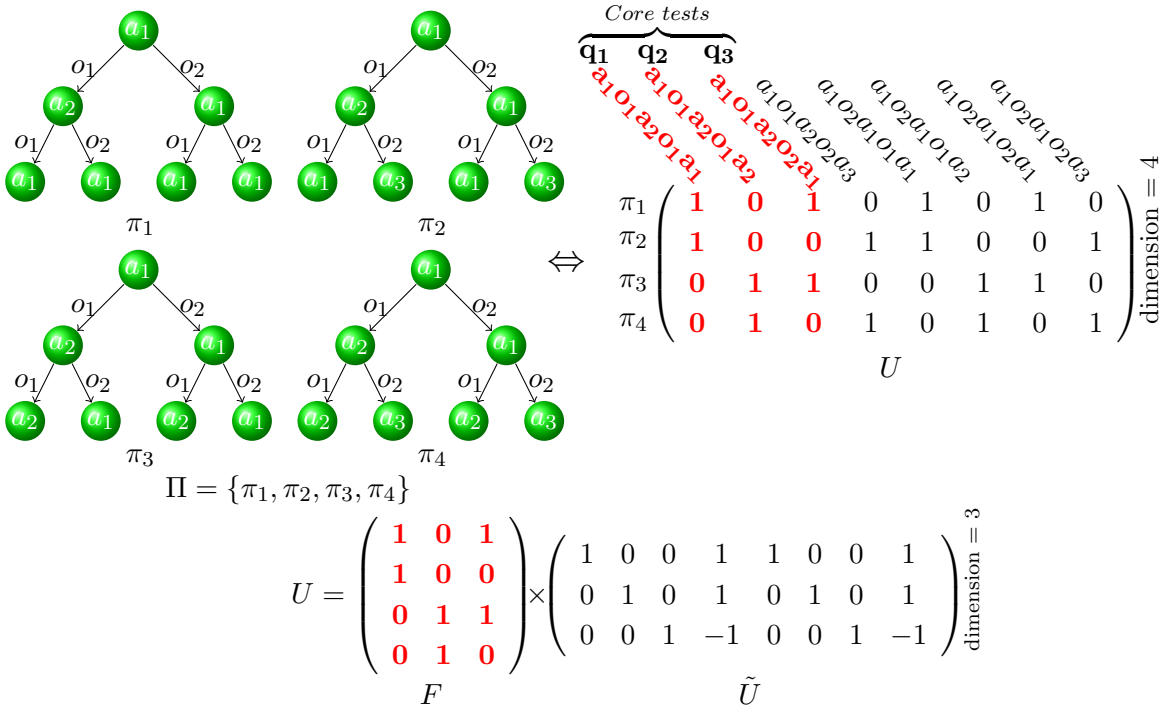


Figure 4.4: Linear reduction of the policy space dimension.

The probability of a test q^i is given by:

$$\begin{aligned} Pr(q^i|h_t) &= \sum_{s\in\mathcal{S}} [b_t^i(s,.)]^T U_t^i(.,q^i) \\ &= \sum_{s\in\mathcal{S}} [b_t^i(s,.)]^T F_t^i \tilde{U}_t^i(.,q^i) \\ &= \sum_{s\in\mathcal{S}} [\tilde{b}_t^j(s,.)]^T \tilde{U}_t^i(.,q^i) \\ &= \sum_{s\in\mathcal{S}} [\tilde{b}_t^j(s,.)]^T m_{q^i} \end{aligned} \tag{4.11}$$

where $m_{q^i} \stackrel{def}{=} \tilde{U}_t^i(.,q^i)$ is a weight vector associated to the test q^i (we have one weight vector per test). This means that given a reduced belief state, the probability of any test is a linear combination of the probabilities contained in this reduced belief state. The matrix F_t^i is a basis for the matrix U_t^i, it corresponds to a set of linearly independent columns of U_t^i. The set Q_t^i of tests corresponding to the columns of F_t^i are the core tests of the equivalent predictive representation. In Figure 4.4, the set Q_t^i contains the three core tests q_1, q_2 and q_3. Note that $|Q_t^i| \leqslant |\Pi_t^i|$, because the linear rank of a matrix cannot be higher than the number of rows in this matrix, and F_t^i has exactly $|\Pi_t^i|$ rows (the same argument used in the proof of Theorem 1).

4.5.3 Finding the core tests

The main problem now is how to find the set Q_t^i of core tests and their corresponding matrix F_t^i without having to factorize the entire matrix U_t^i at each step of the dynamic programming algorithm. The theorem below states that if Q_t^i is the set of core tests for the policies at time-step t, then the core tests of Q_{t-1}^i for the time-step $t-1$ will be among the *one step* extensions of the core tests Q_t^i (Remember that in dynamic programming, the policies that will be executed at time-step $t-1$ are constructed from the policies that will be executed at time-step t). This means that for a given time-step t, only the core tests need to be extended in order to find the core tests for the previous time-step $t-1$.

Theorem 2. *Let Π_t^i be a set of policies for time-step t, Q_t^i the set of core tests corresponding to Π_t^i, and Π_{t-1}^i the set of policies for time-step $t-1$ created from Π_t^i by an exhaustive backup, then Π_{t-1}^i can be represented by a set of core tests Q_{t-1}^i satisfying $Q_{t-1}^i \subseteq \mathcal{A}^i \times \mathcal{O}^i \times Q_t^i$.*

Proof. This theorem claims that one can always find a set of core tests for the policies Π_{t-1}^i by simply extending a set of core tests for the policies Π_t^i. In other words, $\forall q^i \in Q_t^i, \forall a^i \in \mathcal{A}^i, \forall o^i \in \mathcal{O}^i$, the column $U_{t-1}^i(., a^i o^i \tilde{q}^i)$ is a linear combination of the columns of F_{t-1}^i, which is the sub-matrix of U_{t-1}^i corresponding to the tests $a^i o^i q^i : a^i \in \mathcal{A}^i, o^i \in \mathcal{O}^i, q^i \in Q_{t-1}^i$. Let π^i be a policy from the set Π_{t-1}^i, q^i a test for time-step t, a^i an action from $\mathcal{A}^i$ and o an observations from $\mathcal{O}^i$, then we can be in one of the following two situations:

Case 1: The policy tree π^i starts with an action different from a^i, i.e. $A(\pi^i) \neq a^i$, then the test $a^i o^i q^i$ does not appear in the policy π^i, so: $U_t^i(\pi^i, a^i o^i q^i) = 0$.

Case 2: The policy tree π^i starts with the action a^i, then the test $a^i o^i q^i$ appears in the policy π^i if and only if q^i appears in $P(\pi^i, o^i)$, the sub-policy (tree) of π^i under the first action a^i and the observation o^i. We have then $U_{t-1}^i(\pi^i, a^i o^i q^i) = U_t^i(P(\pi^i, o^i), q^i)$, and in particular:

$$\forall q'^t \in Q_t^i : F_{t-1}^i(\pi^i, a^i o^i q'^i) = F_t^i(P(\pi^i, o^i), q'^i) \tag{4.12}$$

Therefore:

$$\begin{aligned} U_{t-1}^i(\pi^i, a^i o^i q^i) &= U_t^i(P(\pi^i, o^i), q^i) \\ &= F_t^i(P(\pi^i, o^i), .) m_{q^i} \ (\textit{since } F_t^i \textit{ is a basis of } U_t^i \ ) \\ &= F_{t-1}^i(\pi^i, .) m_{a^i o^i q^i} \ (\textit{from Equation 4.12}) \end{aligned}$$

where the weight vector $m_{a^i o^i q^i}$ is defined as follows:

$$\begin{cases} m_{a^i o^i q^i}(a^i o^i q'^i) = m_{q^i}(q'^i), \forall q'^i \in Q_{t-1}^i \\ m_{a^i o^i q^i}(a'^i o'^i q'^i) = 0, \ \ \forall q'^i \in Q_{t-1}^i, \forall a'^i \in \mathcal{A}^i - \{a^i\}, \forall o'^i \in \mathcal{O}^i - \{o^i\}. \end{cases} \tag{4.13}$$

Cases 1 and 2 can be regrouped together, we have then:

$$U^i_{t-1}(\pi^i, a^i o^i q^i) = F^i_{t-1}(\pi^i, .) m_{a^i o^i q^i}$$

Notice that the vector $m_{a^i o^i q^i}$ does not depend on the policy $\pi^i \in \Pi^i_{t-1}$. Then, the policies Π^i_{t-1} can be represented by the tests $\{a^i o^i q^i : a^i \in \mathcal{A}^i, o^i \in \mathcal{O}^i, q^i \in Q^i_{t-1}\} = \mathcal{A}^i \times \mathcal{O}^i \times Q^i_t$. □

The corresponding matrix F^i_{t-1} may contain some linearly dependent columns, along with the basis columns. Thus, we perform a Gauss-Jordan elimination on F^i_{t-1} (or any other decomposition technique) to extract a minimal set of core tests in polynomial time.

4.5.4 Reduced value vectors

In order to use the predictive representations for planning, we need to redefine the value function (Equation 4.2), given that the belief state is now on tests instead of policies. The expected value of a joint test $\vec{q} = \vec{a}_0\vec{o}_0\vec{a}_1\vec{o}_1 \dots \vec{a}_H$ in a state s is given by:

$$V^{\vec{q}}(s) \quad = \quad Pr(\vec{q}^o|s, \vec{q}^a) R^{\vec{q}}(s) \tag{4.14}$$

where $Pr(\vec{q}^o|s, \vec{q}^a) \stackrel{def}{=} Pr(\vec{o}_1\vec{o}_2 \dots \vec{o}_{H-1}|s_0 = s, \vec{a}_0\vec{a}_1 \dots \vec{a}_{H-1})$ is the probability that the observations of test $\vec{q}$ will occur if we start executing the actions of test $\vec{q}$ at state s,
and $R^{\vec{q}}(s) \stackrel{def}{=} \mathbb{E}_{s_0,\dots,s_H}[\sum_{t=0}^H \gamma^t R(s_t, a_t)|s_0 = s, \vec{q}]$ is the reward expected from the actions of $\vec{q}$ such that the observations of $\vec{q}$ will occur. This latter term is given by:

$$\begin{aligned}
R^{\vec{q}}(s) &= \mathbb{E}_{s_0,\dots,s_H}[\sum_{t=0}^{H} \gamma^t R(s_t, a_t)|s_0 = s, \vec{q}] \\
&= \sum_{t=0}^{H} \gamma^t \sum_{s' \in \mathcal{S}} Pr(s_t = s'|s_0 = s, \vec{a}_0\vec{o}_0\vec{a}_1\vec{o}_1 \dots \vec{a}_H) R(s', \vec{a}_t) \\
&= \frac{\sum_{t=0}^H \gamma^t \sum_{s' \in \mathcal{S}} Pr(s_t = s', \vec{o}_0 \dots \vec{o}_{H-1}|s_0 = s, \vec{a}_0 \dots \vec{a}_{H-1}) R(s', \vec{a}_t)}{Pr(\vec{q}^o|s, \vec{q}^a)} \ (\textit{Bayes' rule}) \\
&= \frac{\sum_{t=0}^H \gamma^t \sum_{s' \in \mathcal{S}} \alpha_t(s, s', \vec{q}) \beta_t(s', \vec{q}) R(s', \vec{a}_t)}{Pr(\vec{q}^o|s, \vec{q}^a)} \ (\textit{Markov property})
\end{aligned} \tag{4.15}$$

where:

$$\begin{cases}
\alpha_t(s, s', \vec{q}) \stackrel{def}{=} Pr(\vec{o}_0\vec{o}_1 \dots \vec{o}_{t-1}, s_t = s'|s_0 = s, \vec{a}_0\vec{a}_1 \dots \vec{a}_{H-1}) \\
\alpha_0(s, s, \vec{q}) \stackrel{def}{=} 1 \\
\alpha_0(s, s', \vec{q}) \stackrel{def}{=} 0, \forall s' \in \mathcal{S} - \{s\} \\
\beta_t(s', \vec{q}) \stackrel{def}{=} Pr(\vec{o}_t \dots \vec{o}_{H-1}|s_t = s', \vec{a}_t\vec{a}_{t+1} \dots \vec{a}_{H-1}) \\
\beta_H(s', \vec{q}) \stackrel{def}{=} 1
\end{cases} \tag{4.16}$$

We applied Bayes' rule, and then decomposed $Pr(s_t = s', \vec{o}_0 \dots \vec{o}_{H-1}|s_0 = s, \vec{a}_0 \dots \vec{a}_H)$ into $\alpha_t(s, s', \vec{q})$ and $\beta_t(s', \vec{q})$, taking advantage of the Markov property.

From Equations 4.14 and 4.15, the expected value of a joint test $\vec{q}$ is given by:

$$\begin{aligned} V^{\vec{q}}(s) &= Pr(\vec{q}^o|s,\vec{q}^a)\frac{\sum_{t=0}^{H}\gamma^t\sum_{s'\in\mathcal{S}}\alpha_t(s,s',\vec{q})\beta_t(s',\vec{q})R(s',\vec{a}_t)}{Pr(\vec{q}^o|s,\vec{q}^a)} \\ &= \sum_{t=0}^{H}\gamma^t\sum_{s'\in\mathcal{S}}\alpha_t(s,s',\vec{q})\beta_t(s',\vec{q})R(s',\vec{a}_t) \end{aligned} \tag{4.17}$$

Let $\vec{q}'$ be the remaining part of $\vec{q}$ after the first action and observation, i.e. $\vec{q}=\vec{a}_0\vec{o}_0\vec{q}'$. Notice that:

$$\begin{aligned} \alpha_t(s,s',\vec{q}) &\stackrel{def}{=} Pr(\vec{o}_0\vec{o}_1\ldots\vec{o}_{t-1}, s_t=s'|s_0=s,\vec{a}_0\vec{a}_1\ldots\vec{a}_{H-1}) \\ &= \sum_{s''\in\mathcal{S}} Pr(s''|s_0=s,\vec{a}_0)Pr(\vec{o}_0|s'',\vec{a}_0)\alpha_{t-1}(s'',s',\vec{q}') \end{aligned} \tag{4.18}$$

and:

$$\begin{aligned} \beta_t(s',\vec{q}) &\stackrel{def}{=} Pr(\vec{o}_t\ldots\vec{o}_{H-1}|s_t=s',\vec{a}_t\vec{a}_{t+1}\ldots\vec{a}_{H-1}) \\ &= \beta_{t-1}(s',\vec{q}') \text{ for } t>0 \end{aligned} \tag{4.19}$$

The value $V^{\vec{q}}(s)$ is recursively calculated as follows:

$$V^{\vec{q}}(s)=\sum_{t=0}^{H}\gamma^t\sum_{s'\in\mathcal{S}}\alpha_t(s,s',\vec{q})\beta_t(s',\vec{q})R(s',\vec{a}_t)$$

$$=\sum_{s'\in\mathcal{S}}\alpha_0(s,s',\vec{q})\beta_0(s',\vec{q})R(s',\vec{a}_t)+\sum_{t=1}^{H}\gamma^t\sum_{s'\in\mathcal{S}}\alpha_t(s,s',\vec{q})\beta_t(s',\vec{q})R(s',\vec{a}_t)$$

$$=\beta_0(s,\vec{q})R(s,\vec{a}_t)+\sum_{t=1}^{H}\gamma^t\sum_{s'\in\mathcal{S}}\alpha_t(s,s',\vec{q})\beta_t(s',\vec{q})R(s',\vec{a}_t) \text{ (From Equations 4.16)}$$

$$=\beta_0(s,\vec{q})R(s,\vec{a}_t)+\sum_{t=1}^{H}\gamma^t\sum_{s'\in\mathcal{S}}\sum_{s''\in\mathcal{S}}Pr(s''|s_0=s,\vec{a}_0)Pr(\vec{o}_0|s'',\vec{a}_0)\alpha_{t-1}(s'',s',\vec{q}')\beta_t(s',\vec{q})R(s',\vec{a}_t)$$

(From Equation 4.18)

$$=\beta_0(s,\vec{q})R(s,\vec{a}_t)+\sum_{s''\in\mathcal{S}}Pr(s''|s_0=s,\vec{a}_0)Pr(\vec{o}_0|s'',\vec{a}_0)\sum_{t=1}^{H}\gamma^t\sum_{s'\in\mathcal{S}}\alpha_{t-1}(s'',s',\vec{q}')\beta_t(s',\vec{q})R(s',\vec{a}_t)$$

$$=\beta_0(s,\vec{q})R(s,\vec{a}_t)+\sum_{s''\in\mathcal{S}}Pr(s''|s_0=s,\vec{a}_0)Pr(\vec{o}_0|s'',\vec{a}_0)\sum_{t=1}^{H}\gamma^t\sum_{s'\in\mathcal{S}}\alpha_{t-1}(s'',s',\vec{q}')\beta_{t-1}(s',\vec{q}')R(s',\vec{a}_t)$$

(From Equation 4.19)

$$=\beta_0(s,\vec{q})R(s,\vec{a}_t)+\gamma\sum_{s''\in\mathcal{S}}Pr(s''|s_0=s,\vec{a}_0)Pr(\vec{o}_0|s'',\vec{a}_0)\sum_{t=0}^{H-1}\gamma^t\sum_{s'\in\mathcal{S}}\alpha_t(s'',s',\vec{q}')\beta_t(s',\vec{q}')R(s',\vec{a}_t)$$

$$=\beta_0(s,\vec{q})R(s,\vec{a}_t)+\gamma\sum_{s''\in\mathcal{S}}Pr(s''|s_0=s,\vec{a}_0)Pr(\vec{o}_0|s'',\vec{a}_0)V^{\vec{q}'}(s'') \text{ (From Equation 4.17)}$$

Therefore:

$$V^{\vec{q}}(s)=\beta_0(s,\vec{q})R(s,\vec{a}_t)+\gamma\sum_{s''\in\mathcal{S}}Pr(s''|s_0=s,\vec{a}_0)Pr(\vec{o}_0|s'',\vec{a}_0)V^{\vec{q}'}(s'') \tag{4.20}$$

The value function of an individual policy π^j for agent j given a belief $\tilde{b}_t^j$ on the states of the system and the tests of agent i is:

$$V^{\pi^j}(\tilde{b}_t^j) = \sum_{s\in\mathcal{S}} \sum_{q^i\in\{\mathcal{A}^i\times\mathcal{O}^i\}^{H-t}} \sum_{q^j\in\{\mathcal{A}^j\times\mathcal{O}^j\}^{H-t}} Pr(s,\langle q^i,q^j\rangle|\tilde{b}_t^j,\pi^j)V^{\langle q^i,q^j\rangle}(s) \tag{4.21}$$

The value $V^{\langle q^i,q^j\rangle}(s)$ is defined in Equation 4.14. Since $\tilde{b}_t^j$ is a sufficient statistics for the state and tests of agent i, and π^j indicates exactly which tests agent j will execute, we have:

$$\begin{aligned}
Pr(s,\langle q^i,q^j\rangle|\tilde{b}_t^j,\pi^j) &= Pr(s|\tilde{b}_t^j)Pr(q^i|\tilde{b}_t^j)Pr(q^j|\pi^j) \\
&= Pr(s|\tilde{b}_t^j)\sum_{q'^i\in\mathcal{Q}_t^i} Pr(q'^i|\tilde{b}_t^j)m_{q^i}(q'^i)\sum_{q'^j\in\mathcal{Q}_t^j} Pr(q'^j|\pi^j)m_{q^j}(q'^j) \\
&= Pr(s|\tilde{b}_t^j)\sum_{q'^i\in\mathcal{Q}_t^i} Pr(q'^i|\tilde{b}_t^j)m_{q^i}(q'^i)\sum_{\substack{q'^j\in\mathcal{Q}_t^j \\ F_t^j(\pi^j,q'^j)=1}} m_{q^j}(q'^j) \\
&= \sum_{q'^i\in\mathcal{Q}_t^i}\sum_{\substack{q'^j\in\mathcal{Q}_t^j \\ F_t^j(\pi^j,q'^j)=1}} \tilde{b}_t^j(s,q'^i)m_{q^i}(q'^i)m_{q^j}(q'^j)
\end{aligned} \tag{4.22}$$

By substituting Equation 4.22 in Equation 4.21, we find:

$$V^{\pi^j}(\tilde{b}_t^j) = \sum_{s\in\mathcal{S}} \sum_{q^i\in\{\mathcal{A}^i\times\mathcal{O}^i\}^{H-t}} \sum_{q^j\in\{\mathcal{A}^j\times\mathcal{O}^j\}^{H-t}} \sum_{q'^i\in\mathcal{Q}_t^i} \sum_{\substack{q'^j\in\mathcal{Q}_t^j \\ F_t^j(\pi^j,q'^j)=1}} \tilde{b}_t^j(s,q'^i)m_{q^i}(q'^i)m_{q^j}(q'^j)V^{\langle q^i,q^j\rangle}(s)$$

$$= \sum_{s\in\mathcal{S}}\sum_{q'^i\in\mathcal{Q}_t^i}\sum_{\substack{q'^j\in\mathcal{Q}_t^j \\ F_t^j(\pi^j,q'^j)=1}} \tilde{b}_t^j(s,q'^i) \underbrace{\sum_{q^i\in\{\mathcal{A}^i\times\mathcal{O}^i\}^{H-t}} \sum_{q^j\in\{\mathcal{A}^j\times\mathcal{O}^j\}^{H-t}} m_{q^i}(q'^i)m_{q^j}(q'^j)V^{\langle q^i,q^j\rangle}(s)}_{\tilde{V}^{\langle q'^i,q'^j\rangle}(s)}$$

$$= \sum_{s\in\mathcal{S}}\sum_{q'^i\in\mathcal{Q}_t^i}\sum_{\substack{q'^j\in\mathcal{Q}_t^j \\ F_t^j(\pi^j,q'^j)=1}} \tilde{b}_t^j(s,q'^i)\tilde{V}^{\langle q'^i,q'^j\rangle}(s) \tag{4.23}$$

where:

$$\tilde{V}^{\langle q'^i,q'^j\rangle}(s) \stackrel{def}{=} \sum_{q^i\in\{\mathcal{A}^i\times\mathcal{O}^i\}^{H-t}} \sum_{q^j\in\{\mathcal{A}^j\times\mathcal{O}^j\}^{H-t}} m_{q^i}(q'^i)m_{q^j}(q'^j)V^{\langle q^i,q^j\rangle}(s) \tag{4.24}$$

We substituted the probability of each test q^i with a linear combination of the probabilities $\tilde{b}_t^j(s,q'^i)$ of the core tests q'^i, and the probability of each test q^j with a linear combination of the probabilities $F_t^j(\pi^j,q'^j)\in\{0,1\}$ of the core tests q'^j. The reduced vector $\tilde{V}^{\langle q'^i,q'^j\rangle}$ is the contribution of $\langle q'^i,q'^j\rangle$ to the value of a joint policy, by including a proportion of the values of all the tests that depend on $\langle q'^i,q'^j\rangle$. If one wants to calculate the value of a policy π^j for a given multi-agent belief state $\tilde{b}_t^j$, all one needs are the vectors $\tilde{V}^{\langle q'^i,q'^j\rangle}$ and the matrix F_t^j (Equation 4.23).

At time step $t < H$, we know from Theorem 2 that the core tests are extensions of the core test at time-step $t+1$, i.e. they are of the form $a^i o^i q^i$ and $a^j o^j q^j$, where $q^i \in Q^i_{t+1}$ and $q^j \in Q^j_{t+1}$. We will now show how to calculate the value vectors $\tilde{V}^{\langle a^i o^i q^i, a^j o^j q^j \rangle}$ given the value vectors $\tilde{V}^{\langle q^i, q^j \rangle}$ of the previous horizon, where $q_i \in \mathcal{Q}^i_{t+1}$ and $q_j \in \mathcal{Q}^j_{t+1}$. At time-step $t = H$, a policy corresponds to a single action. There are then $\mathcal{A}^i$ core test at $t = H$, i.e. $\mathcal{Q}^i_H = \mathcal{A}^i$, and $\tilde{V}^{\langle a^i, a^j \rangle}(s) = R(s, \langle a^i, a^j \rangle)$. From Equation 4.24, we have:

$$\tilde{V}^{\langle a^i o^i q^i, a^j o^j q^j \rangle}(s) = \sum_{\substack{a'^i o'^i q'^i \in \{\mathcal{A}^i \times \mathcal{O}^i\}^{H-t+1} \\ a'^j o'^j q'^j \in \{\mathcal{A}^j \times \mathcal{O}^j\}^{H-t+1}}} m_{a'^i o'^i q'^i}(a^i o^i q^i) m_{a'^j o'^j q'^j}(a^j o^j q^j) V^{\langle a'^i o'^i q'^i, a'^j o'^j q'^j \rangle}(s)$$

$$= \sum_{\substack{a'^i o'^i q'^i \in \{\mathcal{A}^i \times \mathcal{O}^i\}^{H-t+1} \\ a'^j o'^j q'^j \in \{\mathcal{A}^j \times \mathcal{O}^j\}^{H-t+1}}} m_{a'^i o'^i q'^i}(a^i o^i q^i) m_{a'^j o'^j q'^j}(a^j o^j q^j) \beta_0(s, \langle a'^i o'^i q'^i, a'^j o'^j q'^j \rangle) R(s, \langle a'^i, a'^j \rangle)$$

$$+ \sum_{\substack{a'^i o'^i q'^i \in \{\mathcal{A}^i \times \mathcal{O}^i\}^{H-t+1} \\ a'^j o'^j q'^j \in \{\mathcal{A}^j \times \mathcal{O}^j\}^{H-t+1}}} m_{a'^i o'^i q'^i}(a^i o^i q^i) m_{a'^j o'^j q'^j}(a^j o^j q^j) \gamma \sum_{s' \in \mathcal{S}} [Pr(s', \langle o'^i, o'^j \rangle | s, \langle a'^i, a'^j \rangle)$$

$$V^{\langle q'^i, q'^j \rangle}(s')]$$

(From Equation 4.20)

$$= \sum_{\substack{a'^i o'^i q'^i \in \{\mathcal{A}^i \times \mathcal{O}^i\}^{H-t+1} \\ a'^j o'^j q'^j \in \{\mathcal{A}^j \times \mathcal{O}^j\}^{H-t+1}}} m_{a'^i o'^i q'^i}(a^i o^i q^i) m_{a'^j o'^j q'^j}(a^j o^j q^j) \beta_0(s, \langle a'^i o'^i q'^i, a'^j o'^j q'^j \rangle) R(s, \langle a^i, a^j \rangle)$$

$$+ \sum_{\substack{q'^i \in \{\mathcal{A}^i \times \mathcal{O}^i\}^{H-t} \\ q'^j \in \{\mathcal{A}^j \times \mathcal{O}^j\}^{H-t}}} m_{q'^i}(q^i) m_{q'^j}(q^j) \gamma \sum_{s' \in \mathcal{S}} Pr(s', \langle o'^i, o'^j \rangle | s, \langle a'^i, a'^j \rangle) V^{\langle q'^i, q'^j \rangle}(s')$$

(From Equation 4.13)

$$= R(s, \langle a^i, a^j \rangle) \sum_{\substack{a'^i o'^i q'^i \in \{\mathcal{A}^i \times \mathcal{O}^i\}^{H-t+1} \\ a'^j o'^j q'^j \in \{\mathcal{A}^j \times \mathcal{O}^j\}^{H-t+1}}} m_{a'^i o'^i q'^i}(a^i o^i q^i) m_{a'^j o'^j q'^j}(a^j o^j q^j) \beta_0(s, \langle a'^i o'^i q'^i, a'^j o'^j q'^j \rangle)$$

$$+ \gamma \sum_{s' \in \mathcal{S}} Pr(s', \langle o'^i, o'^j \rangle | s, \langle a'^i, a'^j \rangle) \underbrace{\sum_{\substack{q'^i \in \{\mathcal{A}^i \times \mathcal{O}^i\}^{H-t} \\ q'^j \in \{\mathcal{A}^j \times \mathcal{O}^j\}^{H-t}}} m_{q'^i}(q^i) m_{q'^j}(q^j) V^{\langle q'^i, q'^j \rangle}(s')}_{\tilde{V}^{\langle q^i, q^j \rangle}(s')}$$

Therefore:

$$\tilde{V}^{\langle a^i o^i q^i, a^j o^j q^j \rangle}(s) = R(s, \langle a^i, a^j \rangle) C^{\langle a^i o^i q^i, a^j o^j q^j \rangle}(s) + \gamma \sum_{s' \in S} Pr(\langle o_i, o_j \rangle, s' | s, \langle a_i, a_j \rangle) \tilde{V}^{\langle q^i, q^j \rangle}(s') \tag{4.25}$$

where:

$$C^{\langle a^i o^i q^i, a^j o^j q^j \rangle}(s) \stackrel{def}{=} \sum_{\substack{a'^i o'^i q'^i \in \{\mathcal{A}^i \times \mathcal{O}^i\}^{H-t+1} \\ a'^j o'^j q'^j \in \{\mathcal{A}^j \times \mathcal{O}^j\}^{H-t+1}}} m_{a'^i o'^i q'^i}(a^i o^i q^i) m_{a'^j o'^j q'^j}(a^j o^j q^j) \beta_0(s, \langle a'^i o'^i q'^i, a'^j o'^j q'^j \rangle) \tag{4.26}$$

The vectors $C^{\langle a^i o^i q^i, a^j o^j q^j \rangle}$ can be recursively calculated using the previous horizon vectors $C^{\langle q^i, q^j \rangle}$:

$$
\begin{aligned}
C^{\langle a^i o^i q^i, a^j o^j q^j \rangle}(s) &= \sum_{\substack{a'^i o'^i q'^i \in \{\mathcal{A}^i \times \mathcal{O}^i\}^{H-t+1} \\ a'^j o'^j q'^j \in \{\mathcal{A}^j \times \mathcal{O}^j\}^{H-t+1}}} m_{a'^i o'^i q'^i}(a^i o^i q^i) m_{a'^j o'^j q'^j}(a^j o^j q^j) \beta_0(s, \langle a'^i o'^i q'^i, a'^j o'^j q'^j \rangle) \\
&= \sum_{\substack{q'^i \in \{\mathcal{A}^i \times \mathcal{O}^i\}^{H-t} \\ q'^j \in \{\mathcal{A}^j \times \mathcal{O}^j\}^{H-t}}} m_{q'^i}(q^i) m_{q'^j}(q^j) \beta_0(s, \langle a^i o^i q^i, a^j o^j q'^j \rangle) \quad \text{(From Equation 4.13)} \\
&= \sum_{s' \in S} Pr(s', \langle o^i, o^j \rangle | s, \langle a^i, a^j \rangle) \underbrace{\sum_{\substack{q'^i \in \{\mathcal{A}^i \times \mathcal{O}^i\}^{H-t} \\ q'^j \in \{\mathcal{A}^j \times \mathcal{O}^j\}^{H-t}}} m_{q'^i}(q^i) m_{q'^j}(q^j) \beta_0(s', \langle q'^i, q'^j \rangle)}_{C^{\langle q^i, q^j \rangle}(s')} \\
&\qquad \text{(From Equation 4.16)} \\
&= \sum_{s' \in \mathcal{S}} Pr(s', \langle o^i, o^j \rangle | s, \langle a^i, a^j \rangle) C^{\langle q^i, q^j \rangle}(s') \qquad (4.27)
\end{aligned}
$$

In order to calculate the value of a policy π^j given a reduced belief state, we need to know the value vectors $\tilde{V}^{\langle q^i, q^j \rangle}$ (Equation 4.23). These latter vectors are recursively calculated by using the contribution probability vectors $C^{\langle q^i, q^j \rangle}$ (Equation 4.25). Finally, the contribution probability vectors $C^{\langle q^i, q^j \rangle}$ are also recursively calculated (Equation 4.27).

Now, we show how to update the reduced value vectors $\tilde{V}$ and the probability contribution vectors C after that a core test q^i is removed from Q_t^i. This can happen after removing a policy from Π_t^i, i.e. a row in the matrix U_t^i (see Figure 4.4), which may lead to a linear dependency between the columns of U_t^i. Let m_{q^i} be the weights vector of a former core test q^i, i.e.

$$
Pr(q^i | h_t) = \sum_{q'^i \in Q_t^i - \{q^i\}} m_{q^i}(q'^i) Pr(q'^i | h_t) \qquad (4.28)
$$

Let q''^i be any test to be executed at time-step t, we have

$$
\begin{aligned}
Pr(q''^i | h_t) &= \sum_{q'^i \in Q_t^i} m_{q''^i}(q'^i) Pr(q'^i | h_t) \\
&= \sum_{q'^i \in Q_t^i - \{q^i\}} m_{q''^i}(q'^i) Pr(q'^i | h_t) + m_{q''^i}(q^i) Pr(q^i | h_t) \\
&= \sum_{q'^i \in Q_t^i - \{q^i\}} m_{q''^i}(q'^i) Pr(q'^i | h_t) + m_{q''^i}(q^i) \sum_{q'^i \in Q_t^i - \{q^i\}} m_{q^i}(q'^i) Pr(q'^i | h_t) \\
&\qquad \text{(From Equation 4.28)} \\
&= \sum_{q'^i \in Q_t^i - \{q^i\}} [m_{q''^i}(q'^i) + m_{q^i}(q'^i) m_{q''^i}(q^i)] Pr(q'^i | h_t) \\
&= \sum_{q'^i \in Q_t^i - \{q^i\}} m'_{q''^i}(q'^i) Pr(q'^i | h_t)
\end{aligned}
$$

where the new weight vector $m'_{q''^i}$ is defined as:

$$m'_{q''^i}(q'^i) = m_{q''^i}(q'^i) + m_{q^i}(q'^i)m_{q''^i}(q^i)$$

Notice that eliminating policies from Π_t^i cannot increase the rank of the matrix U_t^i (see Figure 4.4). If a column in U_t^i is linearly dependent on some other columns, then it remains linearly dependent on the same columns, with the same weights, after removing any policy.

Instead of calculating the new weight vectors $m'_{q''^i}$ and using them to calculate the new value vectors $\tilde{V}'$ and the probability contribution vectors C', we show how to directly calculate $\tilde{V}'$ and C'. From definition 4.24, the new reduced value vector after removing a core test q^i is given by:

$$\begin{aligned}
\tilde{V}'^{\langle q''^i, q''^j\rangle}(s) &= \sum_{q'^i \in \{\mathcal{A}^i\times\mathcal{O}^i\}^{H-t}} \sum_{q'^j \in \{\mathcal{A}^j\times\mathcal{O}^j\}^{H-t}} m'_{q'^i}(q''^i)m_{q'^j}(q''^j)V^{\langle q'^i, q'^j\rangle}(s) \\
&= \sum_{q'^i \in \{\mathcal{A}^i\times\mathcal{O}^i\}^{H-t}} \sum_{q'^j \in \{\mathcal{A}^j\times\mathcal{O}^j\}^{H-t}} [m_{q'^i}(q''^i) + m_{q^i}(q''^i)m_{q'^i}(q^i)]m_{q'^j}(q''^j)V^{\langle q'^i, q'^j\rangle}(s) \\
&= \tilde{V}^{\langle q''^i, q''^j\rangle}(s) + m_{q^i}(q''^i) \sum_{q'^i \in \{\mathcal{A}^i\times\mathcal{O}^i\}^{H-t}} \sum_{q'^j \in \{\mathcal{A}^j\times\mathcal{O}^j\}^{H-t}} m_{q'^i}(q^i)m_{q'^j}(q''^j)V^{\langle q'^i, q'^j\rangle}(s) \\
&= \tilde{V}^{\langle q''^i, q''^j\rangle}(s) + m_{q^i}(q''^i)\tilde{V}^{\langle q^i, q''^j\rangle}(s) \qquad (4.29)
\end{aligned}$$

Similarly, from Equation 4.26, the new contribution probability vector after removing a core test q^i is given by:

$$\begin{aligned}
C'^{\langle q''^i, q''^j\rangle}(s) &= \sum_{\substack{q'^i \in \{\mathcal{A}^i\times\mathcal{O}^i\}^{H-t} \\ q'^j \in \{\mathcal{A}^j\times\mathcal{O}^j\}^{H-t}}} m'_{q'^i}(q''^i)m_{q'^j}(q''^j)\beta_0(s, \langle q'^i, q'^j\rangle) \\
&= \sum_{\substack{q'^i \in \{\mathcal{A}^i\times\mathcal{O}^i\}^{H-t} \\ q'^j \in \{\mathcal{A}^j\times\mathcal{O}^j\}^{H-t}}} [m_{q'^i}(q''^i) + m_{q^i}(q''^i)m_{q'^i}(q^i)]m_{q'^j}(q''^j)\beta_0(s, \langle q'^i, q'^j\rangle) \\
&= \tilde{C}^{\langle q''^i, q''^j\rangle}(s) + m_{q^i}(q''^i) \sum_{q'^i \in \{\mathcal{A}^i\times\mathcal{O}^i\}^{H-t}} \sum_{q'^j \in \{\mathcal{A}^j\times\mathcal{O}^j\}^{H-t}} m_{q'^i}(q^i)m_{q'^j}(q''^j)\beta_0(s, \langle q'^i, q'^j\rangle) \\
&= \tilde{C}^{\langle q''^i, q''^j\rangle}(s) + m_{q^i}(q''^i)\tilde{C}^{\langle q^i, q''^j\rangle}(s) \qquad (4.30)
\end{aligned}$$

In summary, in order to calculate the value of a policy π^j given a reduced belief $\tilde{b}_t^j$ using Equation 4.23, we will need to calculate the reduced value vectors $\tilde{V}^{\langle q'^i, q'^j\rangle}$. These vectors are backed up from those of the previous horizon using Equation 4.25 and the contribution probability vectors C. Finally, the vector C is also backed up from the previous horizon using Equation 4.27.

In the next section, we present a dynamic programming algorithm where the policies are evaluated in the space of core tests beliefs, using the reduced value vectors introduced in this section.

4.5.5 Dynamic Programming Algorithm For Decentralized POMDPs with Policy Space Compression

Algorithm 8 describes the principal steps of a dynamic programming algorithm where predictive policy representations are used for compressing the policies. As in the original dynamic programming algorithm for decentralized POMDPs (Algorithm 4.4), the sets of policies Π_t^i and Π_t^j are explicitly represented by decision trees. However, the value vectors are replaced by the reduced value vectors $\tilde{V}_t$, defined in Equation 4.24, and the probability contribution vector C_t, defined in Equation 4.26, which have a smaller dimensionality. We also need to keep in memory the sets of core tests Q_t^i and Q_t^j, as well as the matrices F_t^i and F_t^j. Since these latter matrices are binary, they are implicitly represented by specifying for each core test the list of policies where this test occurs.

At time-step $t = H$, we have $\Pi_H^i = Q_H^i = \mathcal{A}^i$, $\Pi_H^j = Q_H^j = \mathcal{A}^j$, and $\forall a^i \in \mathcal{A}^i, \forall a^j \in \mathcal{A}^j$, $\forall s \in \mathcal{S} : \tilde{V}^{\langle a^i, a^j \rangle}(s) = R(s, \langle a^i, a^j \rangle)$ and $C^{\langle a^i, a^j \rangle}(s) = \beta_H(s, \langle a^i, a^j \rangle) = 1$ by definition (Equation 4.16). The sets Π_t^i and Π_t^j contain all the possible policies that can be executed at time step $t < H$ such that the sub-policies to be executed at time-step $t+1$ are contained in the sets Π_{t+1}^i and Π_{t+1}^j respectively (step 1). The sets of core tests at time-step $t < H$ are constructed by extending the core tests contained in Q_{t+1}^i and Q_{t+1}^j by one observation and one action (step 2, from Theorem 2).

The next steps (4 and 5) consist in calculating the new contribution probability vector C_t by using the vectors C_{t+1} (Equation 4.27), as well as the new reduced value vectors $\tilde{V}_t$ using the vectors C_t and $\tilde{V}_{t+1}$ (Equation 4.25). The value vectors $\tilde{V}_t$ are used to determine which policies of agents i and j are dominated and should be removed (steps 7 and 8). To determine if a policy π^i is dominated or not, the following linear program is solved:

$$
\begin{aligned}
&\textit{minimize:} \quad \epsilon \qquad\qquad (4.31)\\
&\textit{subject to:}\\
&\forall s \in \mathcal{S}, \forall q^j \in Q_t^j :\\
&\quad 0 \leqslant \tilde{b}^i(s, q^j) \leqslant 1\\
&\forall \pi_t'^i \in \Pi_t^i - \{\pi_t^i\} :\\
&\underbrace{\sum_{s \in \mathcal{S}} \sum_{\substack{q^i \in Q_t^i, F_t^i[\pi^i, q^i]=1 \\ q^j \in Q_t^j}} \tilde{b}^i(s, q^j) \tilde{V}^{\langle q^i, q^j \rangle}(s)}_{\text{the value of policy } \pi^i \text{ in the belief } \tilde{b}^i} - \underbrace{\sum_{s \in \mathcal{S}} \sum_{\substack{q^i \in Q_t^i, F_t^i[\pi'^i, q^i]=1 \\ q^j \in Q_t^j}} \tilde{b}^i(s, q^j) \tilde{V}^{\langle q^i, q^j \rangle}(s)}_{\text{the value of policy } \pi'^i \text{ in the belief } \tilde{b}^i} + \epsilon \geqslant 0
\end{aligned}
$$

The variables of this linear program are: the margin ϵ, and the probabilities $\tilde{b}^i(.,.)$ of the reduced belief state, we have then $|\mathcal{S}||Q_t^j|+1$ variables. There are also $|\mathcal{S}||Q_t^j|$ linear constraints specifying that the probabilities of the belief states $\tilde{b}^i$ are between 0 and 1. The second set of constraints specifies that the minimum margin between the value of any policy $\pi'^i \neq \pi^i$ and that of policy π^i for each belief state $\tilde{b}^i$. A policy π^i is weakly dominated if and only if $\epsilon > 0$

Input: Q^i_{t+1},Q^j_{t+1},Π^i_{t+1},Π^j_{t+1}, $\tilde{V}_{t+1}$, and C_{t+1} ;
$\Pi^i_t = fullBackup(\Pi^i_{t+1})$, $\Pi^j_t = fullBackup(\Pi^j_{t+1})$;
$Q^i_t = \mathcal{A}^i \times \mathcal{O}^i \times Q^i_{t+1}$, $Q^i_t = \mathcal{A}^i \times \mathcal{O}^i \times Q^i_{t+1}$;
The binary matrices F^i_t and F^j_t are implicitly represented by Π^i_t, Π^j_t,Q^i_t and Q^j_t ;
Calculate the probability contribution vectors C_t by using C_{t+1} (Equation 4.27);
Calculate the reduced value vectors $\tilde{V}_t$ by using $\tilde{V}_{t+1}$ (Equation 4.25);
repeat
    Remove the policies of Π^i_t that are dominated (Linear Program 4.32);
    Remove the policies of Π^j_t that are dominated (Linear Program 4.32);
until *no more policies in* Π^i_t *or* Π^j_t *can be removed* ;
Use a matrix decomposition method (such as Gauss-Jordan elimination) to find linearly dependent columns in F^i_t and F^j_t, and remove the corresponding tests from Q^i_t and Q^j_t;
foreach *core test* $q^i \in Q^i_t$ **do**
    foreach *core test* $q^j \in Q^j_t$ **do**
        foreach *core test* q'^i *removed from* Q^i_t **do**
            $\tilde{V}^{\langle q^i,q^j\rangle} \leftarrow \tilde{V}^{\langle q^i,q^j\rangle} + m_{q'^i}(q^i)\tilde{V}^{\langle q'^i,q^j\rangle}(s)$ (Equation 4.29) ;
            $C^{\langle q^i,q^j\rangle} \leftarrow C^{\langle q^i,q^j\rangle} + m_{q'^i}(q^i)C^{\langle q'^i,q^j\rangle}(s)$ (Equation 4.30) ;
        end
        foreach *core test* q'^j *removed from* Q^j_t **do**
            $\tilde{V}^{\langle q^i,q^j\rangle} \leftarrow \tilde{V}^{\langle q^i,q^j\rangle} + m_{q'^j}(q^j)\tilde{V}^{\langle q^i,q'^j\rangle}(s)$ (Equation 4.29) ;
            $C^{\langle q^i,q^j\rangle} \leftarrow C^{\langle q^i,q^j\rangle} + m_{q'^j}(q^j)C^{\langle q^i,q'^j\rangle}(s)$ (Equation 4.30) ;
        end
    end
end
Output: Q^i_t,Q^j_t,Π^i_t,Π^j_t, $\tilde{V}_t$, and C_t ;

Algorithm 8: Dynamic Programming for decentralized POMDPs with Policy Space Compression.

(see the discussion of the linear program 4.4 in Section 4.2.2). Therefore, this linear program contains a total of $|\mathcal{S}||Q^j_t| + |\Pi| - 1$ constraints.

Notice that contrary to the original belief state $b^i(s,.)$ which is a probability distribution on the policies of agent j multiplied by the probability of state s, the reduced belief space $\tilde{b}^i(s,.)$ is not necessarily a probability distribution, because the tests are not mutually exclusive events. In Figure 4.4 for example, if $b^i(s,\pi_1) = 1$ then $\tilde{b}^i(s,q_1) = 1$ and $\tilde{b}^i(s,q_3) = 1$. The relation between different tests is more complex and more constraints should be considered to ensure that any reduced belief considered in the linear program will really correspond to some belief in the original space. For instance, if two tests contain the same sequence of observations, then their corresponding probabilities should sum to less than 1. In fact, any belief $b^i(s,.)$ can be transformed into a reduced belief $\tilde{b}^i(s,.) = [b^i(s,.)]^T F^j$ (from Equation 4.10), but given a vector $\tilde{b}^i(s,.)$ with all entries between 0 and 1, there is not necessarily a belief $b^i(s,.)$ in

the original space such that $\tilde{b}^i(s,.) = [b^i(s,.)]^T F^j$. To solve this problem, we should add the following constraint:

$$\sum_{s\in\mathcal{S}} \sum_{q^j\in\{\mathcal{A}^j\times\mathcal{O}^j\}^H} \sum_{q'^j\in Q_t^j} m_{q^j}(q'^j)\tilde{b}^i(s,q^j) = 1 \tag{4.32}$$

However, one needs to keep the weight vectors m_{q^j} of all the tests $q^j \in \{\mathcal{A}^j\times\mathcal{O}^j\}^H$ in order to define this new constraint. Nevertheless, even without adding this constraint, if a policy π^i is not dominated then there is at least one belief state $b_i(.,.)$ where $V^{\pi^i}(b^i) > V_{\pi'^i}(b^i)$, $\forall\pi'^i \in \Pi^i - \{\pi^i\}$, and in the corresponding reduced belief $\tilde{b}^i(s,.) = [b^i(s,.)]^T F^j$, we will also have $V^{\pi^i}(\tilde{b}^i) > V_{\pi'^i}(\tilde{b}^i), \forall\pi'^i \in \Pi^i - \{\pi^i\}$. Therefore, the linear program 4.32 keeps at least all the dominant policies, but can also keep some dominated policies, we will return to this question in the next section.

After removing dominated policies, some core tests become linearly dependent (the columns corresponding to these tests in matrices F^i and F^j become linearly dependant). These latter tests are detected by using a matrix decomposition technique, such as Gauss-Jordan (step 10). Finally, the vectors $\tilde{V}$ and C are updated by using Equations 4.29 and 4.30 (steps 11 to 12). The algorithm returns: the sets of core tests Q_t^i and Q_t^j, the sets of policies Π_t^i and Π_t^j, the reduced value vectors $\tilde{V}_t$, and the probability vector C_t. This output is used as an input to in the next iteration, until $t = 0$.

4.5.6 Computational complexity

For the sake of simplifying the analysis, we assume that agents i and j have the same number of actions and observations, and $|\mathcal{O}_i| > 1$. The worst-case complexity of steps 1 to 5 is upper-bounded by the complexity of the full backup operations, which is $O(|\mathcal{A}_i||\Pi_{t+1}^j|^{|\mathcal{O}_i|})$, where $|\Pi_{t+1}^j| = |\mathcal{A}_i|^{\frac{|\mathcal{O}_i|^H-1}{|\mathcal{O}_i|-1}}$, thus, the complexity of this latter operation is $O(|\mathcal{A}_i|^{\frac{|\mathcal{O}_i|^H-1}{|\mathcal{O}_i|-1}})$. However, the longest operation in this algorithm, as well as in the original dynamic programming algorithm 6, is the pruning of weakly dominated policies (steps 6 to 9) where up to $|\mathcal{A}_i|^{\frac{|\mathcal{O}_i|^H-1}{|\mathcal{O}_i|-1}}$ calls of a linear program solver can be made (Equation 4.32). The worst-case computational complexity of solving a linear program with n variable and m constraints, using Karmarkar's algorithm for example, is: $O(n^{4.5}m)$ [Powell, 1993]. Therefore, the complexity of the pruning loop of our algorithm is

$$O(\underbrace{|\mathcal{A}_i|^{\frac{|\mathcal{O}_i|^H-1}{|\mathcal{O}_i|-1}}}_{\text{number of iterations}} \underbrace{[|\mathcal{S}|\ \underbrace{|\mathcal{A}_i|^H|\mathcal{O}_i|^H}_{\text{number of tests}} +1]^{4.5}}_{\text{number of variables}} \underbrace{[(|\mathcal{A}_i|^{\frac{|\mathcal{O}_i|^H-1}{|\mathcal{O}_i|-1}} - 1)(|\mathcal{S}|+1)]}_{\text{number of constraints}})$$

which can be reduced to $O(|\mathcal{A}_i|^{2|\mathcal{O}_i|^{H-1}})$. This is substantially smaller than the complexity of the original dynamic programming algorithm:

$$O(\underbrace{|\mathcal{A}_i|^{\frac{|\mathcal{O}_i|^H-1}{|\mathcal{O}_i|-1}}}_{\text{number of iterations}}\underbrace{[|\mathcal{S}|\underbrace{|\mathcal{A}_i|^{\frac{|\mathcal{O}_i|^H-1}{|\mathcal{O}_i|-1}}}_{\text{number of policies}}+1]^{4.5}}_{\text{number of variables}}\underbrace{[(|\mathcal{A}_i|^{\frac{|\mathcal{O}_i|^H-1}{|\mathcal{O}_i|-1}}-1)(|\mathcal{S}|+1)+1]}_{\text{number of constraints}})$$

which can be reduced to $O(|\mathcal{A}_i|^{6.5|\mathcal{O}_i|^{H-1}})$.

This is basically due to the fact that the number of tests is a polynomial function of the number of observations and an exponential function of the horizon, whereas the number of policies is exponential in the number of observations and doubly exponential in the horizon.

Notice that our algorithm generates a small overhead due to the decomposition of matrices F_t^i and F_t^j (step 11), which is achieved in $O(|\mathcal{A}_i|^{\frac{|\mathcal{O}_i|^H-1}{|\mathcal{O}_i|-1}}|\mathcal{A}_i|^{2H}|\mathcal{O}_i|^{2H})$ by using Gauss-Jordan elimination, and due to the update of the reduced value vectors and the contribution probability vector, which is achieved in $O(|\mathcal{A}_i|^{3H}|\mathcal{O}_i|^{3H})$.

The major advantage of our technique is related to the efficient use of the memory space. In both algorithms 6 and 4.4, most of the memory is allocated for saving (reduced) value vectors. The size of value vectors is $O(|\mathcal{S}||\mathcal{A}_i|^{2^{\frac{|\mathcal{O}_i|^H-1}{|\mathcal{O}_i|-1}}})$, while the size of reduced value vectors is only $O(|\mathcal{S}||\mathcal{A}_i|^{2H}|\mathcal{O}_i|^{2H})$.

Finally, we should emphasize the fact that this is only a worst-case analysis. Given an arbitrary decentralized POMDP problem, Algorithm 4.4 may keep more policies than Algorithm 6 due to the fact that linear program 4.32 is underconstrained, which may lead to a higher number of core tests and a longer runtime. However, this problem can be solved by adding the constraint 4.32 (see the related discussion in Section 4.5.5) and introducing some modifications to Algorithm 4.4, without a major impact on the computational complexity of the algorithm. In the next section, we present an empirical comparison of Algorithms 4.4 and 6, and show that this latter issue is not encountered in practice.

4.5.7 Empirical Analysis

We implemented both of Algorithm 6 (DP) and Algorithm 8 (DP with Policy Compression) using ILOG Cplex 10 solver on an AMD Athlon machine with a 1.80 GHz processor and 1.5 GB RAM. We used Gauss-Jordan elimination method to find the core tests. For the last step of planning, we omit pruning the dominated policies since we will not use them to generate further policies, thus, we only generate the value vectors of each joint policy and each joint test. We compared the performances of these two algorithms on two benchmark problems MA-Tiger and MABC Hansen et al. [2004]. Both of the two algorithms find the same optimal

values. However, the memory space used to represent value vectors is significantly smaller when we use the compression approach. In fact, the value vectors in the original DP algorithm are defined on states and policies, whereas the reduced value vectors are defined on states and core tests. Notice also that the compression ratio (policies number/core tests number) grows exponentially w.r.t the planning horizon. Also, the runtime of DP is improved when the compression algorithm is used. Indeed, the backup of reduced value vectors (equations 4 and 5) takes less time than the backup of original value vectors (equation 1). However, given that the linear program of Table 2 is under-constrained, our algorithm can keep some additional policies that are dominated. In MABC for example, our algorithm generates 3528×3528 joint policies at the beginning of horizon 4, while only 1800×1458 joint policies are not dominated. This problem can be solved by adding a larger system of constraints on the probabilities of the tests, but the computational efficiency will be possibly affected. We can see that as in most of the compression techniques, there is a tradeoff between the space performance and the time performance.

		DP		**DP with Policy Compression**			
Problem	H	runtime	policies	runtime	policies	core tests	ratio
MA-Tiger	2	0.20	(27,27)	0.17	(27,27)	(18,18)	1.5
	3	2.29	(675,675)	1.79	(675,675)	(90,90)	7.5
	4	-	-	534.90	(195075,195075)	(540,540)	361.25
MABC	2	0.12	(8,8)	0.14	(8,8)	(8,8)	1
	3	0.46	(72,72)	0.36	(72,72)	(24,24)	3
	4	17.59	(1800,1458)	4.59	(3528,3528)	(80,80)	44.1

Table 4.2: The runtime (in seconds) and the number of policies and tests of Dynamic Programming (DP) algorithms, with and without compression. The compression ratio is the number of policies divided by the number of core tests.

4.6 Conclusion

In many multiagent systems, the uncertainty of an agent is not only about the environment states, but also about the policies of other agents. Decentralized POMDPs are an extension of POMDPs that provides a rich mathematical framework for dealing with this type of uncertainty. The dimensionality of the policy space is a crucial factor in the complexity of Dynamic Programming algorithms for decentralized POMDPs. In this thesis, we proposed a new model for representing the agent's belief state based on predicting other agents future actions. The advantage of this model, called Predictive Policy Representations (PPRs), is that the agents use only a minimal and sufficient amount of information to represent their beliefs. We compared the computational performance of a point based algorithm for decentralized POMDP

using decisions trees to the performance of the same algorithm using a simplified version of PPRs, and the preliminary results were promising.

Based on these results, we developed a second Dynamic Programming algorithm exploiting the potential dependencies between different tests to reduce even more the dimensionality of the belief points. This approach is based on projecting the policy beliefs from the high dimensional space of trees to the low dimensional space of tests, and using matrix factorization methods to reduce the number of tests. Consequently, the memory space used in this algorithm is significantly smaller, while the runtime is lower compared to the original Dynamic Programming algorithm. This method can also be used in approximate Dynamic Programming, mainly for domains with large observations spaces. However, the major drawback of this new approach is the relaxation of necessary constraints on the belief states, leading to a larger belief space containing invalid beliefs. As a consequence of that, the number of policies retained in our algorithm is always higher than in the original algorithm.

A next step in developing this approach furthermore would be to investigate this issue of constraining the reduced belief states, and to find a tradeoff between the number of constraints used for defining the belief space and the number of policies retained by the algorithm. Another possible extension is to use a fast and lossy matrix factorization method in our algorithm, and more particularly, a factorization method for binary matrices.

Chapter 5

Reinforcement Learning with Predictive Representations

Reinforcement Learning (RL) refers to the problem of finding a policy for controlling a dynamical system when the parameters of this system are, completely or partly, unknown. This problem becomes particulary challenging when the states of the system are partially observable, because the policy can no longer be expressed as a function of the states of the system. A promising approach for solving this problem consists in maintaining a parametric policy, defined on belief states or histories, and using the experience of interacting with the system in order to calculate the gradient of the policy. This gradient is then used for tuning the parameters of the policy and gradually improving its performance. In this chapter, we consider the problem of estimating the policy gradient in Partially Observable Markov Decision Processes (POMDPs) with a Predictive Policy Representation (PPR), which was presented in the previous chapter (Chapter 4). We compare PPR policies to Finite-State Machines (FSMs), as a standard model for policy gradient methods in POMDPs, and we present a general Actor-Critic algorithm for learning both FSMs and PPRs. The critic part computes a value function that takes as variables the parameters of the policy. These latter parameters are gradually updated to maximize the value function. We also show that the value function is polynomial for both FSMs and PPR policies, with a potentially smaller degree in the case of PPRs. Therefore, the value function of a PPR can have less local optima than the equivalent FSM, and consequently, the gradient algorithm is more likely to converge to a global optimal solution.

5.1 Introduction

Learning an optimal policy in a partially observable environment is still considered as one of the most difficult challenges in Reinforcement Learning (RL). To achieve this, one must

consider two main issues: (1) mapping histories of interaction with the environment into an internal state representation, (2) finding the optimal decisions associated to each internal state.

Partially Observable Markov Decision Processes (POMDPs, introduced in Section 2.3) provide a rich mathematical framework for studying such problems. In POMDPs, the environment is represented by a finite set of states, and the effects of the actions on the environment are represented by probabilistic transition functions. After every executed action, the agent perceives a numerical reward and a partial, possibly noisy, observation.

Instead of keeping track of ever growing histories, Finite-States Machines (FSMs) provide an efficient graphical model for mapping histories into equivalence classes called *internal-states* (I-states). Many policy gradient methods for POMDPs are based on this family of policies [Meuleau et al., 1999; Baxter and Bartlett, 2000; Peshkin, 2001; Shelton, 2001; Yu, 2005; Yu and Bertsekas, 2008]. An FSM is a probabilistic automaton where the observations are the inputs and the actions are the outputs, with stochastic transitions between the internal states. The main drawback of FSMs is that the internal states do not directly affect the state of the environment. In fact, only the sequence of executed actions determines the returned outcome, and the same sequence of actions may be generated by different sequences of internal states. Thus, considering the sequence of internal states within the learning data slows down the convergence of RL algorithms [Aberdeen and Baxter, 2002].

In this chapter, we introduce a new policy gradient method for finite-horizon POMDPs based on a Predictive Policy Representation (PPR). Aberdeen et al. [2007] have already proposed to use PSRs for reinforcement learning in partially observable domains. However, they used PSRs for learning the dynamics of the environment, and mapped the *belief states* into a distribution over the actions through a linear approximation. In our approach, we use PPRs to directly represent policies without learning a model of the environment. The parameters of PPRs are updated according to the gradient of a *value function*. Our second contribution is a comparison of PPR policies with FSMs. For this purpose, we use the same basic method for learning both FSMs and PPRs, and show that the value function of a PPR policy is potentially simpler than the value function of an FSM policy. This result is due to the fact that the parameters of a PPR are based on only observable data. Finally, we empirically demonstrate the effectiveness of PPR policies through several standard benchmarks.

5.2 Model-free Policies in POMDPs

The goal of a rational agent is to find an action that maximizes its expected long-term reward after a given history (see Section 2.2.4). The function that selects an action a given a history

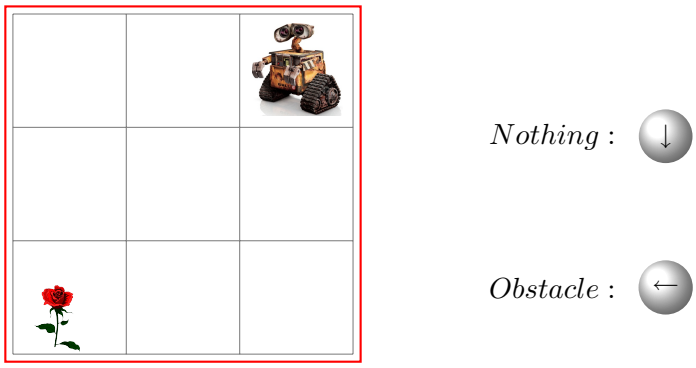
h_t is called policy. A parametric stochastic policy π is a function defined by:

$$\pi(a, h_t, \theta) = Pr(a_{t+1} = a | h_t, \theta)$$

where θ is a vector of real-valued parameters $\theta = (\theta_1, \theta_2, \ldots, \theta_n)$. In this section, we review different approaches for representing policies in POMDPs when the parameters of the POMDP model, namely the transition and the observation functions, are unknown. This includes History-based representations, Finite-State Machines and Neural Networks. We also introduce the Predictive Policy Representations (PPRs) with core histories.

5.2.1 Memoryless policies

A memoryless (*reactive*) policy is function that maps each observation into an action, or a distribution on actions, without considering any of the previous events (see the example showed in Figure 5.1). An algorithm for learning such policies, using a branch and bound heuristic, was proposed by Littman [1996]. Because of their simplicity, memoryless policies have been widely used in policy gradient methods [Baird and Moore, 1999; Baird, 1993], we will return to gradient methods in Section 5.3.1.



(a) A simple POMDP problem. (b) A memoryless policy.

Figure 5.1: A memoryless policy for a simple POMDP problem. The robot can detect the presence or absence of an obstacle after moving into it, the actions correspond to movements to left, right, up, and down.

However, given that the observations are generally stochastic and aliasing (the same observation can be perceived in different states), memoryless policies tend to perform very poorly in practice, except for special domains with simple structures. Singh et al. [1994] presented a simple example of POMDPs where the best memoryless policy can be arbitrary poor compared to the optimal solution. They also showed that the best stochastic memoryless policy can be arbitrary better that the best memoryless deterministic policy. An interesting ranking of four natural classes of memoryless policies was given by Bagnell et al. [2003]. The classes considered in this ranking were: Stationary Deterministic policies (SD), Stationary Stochastic policies (SS), Non-stationary Deterministic policies (ND), and Non-stationary Stochastic policies (NS). A non-stationary memoryless policy is one that is a function of the

time-step. Bagnell et al. [2003] showed that for any finite-state, finite-action POMDP:

$$optimal(\text{SD}) \leqslant optimal(\text{SS}) \leqslant optimal(\text{ND}) = optimal(\text{NS})$$

where the operator *optimal* returns the value of the optimal policy in a class.

The potentially superior performance of non-stationary policies contrasted with stationary policies can be intuitively explained by the time-dependance property of non-stationary policies is a form of memory that provides an additional information about the underlying states. This latter result provides a strong justification for the use of non-stationary policies when it comes to memoryless ones.

5.2.2 History-based policies

This class of models was already presented in Section 2.5.1, we briefly review this method here. A history h_t refers to the sequence of actions and observations $a_1 o_1 \dots a_t o_t$ that were executed and received by the agent since the beginning of the interaction. A typical history-based policy keeps in memory the full history h_t, and uses it as the agent's internal state.

This approach is very useful to overcome the problem of learning policies without using the parameters of the POMDP model, since the histories are completely observable in the training data.

In fact, a POMDP model $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R \rangle$ can always be transformed into an MDP $\langle \mathcal{S}', \mathcal{A}, T', R' \rangle$, where $\mathcal{S}' = \{\mathcal{A} \times \mathcal{O}\}^*$ is an infinite set of states corresponding to histories, T' is a transition function between histories, defined as: $T'(h, a, hao) = Pr(o|h, a)$ and $T'(h, a, ha'o) = 0$ for $a' \neq a$, and R', defined as $R'(h, a) = \sum_{s \in \mathcal{S}} Pr(s|h) R(s, a)$, is a reward function. In this context, one should simply learn the values of actions after each history, or use a reinforcement learning algorithm for MDPs such as Q-Learning (Algorithm 3). This latter approach was recently investigated in [Timmer, 2009].

One problem with history-based models is the large number of histories that one needs to keep in memory: $(|\mathcal{A}||\mathcal{O}|)^t$ different histories for t steps of interacting [Mccallum, 1996], where $|\mathcal{A}|$ and $|\mathcal{O}|$ are respectively the number of actions and observations. This problem can be partly addressed by using compact representations of histories, such as decision trees (Figure 3.1).

However, the principal drawback of history-based models lies in the high variance of value function estimators. This is due, on the one hand, to the definition of the value function as a function of histories and, on the other hand, to the low likelihood of observing the same history several times. A interesting solution to this problem has been given in Utile Distinction Memory (UDM) [McCallum, 1993] and Utile Suffix Memory (USM) [McCallum, 1995] models,

where only the last sequence (suffix) of each history that is necessary to distinguish it from other histories is kept in memory. Two histories are considered as equivalent if all the actions have almost the same value at these histories.

Figure 5.2 shows an example of a USM policy for solving the POMDP problem described in Figure 5.1. The classes of equivalent histories are denoted by Q^i. For instance, all the histories that end with the suffix ($\leftarrow$ obstracle $\leftarrow$ obstracle) are considered as equivalent and regrouped in class Q^4. A statistical test (Kolmogorov-Smirnov for example) is used to verify if there is a significant divergence between two histories, in such case one considers longer suffixes for them, until an acceptable low divergence is achieved.

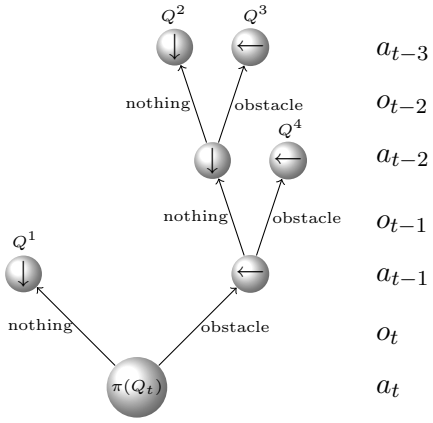


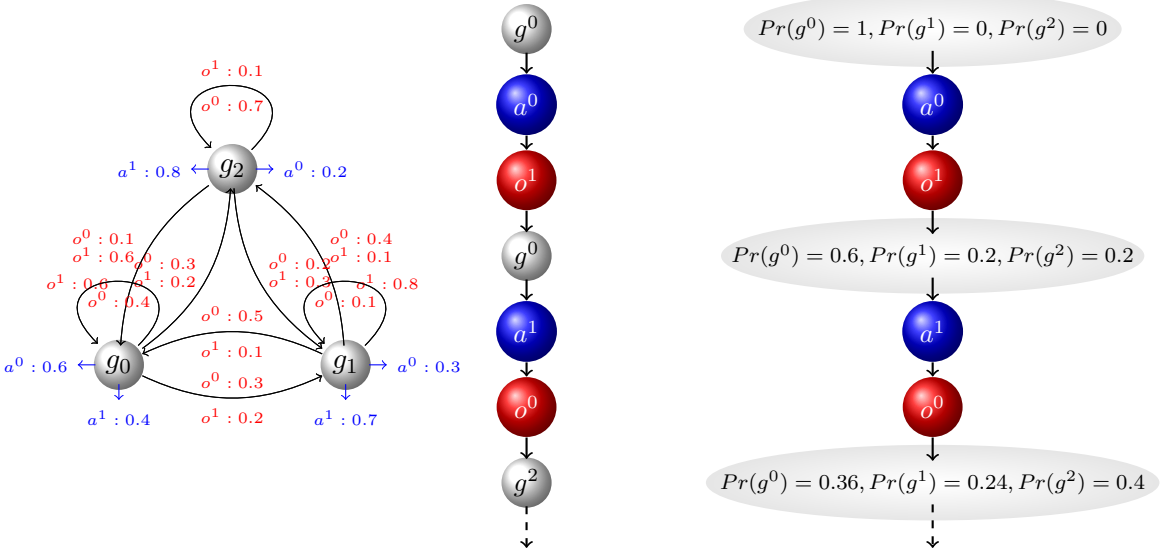
Figure 5.2: A Utile Suffix Memory.

The main problem of history-based models remains the fact that the agent often should remember the full history from the beginning in order to make accurate predictions about the future behavior, and to choose an optimal action. As an example of that, suppose that in the POMDP illustrated in Figure 5.1, the actions of the robot succeed with a probability 0.9, and the robot remains in the same location with a probability 0.1. Since the robot observes an obstacle only after hitting it, then the probability that the history $h_k(a, \text{nothing})^{t-k}$, for $a \in \mathcal{A}$, occurs is non-null for any $t \geqslant k$ and arbitrary history h_k. Therefore, the memory of any history-based policy with bounded length will correspond to $(a, \text{nothing})^n$ after a certain number of time-steps. Consequently, the history h_k, which contains information about the location of the robot, will be erased. This problem was a major factor that severely restrained the applicability of this type of policies. A recent solution to this issue for was proposed by Holmes and Isbell [2006]. The proposed model, known as Prediction Suffix Trees (PSTs), is a generalization of USM where loops between different nodes of the suffix tree were used. This model was trained for predicting the observations of a POMDP, and the empirical results on small domains showed that they outperform other history-based models, such as Causal State Machines (Section 2.5.6). However, this approach is limited to POMDPs with deterministic transition and observation functions.

5.2.3 Finite-State Machines (FSMs)

Finite-State Machines are an efficient graphical model used for representing stochastic policies without memorizing all the histories [Hansen, 1998] (see Section 4.3). An FSM is defined by: a finite set of internal states $\mathcal{G}$; a set of action-selection functions $\mu^{o,a}$, where $\mu^{o,a}(g, \theta)$ is the probability that the agent will execute action a if its internal state is g, its last observation was

o, and the parameters of the policy are θ; a set of transition functions ω^o, where $\omega^o(g, g', \theta)$ is the probability that the agent will select the internal state g if the current internal state is g and the perceived observation is o. An example of FSMs is presented in Figure 5.3. Implicitly, an internal state of a decision-maker regroups all the histories (or the belief states) that have the same optimal policy.



(a) A Finite-State Machine.

(b) A history with internal states.

(c) A history with internal belief states.

Figure 5.3: (a) An example of a Finite-State Machine. (b) A history is a sequence of internal states, actions and observations. (c) The internal states can be replaced by internal belief states.

FSMs are particularly appealing for representing policies when the parameters of the POMDP are unknown, since these latter parameters are not used in the definition of FSMs. The problem of finding a finite-state machine for controlling a POMDP was initially studied by Sondik [1978], where it has been showed that for a certain class of POMDPs, there always exists a deterministic finite state controller equivalent to the optimal policy.

The research on finding FSMs for POMDPs was spawned by the work of Eric Hansen in the late 90s [Hansen, 1998; Hansen and Zilberstein, 1998]. This latter work was focused on finding an optimal FSM for a POMDP with known parameters. In the context of unknown parameters, most of the proposed algorithms for learning FSMs are based on gradient optimization methods [Meuleau et al., 1999; Baxter and Bartlett, 2000; Peshkin, 2001; Shelton, 2001; Yu, 2005; Yu and Bertsekas, 2008]. We return to these methods in Section 5.3.1.

Contrary to the states of a POMDP, the internal states of an FSM are completely observable, thus, a history h_t should be a sequence of actions, observations and sampled internal states, as showed in the example of Figure 5.3. However, the cumulated reward of a given

history depends only on the executed actions and the perceived observations. Moreover, the same sequence of actions and observations can be used to update the parameters (μ and ω) of all the internal states that may have generated that sequence.

To solve this problem, Shelton [2001], Aberdeen and Baxter [2002] proposed a *Rao-Blackwellized* version of FSMs where *internal-belief states* are calculated at each step instead of sampling a single internal state. Rao-Blackwellization technique is known to reduce the variance of Monte-Carlo estimators, the interested reader can find a presentation of these methods in [Casella and Robert, 1996]). The internal belief state is a vector $b_t(.,\theta)$, defined as:

$$b_t(g,\theta) = Pr(g_t = g|h_t,\theta), \forall g \in \mathcal{G}$$

The initial internal belief state b_0 is given in the FSM vector of parameters θ. At each time-step, the agent receives an observation o and executes an action a, the internal belief state is updated by Bayes' Rule as in POMDPs (Equation 2.2):

$$\begin{aligned} b_{t+1}(g,\theta) &= \frac{Pr(g,a|b_t,o)}{Pr(a|b_t,o)} \\ &= \frac{\sum\limits_{g'\in\mathcal{G}} b_t(g',\theta)\omega^o(g',g,\theta)\mu^{o,a}(g,\theta)}{\sum\limits_{g'\in\mathcal{G}}\sum\limits_{g''\in\mathcal{G}} b_t(g',\theta)\omega^o(g',g'',\theta)\mu^{o,a}(g'',\theta)} \end{aligned} \tag{5.1}$$

The sequence of internal belief states is principally used for learning, as we will show in Section 5.4.2, but it can also be used for sampling actions as $Pr(a|b_t) = \sum_{g\in\mathcal{G}} b_t(g,\theta)\mu^{o,a}(g,\theta)$. The likelihood of a sequence of actions and observations $o^1a^1 \dots o^ka^k$ is the same, whether one uses the internal belief state b_t at each time-step for sampling an action, or samples first an internal state g_t according to $\omega^o(g_{t-1},g_t,\theta)$, and then an action according to $\mu^{o,a}(g_t,\theta)$. In fact, $Pr(o^1a^1 \dots o^ka^k)$ can always be written as

$$\begin{aligned} Pr(o^0a^0 \dots o^ka^k) &= \prod_{t=0}^{k} Pr(o^ta^t|o^0a^0 \dots o^{t-1}a^{t-1}) \\ &= \prod_{t=0}^{k}\sum_{g\in\mathcal{G}} Pr(a^t|g)Pr(g^t = g|o^0a^0 \dots o^{t-1}a^{t-1}o^t)Pr(o^t|o^0a^0 \dots o^{t-1}a^{t-1}) \\ &= \prod_{t=0}^{k}\sum_{g\in\mathcal{G}} Pr(a^t|g)b_t(g,\theta)Pr(o^t|o^0a^0 \dots o^{t-1}a^{t-1}) \end{aligned}$$

Using internal belief in the learning trial provides an explicit modeling of the FSM dynamics, and allows a single trial experience to affect every policy that could have produced the action sequence without considering the sequence of internal belief states that actually occurred.

5.2.4 Recurrent Neural Networks

Artificial neural networks have been successfully applied in reinforcement learning since the early developments of this field. A famous example of that is the work of Tesauro [1994] were a neural network was trained for evaluating strategies in the Backgammon game. Neural networks can be used either as a function approximator for representing the value function, or directly as a policy. A neural network is composed of: an input layer, one or many hidden layers, and an output layer. Each layer is a collection of several neurons, which are activated by a function that takes as inputs the values of the neurons of the previous layer. Therefore, a neural network can be seen as a simple graphical representation of a complex function, with parameters corresponding to the weights of the connections between different neurons. Usually, a gradient descent is used for finding the optimal parameters of a network. In the context of reinforcement learning in POMDPs, the input of a neural network at a time-step t would be the history h_t, or a part of it, and the output would correspond to an action to execute. Therefore, the memory of a neural network policy corresponds to its input.

Figure 5.4 shows an example of a policy represented by a simple recurrent neural network. The actions a_t correspond to the 4 movements: $\leftarrow$,$\rightarrow$,$\uparrow$, and $\downarrow$. The observations are indicated by o_t. There are two internal states of this network, g_1 and g_2, that take values in $\{0, 1\}$. The input is a window of the history, and the output is an action to execute. The nodes of this network can be interconnected using any activation function, such as a sigmoid for example.

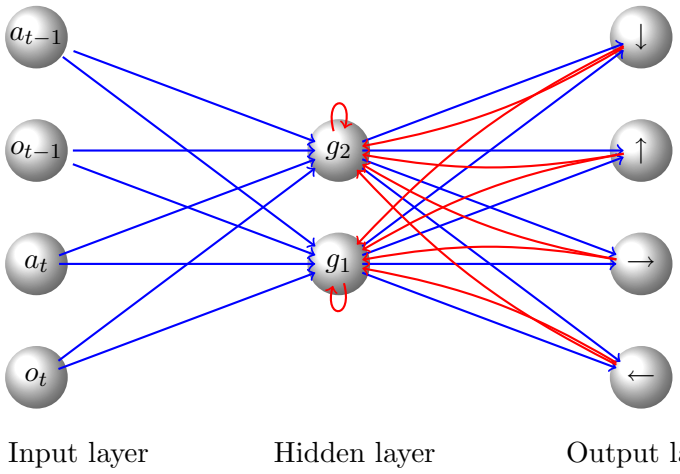


Figure 5.4: An example of a policy represented by a simple recurrent neural network.

An interesting variant of this model is *recurrent* neural networks, where the activation function of a given neuron can take as input the value of any neuron in the network. Since the values of the neurons at time-step t are fed back to the network at time-step $t + 1$, the neurons of the hidden layer can be seen as an internal state (or belief state). Consequently, recurrent networks are potentially useful for solving POMDPs.

Long Short-Term Memory (LSTM) is a well-known architecture of recurrent neural net-

works used for reinforcement learning in POMDPs [Bakker, 2001]. In this architecture, part of the network is trained to approximate a Q-value function, which is a function of an internal state. The internal state is represented by a set of hidden neurons, in addition to special neurons called *memory cells*, which are used to store previous values of hidden neurons. Access to the memory cells is controlled through a special layer of neurons called *gates*. An extension of this framework was proposed by Gomez and Schmidhuber [2005], where the topology of the network as well as the weights of the neurons were learned using an evolutionary approach. Finally, the same architecture was recently used for learning efficient policies in POMDPs with continuous state spaces [Wierstra and Schmidhuber, 2007; Sehnke et al., 2010; Wierstra et al., 2009].

Recurrent neural networks are a very practical approach for solving POMDPs, they can deal with multidimensional and continuous actions and observations. However, despite their impressive results, neural networks lack strong theoretical guarantees related to their applicability in reinforcement learning. In fact, it is not yet completely understood how neural networks represent internal states, and what the limitation of such approach are. Also, it is not possible to relate the performance of a policy learned by a neural network to the optimal one.

5.2.5 Predictive Policy Representation (PPRs)

In Section 4.3, we showed that PSRs can be used to represent policies instead of the environment dynamics by switching the roles of actions and observations, and we called this model Predictive Policy Representation (PPRs). In this section, we introduce PPRs with core histories.

A test q is defined as an ordered sequence of (*observation, action*) couples, i.e.

$$q = o^1 a^1 \dots o^k a^k$$

Given the vector θ of policy parameters (defined latter), the probability of q starting after h_t is redefined as:

$$Pr(q^a|h_t, q^o, \theta) \stackrel{def}{=} Pr(a_{t+1} = a^1, \dots, a_{t+k} = a^k | h_t, o_{t+1} = o^1, \dots, o_{t+k} = o^k, \theta) \quad (5.2)$$

where q^a denotes the actions of q and q^o denotes its observations.

As for the environment dynamics (Section 2.4), the probability $Pr(q^a|h_t, q^o, \theta)$ of any test q starting after a history h_t is given by a linear combination of the probabilities of the same test q starting after different core histories $h \in \mathcal{H}$. The internal belief state b_t corresponds to the weights of these core histories in $Pr(q^a|h_t, q^o, \theta)$. In particular, the probability of executing action a at time t after observing o is given by:

$$Pr(a|h_t, o) = \sum_{h \in \mathcal{H}} b_t(h, \theta) Pr(a|h, o, \theta) = b_t^T m_{oa} \quad (5.3)$$

The core histories of a policy are independent from the core histories of the controlled environment. We will use $\mathcal{H}$ only to indicate the policy core histories, since no model of the environment is used in this approach.

After receiving an observation o and executing an action a, the internal belief state $b_t(.,\theta)$ is updated by using the following rule:

$$b_{t+1}(h,\theta) = \frac{\sum_{h' \in \mathcal{H}} b_t(h',\theta) b_{h'oa}(h,\theta) m_{oa}(h',\theta)}{\sum_{h' \in \mathcal{H}} b_t(h',\theta) m_{oa}(h',\theta)} \tag{5.4}$$

This latter rule is directly derived from Bayes' Rule. In fact, we can verify that the probability of any sequence q after receiving an observation o and executing an action a can be calculated using the updated belief b_{t+1}:

$$\begin{aligned}
Pr(q^o|h_t,a,o,q^a) &= \frac{Pr(aq^a|h_t,o,q^o)}{Pr(a|h_t,o)} \\
&= \frac{\sum_{h\in\mathcal{H}} b_t(h,\theta) Pr(aq^a|h,o,q^o)}{\sum_{h\in\mathcal{H}} b_t(h,\theta) Pr(a|h,o)} \\
&= \frac{\sum_{h\in\mathcal{H}} b_t(h,\theta) Pr(a|h,o) Pr(q^a|h,o,a,q^o)}{\sum_{h\in\mathcal{H}} b_t(h,\theta) Pr(a|h,o)} \\
&= \frac{\sum_{h\in\mathcal{H}} b_t(h,\theta) m_{oa}(h) \sum_{h'\in\mathcal{H}} b_{hoa}(h',\theta) m_q(h')}{\sum_{h\in\mathcal{H}} b_t(h,\theta) m_{oa}(h)} \\
&= \sum_{h\in\mathcal{H}} b_{t+1}(h,\theta) m_q(h)
\end{aligned}$$

A PPR policy is defined by: the set of core histories $\mathcal{H}$, and the vectors m_{oa} and b_{hoa}, $\forall o \in \mathcal{O}, a \in \mathcal{A}, h \in \mathcal{H}$. The vector of parameters θ of a PPR corresponds to m_{oa} and b_{hoa}.

An interesting property of belief states in PPRs is that the entries of a belief state can take any real value, but they always sum to 1, i.e. $\forall t \in \{0,\ldots,H\} : \sum_{h\in\mathcal{H}} b_t(h,\theta) = 1$. Indeed, for any observation o, we have:

$$\begin{aligned}
\sum_{h\in\mathcal{H}} b_t(h,\theta) &= \sum_{h\in\mathcal{H}} b_t(h,\theta) \underbrace{\sum_{a\in\mathcal{A}} m_{oa}(h,\theta)}_{1} \text{ (from Equation 5.3)} \\
&= \sum_{a\in\mathcal{A}} \sum_{h\in\mathcal{H}} b_t(h,\theta) m_{oa}(h,\theta) \\
&= \sum_{a\in\mathcal{A}} Pr(a|h_t,o,\theta) = 1
\end{aligned} \tag{5.5}$$

The initial history $h_0 = \varepsilon$ is always a core history, thus $b_0(\varepsilon,\theta) = 1$ and $\forall h \in \mathcal{H} - \{\varepsilon\}$:

$b_0(h, \theta) = 0$. We will show in Section 5.4 how to both discover the set of core histories $\mathcal{H}$, and how to learn the parameters m_{oa} and b_{hoa}.

Figure 5.5 shows an example of a predictive policy Representation with core histories. This policy is represented by two core histories: the initial history ε, and the history $h = o^0 \downarrow$. The probability of executing a given action after a given core history and observation is known (part of the model). For instance, $Pr(\leftarrow |h, o^0) = 0.1$. These probabilities can be represented by a soft-max function. Therefore, a PPR proceeds as a stochastic decision tree as long as the current history h_t is a core history. When $h_t \notin \mathcal{H}$, the probability of executing an action a after h_t and observation o is a linear combination of the probabilities of executing the same action a after different core histories and same observation o. For instance $Pr(\leftarrow |h_t, o^0) = b_t(\varepsilon)Pr(\leftarrow |\varepsilon, o^0) + b_t(h)Pr(\leftarrow |h, o^0)$, where b_t is the belief state corresponding to h_t.

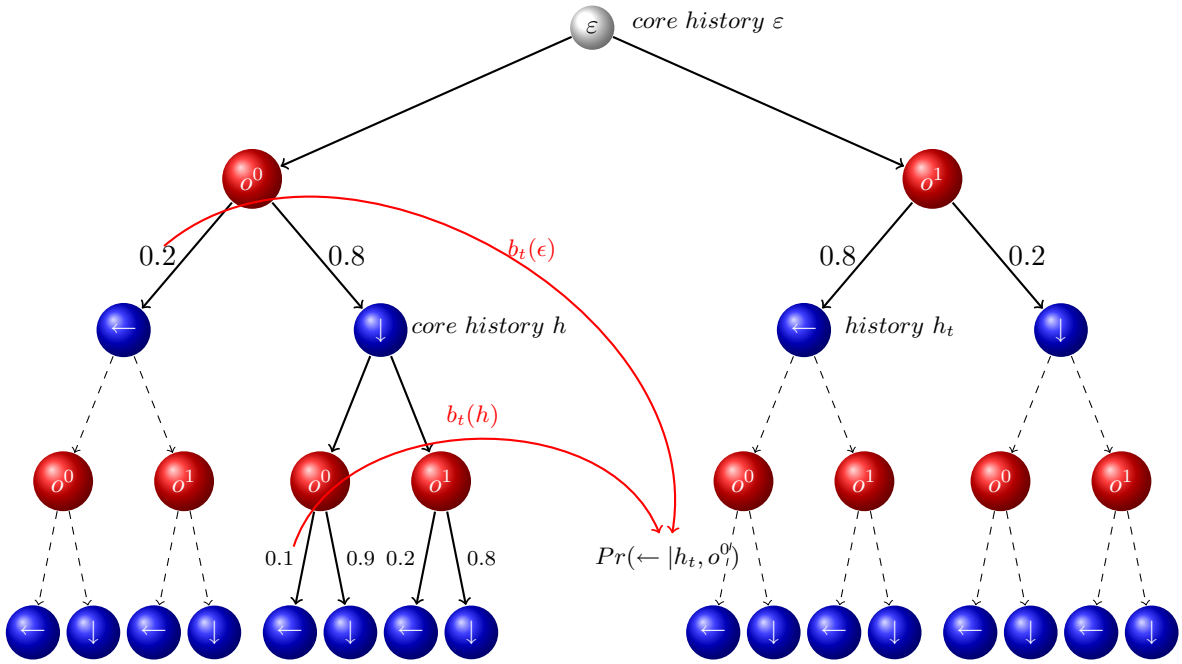


Figure 5.5: A Predictive Policy Representation with Core Histories.

5.3 Policy Gradient Methods

5.3.1 Overview

Given a parametric model of a policy, there are two major families of reinforcement learning algorithms that can be used to find the optimal parameters of a policy. In the first family, a value function is used to estimate the expected long-term reward of each action after every possible history h_t. The parameters of the policy are adjusted so that in each state, the

actions with the highest estimated long-term reward will be executed more frequently. In the second family, a locally optimal policy is gradually learned by searching in the space of its parameters. The parameters are updated according to the immediate reward after executing each action, or according to the cumulated reward at the end of one or many trials.

Policy Gradient methods are a type of policy search where a parametric policy is optimized by performing a gradient ascent (or descent) on its estimated value function [Williams, 1992]. One advantage of these methods is that they can be generally described independently of the model used for representing the policy. In fact, given a policy π^k, with parameters $\theta^k \in \mathbb{R}^d$, a policy gradient algorithm consists in first estimating the gradient $\nabla_\theta V|_{\theta=\theta^k}$, where V is the value function of the policy, defined as (assuming the initial belief state does not change):

$$V(\theta^k) = \mathbb{E}_{s_t,a_t}\left[\sum_{t=0}^{H} \gamma^t R(s_t, a_t)|\theta^k, T, Z\right] \tag{5.6}$$

where T and Z are respectively the transition and observation functions (see the definition of a POMDP in Section 2.3).

Then, the parameters θ^k are updated as:

$$\theta^{k+1} = \theta^k + \eta^k \nabla_\theta V|_{\theta=\theta_k} \tag{5.7}$$

where $\eta_k \in \mathbb{R}$ is a step-size. The same operation is repeated again with parameters θ^{k+1}, the algorithms stops when $\nabla_\theta V|_{\theta=\theta_k} = 0$, in which case $V(\theta^k)$ is a local maximum.

As a result of their flexibility, policy gradient methods can be used with policy representations that are more adapted to the specific task that one wants to perform, incorporating previous domain knowledge about the task. Also, policy gradient can be either model-free or model-based, depending on the problem. As a consequence of that, most robotic applications of reinforcement learning are based on this type of methods [Peters and Schaal, 2006].

In the remaining of this section, we briefly review some policy gradient techniques, based on a survey of Peters and Schaal [2008]. These methods are regrouped into three categories: *Finite-difference methods*, *Likelihood ratio methods*, and *Natural policy gradient methods*.

5.3.2 Finite-difference methods

The main problem in policy gradient methods is the estimation of the gradient $\nabla_\theta V|_{\theta=\theta^k}$ when the parameters of the system, namely the transition function T and the observation function Z (see Section 2.3), are unknown. Finite-difference methods were among the first solutions proposed to solve this problem [Glynn, 1990], they emerged from the Monte Carlo and Stochastic Simulation community (see [Glynn, 1987; L'Ecuyer, 1991] for an overview of these methods).

Given a policy with parameters $\theta^k = (\theta^k_1, \theta^k_2, \ldots, \theta^k_d)$, the gradient $\nabla_\theta V|_{\theta=\theta^k}$ is estimated by first estimating the value $V(\theta^k)$ by sampling N trials, as follows:

$$\hat{V}(\theta^k) = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{H}\gamma^t r_{i,t}$$

where $r_{i,t}$ is the reward received at time-step t of trial i.
Then, a small variation $\Delta\theta^k_i = (0, \ldots, \delta\theta^k_i, \ldots, 0)$ is applied on each parameter θ^k_i, and the value $V(\theta^k + \Delta\theta^k_i)$ of the policy with perturbed parameters $\theta^k + \Delta\theta^k_i$ is estimated in a similar way as for $V(\theta^k)$. Finally, the gradient $\nabla_\theta V|_{\theta=\theta^k}$ is estimated as:

$$\hat{\nabla}_\theta V|_{\theta=\theta^k} = \left(\frac{\hat{V}(\theta^k + \Delta\theta^k_1) - \hat{V}(\theta^k)}{\delta\theta^k_1}, \ldots, \frac{\hat{V}(\theta^k + \Delta\theta^k_i) - \hat{V}(\theta^k)}{\delta\theta^k_i}, \ldots, \frac{\hat{V}(\theta^k + \Delta\theta^k_d) - \hat{V}(\theta^k)}{\delta\theta^k_d}\right)$$

Notice from the approximation $\hat{\nabla}_{\theta_i} V|_{\theta=\theta^k} = \frac{1}{\delta\theta^k_i}[\hat{V}(\theta^k + \Delta\theta^k_i) - \hat{V}(\theta^k)]$ that $\hat{\nabla}_{\theta_i} V|_{\theta=\theta^k}$ is a linear function of $\frac{1}{\delta\theta^i_d}$ with an estimated factor of $\hat{V}(\theta^k + \Delta\theta^k_i) - \hat{V}(\theta^k)$. Therefore, this latter factor can be more precisely approximated by applying several variations on each parameter θ^k_i, and using a linear regression to find $\hat{\nabla}_{\theta_i} V|_{\theta=\theta^k}$.

Despite their appealing simplicity, finite-difference methods can hardly be used for a real-world application due to their hazardous policy variations that can result in arbitrary costly actions.

5.3.3 Likelihood ratio methods

Likelihood ratio methods, also known as REINFORCE algorithms [Williams, 1992], regroup the majority of policy gradient methods, such as GAPS [Peshkin, 2001] and GPOMDP [Baxter and Bartlett, 2000]. Let us denote by h_t a history $a_1 o_1 \ldots a_t o_t$ a time-step t, and by $R(a|h_t)$ the expected reward of executing action a after the history h_t, i.e.

$$R(a|h_t) = \sum_{s\in S} b_t(s) R(s,a) \tag{5.8}$$

The gradient $\nabla_\theta V|_{\theta=\theta^k}$ is estimated as:

$$\begin{aligned}
\nabla_\theta V(\theta) &= \nabla_\theta \mathbb{E}_{s_t,a_t}\left[\sum_{t=0}^{H}\gamma^t R(s_t,a_t)|\theta, T, Z\right] \text{ (from Equation 5.6)} \\
&= \nabla_\theta \sum_{t=0}^{H-1}\sum_{h_t\in\{\mathcal{A}\times\mathcal{O}\}^t}\sum_{a\in\mathcal{A}}\gamma^t R(a|h_t) Pr(h_t a|\theta) \\
&= \sum_{t=0}^{H-1}\sum_{h_t\in\{\mathcal{A}\times\mathcal{O}\}^t}\sum_{a\in\mathcal{A}}\gamma^t R(a|h_t)\nabla_\theta Pr(h_t a|\theta)
\end{aligned}$$

$$\begin{aligned}
\nabla_\theta V(\theta) &= \sum_{t=0}^{H-1} \sum_{h_t \in \{\mathcal{A}\times\mathcal{O}\}^t} \sum_{a\in\mathcal{A}} \gamma^t R(a|h_t) Pr(h_t a|\theta) \frac{\nabla_\theta Pr(h_t a|\theta)}{Pr(h_t a|\theta)} \\
&= \sum_{t=0}^{H-1} \sum_{h_t \in \{\mathcal{A}\times\mathcal{O}\}^t} \sum_{a\in\mathcal{A}} \gamma^t R(a|h_t) Pr(h_t a|\theta) \nabla_\theta \log Pr(h_t a|\theta) \\
&= \nabla_\theta \mathbb{E}_{h_t, a_{t+1}} \left[\sum_{t=0}^{H-1} \gamma^t R(a_{t+1}|h_t) \nabla_\theta \log Pr(h_t a_{t+1}|\theta) | \theta, T, Z \right] \qquad (5.9)
\end{aligned}$$

Therefore, the gradient $\nabla_\theta V|_{\theta=\theta^k}$ can be estimated by sampling several trajectories using the policy parameters θ^k, and then calculating the expected discounted value of the product $R(a_{t+1}|h_t)\nabla_\theta \log Pr(h_t a_{t+1}|\theta)|_{\theta=\theta^k}$. Interestingly, the gradient of the log-likelihood $\log Pr(h_t a_{t+1}|\theta)$ can be exactly calculated since it does not involve any unknown parameters, in fact:

$$\begin{aligned}
\nabla_\theta \log Pr(h_t a_{t+1}|\theta) &= \nabla_\theta \log \prod_{i=1}^{t} Pr(o_i|a_1 o_1 \dots a_i) Pr(a_{i+1}|a_1 o_1 \dots a_{i-1} o_i, \theta) \\
&= \sum_{i=1}^{t} \left[\underbrace{\nabla_\theta \log Pr(o_i|a_1 o_1 \dots a_i)}_{0} + \nabla_\theta \log Pr(a_{i+1}|a_1 o_1 \dots a_{i-1} o_i, \theta) \right] \\
&= \sum_{i=1}^{t} \nabla_\theta \log Pr(a_{i+1}|a_1 o_1 \dots a_{i-1} o_i, \theta)
\end{aligned}$$

This latter term, $\nabla_\theta \log Pr(a_{i+1}|a_1 o_1 \dots a_{i-1} o_i, \theta)$, depends on the model used for representing the policy. For instance, if the policy is represented by an FSM (presented in Section 5.2.3), then $\nabla_\theta \log Pr(a_{i+1}|a_1 o_1 \dots a_{i-1} o_i, \theta)$ is a function of $\mu^{o,a}$ and ω^o only.

Moreover, Williams [1992] showed that the variance of the estimator of $\nabla_\theta V(\theta)$ (Equation 5.9) can be reduced by adding a baseline $b \in \mathbb{R}$ to the immediate rewards as:

$$\nabla_\theta V(\theta) = \nabla_\theta \mathbb{E}_{h_t, a_{t+1}} \left[\sum_{t=0}^{H-1} \gamma^t (R(a_{t+1}|h_t) - b) \nabla_\theta \log Pr(h_t a_{t+1}|\theta) | \theta, T, Z \right]$$

In fact, this latter formula is derived from the fact that the different histories h_t for a given t are disjoint events:

$$\begin{aligned}
& \forall t \in \{0, \dots, H-1\} : \sum_{h_t \in \{\mathcal{A}\times\mathcal{O}\}^t} \sum_{a\in\mathcal{A}} Pr(h_t a|\theta) = 1 \\
\Rightarrow \quad & \forall t \in \{0, \dots, H-1\} : \nabla_\theta \sum_{h_t \in \{\mathcal{A}\times\mathcal{O}\}^t} \sum_{a\in\mathcal{A}} Pr(h_t a|\theta) = 0 \\
\Rightarrow \quad & \nabla_\theta \sum_{t=0}^{H-1} \sum_{h_t \in \{\mathcal{A}\times\mathcal{O}\}^t} \sum_{a\in\mathcal{A}} \gamma^t Pr(h_t a|\theta) = 0
\end{aligned}$$

Therefore:

$$\forall b \in \mathbb{R} : \nabla_\theta \sum_{t=0}^{H-1} \sum_{h_t \in \{\mathcal{A}\times\mathcal{O}\}^t} \sum_{a \in \mathcal{A}} \gamma^t b Pr(h_t a|\theta) = 0$$

The baseline b can be chosen arbitrary, however, Peters and Schaal [2006] showed how to derive a baseline that minimizes the variance of the gradient estimator.

Compared to finite-differences, likelihood ratio methods are appealing for two principal reasons. First, the variance of the gradient estimator is lower, since the known parameters of the policy are used in estimating the gradient. Second, the gradient is estimated without applying any perturbation to the parameters of the policy. In fact, these latter parameters are updated in greedy way (Equation 5.7), improving the quality of the policy after each update.

5.3.4 Natural policy gradient methods

A major inconvenient of the so called *vanilla* gradient method presented in the previous subsections is that its performance largely depends on the choice of the step-size η_k used in the updates $\theta^{k+1} = \theta^k + \eta_k \nabla_\theta V|_{\theta=\theta^k}$. The step-size η_k is used for two reasons. The first one is controlling the change in the policy in order to make it smooth and to avoid premature convergence. The second one is rescaling the gradient $\nabla_\theta V|_{\theta=\theta^k}$ into the space of parameters θ. In fact, the update rule used in the gradient methods is inconsistent since the left hand side has units of θ, and the right hand side has units of $1/\theta$ [Kakade, 2002].

Natural gradient methods, proposed by Amari [1998] and introduced to reinforcement learning by Kakade [2002], solve these two problems by using a step-size related to the space of the value function $V(\theta)$ instead of the parameters θ. In other terms, the distance between $V(\theta^{k+1})$ and $V(\theta^k)$ is kept constant after each updated of the policy parameters by $\theta^{k+1} = \theta^k + \tilde{\nabla}_\theta V|_{\theta=\theta^k}$, where $\tilde{\nabla}_\theta V$ is the natural gradient given by $\tilde{\nabla}_\theta V = F_\theta^{-1} \nabla_\theta$, and F_θ is the *Fisher information matrix*, defined as:

$$F_\theta(i,j) = \mathbb{E}_{h_t, a_{t+1}} \left[\sum_{t=0}^{H-1} \gamma^t \frac{\partial \log Pr(h_t a_{t+1}|\theta)}{\partial \theta_i} \frac{\partial \log Pr(h_t a_{t+1}|\theta)}{\partial \theta_j} | \theta, T, Z \right]$$

Note that this latter matrix can be estimated from the sampled trajectories, since the function returning $Pr(h_t a_{t+1}|\theta)$ is known. The complete derivation of the natural gradient can be found in [Peters et al., 2005].

The principal advantage of natural policy gradient methods is that they are independent of the frame used for representing the policy parameters. For instance, the same policy gradient algorithm can be used for parameters that are probabilities as for real-valued parameters. Also, the path followed by the parameter updates is more direct, therefore, the natural gradient method converges faster and avoids premature convergence [Peters and Schaal, 2008].

5.4 A General Actor-Critic Approach

5.4.1 Overview

Actor-critic algorithms [Sutton et al., 2000; Peters and Schaal, 2006; Aberdeen et al., 2007] combine the advantages of both value-function and policy search methods. The actor part maintains a parametric policy that is gradually improved according to the evaluation provided by the critic part. The critic learns a value function where the variables correspond to the parameters of the policy.

In this section, we describe a general method where the critic part learns unbiased estimates of immediate rewards which are used to calculate the gradient of the value function with respect to the policy parameters. We show how history-action Q-values are eliminated from the expression of the gradient and replaced by immediate rewards. Finally, we present a likelihood ratio expression of the gradient for general parameterized policies. Since the main objective of this work is to compare predictive policy representations to other types of policies, namely FSMs, we will not consider natural gradients in the current study.

The actor is a stochastic policy with a vector of parameters θ. The critic is a function that returns the gradient of the value function $V(h_0, \theta)$, which is the total, expected, discounted reward when executing a policy parameterized by θ after a starting void history $h_0 = \varepsilon$.

$$V(h_0, \theta) = \sum_{a \in \mathcal{A}} Pr(a|h_0, \theta) Q(h_0, a, \theta)$$

where the history-action Q-values are given by:

$$Q(h_t, a, \theta) = R(a|h_t) + \gamma \sum_{o \in \mathcal{O}} \sum_{a' \in \mathcal{A}} Pr(oa'|h_t a, \theta) Q(h_t ao, a', \theta) \tag{5.10}$$

5.4.2 Gradient Expression

The following derivation is an adaptation to POMDPs of the proof of the policy gradient theorem for MDPs, proposed by Sutton et al. [2000]. We indicate by h_t^a the actions of a history h_t and by h_t^o its observations.

$$\begin{aligned} \frac{\partial V(h_0, \theta)}{\partial \theta_i} &= \frac{\partial}{\partial \theta_i} \sum_{a \in \mathcal{A}} Pr(a|h_0, \theta) Q(h_0, a, \theta) \\ &= \sum_{a \in \mathcal{A}} \frac{\partial Pr(a|h_0, \theta)}{\partial \theta_i} Q(h_0, a, \theta) + Pr(a|h_0, \theta) \frac{\partial Q(h_0, a, \theta)}{\partial \theta_i} \end{aligned}$$

By substituting $Q(h_0, a, \theta)$ with Equation (5.10), we find:

$$\begin{aligned}
\frac{\partial V(h_0,\theta)}{\partial \theta_i} &= \sum_{a\in\mathcal{A}} \frac{\partial Pr(a|h_0,\theta)}{\partial \theta_i} Q(h_0,a,\theta) + Pr(a|h_0,\theta)\frac{\partial Q(h_0,a,\theta)}{\partial \theta_i} \\
&= \sum_{a\in\mathcal{A}} \frac{\partial Pr(a|h_0,\theta)}{\partial \theta_i} Q(h_0,a,\theta) + Pr(a|h_0,\theta)\gamma\frac{\partial}{\partial \theta_i}\left[\sum_{o\in\mathcal{O}}\sum_{a'\in\mathcal{A}} Pr(oa'|h_0a,\theta)Q(h_0ao,a',\theta)\right] \\
&= \sum_{a\in\mathcal{A}} \frac{\partial Pr(a|h_0,\theta)}{\partial \theta_i} Q(h_0,a,\theta) + \gamma\sum_{o\in\mathcal{O}} Pr(ao|h_0,\theta)\sum_{a'\in\mathcal{A}}\frac{\partial}{\partial \theta_i}\left[Pr(a'|h_0ao,\theta)Q(h_0ao,a',\theta)\right] \\
&= \sum_{a\in\mathcal{A}} \frac{\partial Pr(a|h_0,\theta)}{\partial \theta_i} Q(h_0,a,\theta) + \gamma\sum_{o\in\mathcal{O}} Pr(ao|h_0,\theta)\sum_{a'\in\mathcal{A}}\frac{\partial Pr(a'|h_0ao,\theta)}{\partial \theta_i} Q(h_0ao,a',\theta) \\
&\qquad + \gamma\sum_{o\in\mathcal{O}} Pr(ao|h_0,\theta)\sum_{a'\in\mathcal{A}} Pr(a'|h_0ao,\theta)\frac{\partial Q(h_0ao,a',\theta)}{\partial \theta_i} \\
&= \sum_{a\in\mathcal{A}} Q(h_0,a,\theta)\frac{\partial Pr(a|h_0,\theta)}{\partial \theta_i} \\
&\quad + \sum_{a\in\mathcal{A}}\sum_{o\in\mathcal{O}}\sum_{a'\in\mathcal{A}} \gamma Pr(ao|h_0,\theta)Q(h_0ao,a',\theta)\frac{\partial Pr(a'|h_0ao,\theta)}{\partial \theta_i} \\
&\quad + \sum_{a\in\mathcal{A}}\sum_{o\in\mathcal{O}}\sum_{a'\in\mathcal{A}} \gamma Pr(aoa'|h_0,\theta)\frac{\partial Q(h_0ao,a',\theta)}{\partial \theta_i}
\end{aligned}$$

Repeating the substitution for H time-steps yields to:

$$\frac{\partial V(h_0,\theta)}{\partial \theta_i} = \sum_{t=0}^{H-1}\sum_{h_t\in\{\mathcal{A}\times\mathcal{O}\}^t}\sum_{a\in\mathcal{A}} \gamma^t Pr(h_t|\theta)Q(h_t,a,\theta)\frac{\partial Pr(a|h_t,\theta)}{\partial \theta_i} \tag{5.11}$$

By applying Bayes' Rule $Pr(a|h_t,\theta) = \frac{Pr(h_t^a a|h_t^o,\theta)}{Pr(h_t^a|h_t^o,\theta)}$, we find:

$$\begin{aligned}
\frac{\partial Pr(a|h_t,\theta)}{\partial \theta_i} &= \frac{1}{Pr(h_t^a|h_t^o,\theta)}\frac{\partial Pr(h_t^a a|h_t^o,\theta)}{\partial \theta_i} - \frac{\partial Pr(h_t^a|h_t^o,\theta)}{\partial \theta_i}\frac{Pr(h_t^a a|h_t^o,\theta)}{Pr(h_t^a|h_t^o,\theta)^2} \\
&= \frac{1}{Pr(h_t^a|h_t^o,\theta)}\frac{\partial Pr(h_t^a a|h_t^o,\theta)}{\partial \theta_i} - \frac{\partial Pr(h_t^a|h_t^o,\theta)}{\partial \theta_i}\frac{Pr(a|h_t,\theta)}{Pr(h_t^a|h_t^o,\theta)}
\end{aligned} \tag{5.12}$$

By replacing this latter formula within Equation (5.11), and adequately rearranging the terms, we find:

$$\begin{aligned}
\frac{\partial V(h_0,\theta)}{\partial \theta_i} &= \sum_{t=0}^{H-1}\sum_{h_t\in\{\mathcal{A}\times\mathcal{O}\}^t}\sum_{a\in\mathcal{A}} \gamma^t\Bigg[Pr(h_t|\theta)Q(h_t,a,\theta)\frac{1}{Pr(h_t^a|h_t^o,\theta)}\frac{\partial Pr(h_t^a a|h_t^o,\theta)}{\partial \theta_i} \\
&\qquad - Pr(h_t|\theta)Q(h_t,a,\theta)\frac{\partial Pr(h_t^a|h_t^o,\theta)}{\partial \theta_i}\frac{Pr(a|h_t,\theta)}{Pr(h_t^a|h_t^o,\theta)}\Bigg] \\
&= \sum_{t=0}^{H-1}\sum_{h_t\in\{\mathcal{A}\times\mathcal{O}\}^t}\sum_{a\in\mathcal{A}} \gamma^t\Bigg[\frac{\partial Pr(h_t^a a|h_t^o,\theta)}{\partial \theta_i}\frac{Pr(h_t a|\theta)}{Pr(h_t^a a|h_t^o,\theta)}Q(h_t,a,\theta) \\
&\qquad - Pr(a|h_t,\theta)\frac{\partial Pr(h_t^a|h_t^o,\theta)}{\partial \theta_i}\frac{Pr(h_t|\theta)}{Pr(h_t^a|h_t^o,\theta)}Q(h_t,a,\theta)\Bigg]
\end{aligned}$$

$$\frac{\partial V(h_0,\theta)}{\partial \theta_i} = \sum_{t=0}^{H-1} \sum_{h_t \in \{\mathcal{A}\times\mathcal{O}\}^t} \sum_{a\in\mathcal{A}} \gamma^t \frac{\partial Pr(h_t^a a|h_t^o,\theta)}{\partial\theta_i} \frac{Pr(h_t a|\theta)}{Pr(h_t^a a|h_t^o,\theta)} \underbrace{[Q(h_t,a,\theta) - \gamma \sum_{o\in\mathcal{O}} \sum_{a'\in\mathcal{A}} Pr(oa'|h_t a) Q(h_t ao, a',\theta)]}_{R(a|h_t)}$$

In this latter derivation, we rearranged the terms of time-step t with those of time-step $t+1$. Thus:

$$\frac{\partial V(h_0,\theta)}{\partial \theta_i} = \sum_{t=0}^{H-1} \sum_{h_t \in \{\mathcal{A}\times\mathcal{O}\}^t} \sum_{a\in\mathcal{A}} \gamma^t \underbrace{\frac{\partial Pr(h_t^a a|h_t^o,\theta)}{\partial\theta_i}}_{policy} \overbrace{Pr(h_t^o|h_t^a) R(a|h_t)}^{environment} \tag{5.13}$$

Note that the term $Pr(h_t^o|h_t^a)R(a|h_t)$ is independent of θ. It can be learned by a simple Monte Carlo method with Importance Sampling, using a look-up table:

$$\begin{cases} \hat{Pr}(h_t^o|h_t^a) = \frac{1}{Pr(h_t^a|h_t^o,\theta)} \frac{\#h_t}{N} \\ \\ \hat{R}(a|h_t) = \frac{1}{N} \sum_{i=1}^{N} r_{i,t} \delta_{h_t,h_{i,t}} \delta_{a,a_{i,t+1}} \end{cases}$$

where N is the number of trials, $a_{i,t}$ and $r_{i,t}$ are respectively the action and the reward observed at time t in trial i, $h_{i,t}$ is the history at time t in trial i, and δ is the Kronecker delta. A more efficient method for reusing samples is given in [Hachiya et al., 2009].

In the following two subsections, we will show how to calculate the other term, $\frac{\partial Pr(h_t^a a|h_t^o,\theta)}{\partial\theta_i}$, depending on the model of the policy.

5.4.3 Gradient Estimation for Finite-State Machines

If we use an FSM (Section 5.2.3) to represent the policy, then:

$$Pr(h_t^a|h_t^o,\theta) = b_0^T(.,\theta) M_\theta^{o_1 a_1} \dots M_\theta^{o_t a_t} e \tag{5.14}$$

where

$$\begin{cases} M_\theta^{o_j a_j}(g,g') \stackrel{def}{=} \omega^{o_j}(g,g',\theta) \mu^{o_j,a_j}(g',\theta) \\ e^T = (1,1,\dots,1) \end{cases}$$

and $b_0^T(.,\theta)$ is the transpose of the initial internal belief state, corresponding to the initial probability distribution on the internal states.
Thus:

$$\frac{\partial Pr(h_t^a|h_t^o,\theta)}{\partial\theta_i} = \sum_{j=1}^{t} \sum_{g\in\mathcal{G}} \sum_{g'\in\mathcal{G}} [\alpha_{j-1}(g,\theta) \beta_{j+1}^t(g',\theta) \frac{\partial(\omega^{o_j}(g,g',\theta)\mu^{o_j,a_j}(g',\theta))}{\partial\theta_i}] + \sum_{g\in\mathcal{G}} \beta_0^t(g,\theta) \frac{\partial b_0(g,\theta)}{\partial\theta_i}$$

where

$$\begin{cases} \alpha_0(g,\theta) = b_0(h,\theta) \\ \alpha_i(g,\theta) = b_0^T(.,\theta) M_\theta^{o_1 a_1} \dots M_\theta^{o_i a_i}(.,g) \\ \beta_i^t(g,\theta) = M_\theta^{o_i a_i}(g,.) M_\theta^{o_{i+1} a_{i+1}} \dots M_\theta^{o_t a_t} e \\ \beta_{t+1}^t(g,\theta) = 1 \end{cases}$$

This latter expression of the gradient corresponds to the same one proposed in [Shelton, 2001].

5.4.4 Gradient Estimation for Predictive Policy Representations

If we use a PPR to represent the policy, then:

$$Pr(h_t^a | h_t^o, \theta) = b_0^T(.,\theta) M_\theta^{o_1 a_1} \dots M_\theta^{o_t a_t} e \tag{5.15}$$

where

$$\begin{cases} M_\theta^{o_j a_j}(h,h') \stackrel{def}{=} m_{o_j a_j}(h,\theta) b_{h o_j a_j}(h',\theta) \\ e^T = (1,1,\dots,1) \end{cases}$$

Therefore:

$$\frac{\partial Pr(h_t^a | h_t^o, \theta)}{\partial \theta_i} = \sum_{j=1}^{t} \sum_{h \in \mathcal{H}} \sum_{h' \in \mathcal{H}} [\alpha_{j-1}(h,\theta) \beta_{j+1}^t(h',\theta) \frac{\partial (m_{o_j a_j}(h,\theta) b_{h o_j a_j}(h',\theta))}{\partial \theta_i}]$$

where

$$\begin{cases} \alpha_0(h,\theta) = b_0(h,\theta) \\ \alpha_i(h,\theta) = b_0^T(.,\theta) M_\theta^{o_1 a_1} \dots M_\theta^{o_i a_i}(.,h) \\ \beta_i^t(h,\theta) = M_\theta^{o_i a_i}(h,.) M_\theta^{o_{i+1} a_{i+1}} \dots M_\theta^{o_t a_t} e \\ \beta_{t+1}^t(h,\theta) = 1 \end{cases}$$

5.4.5 Updating the Parameters of the Policy

The gradient calculated by the critic (Equation 5.13) is used to update the parameters θ_i:

$$\theta_i \leftarrow \theta_i + \eta \frac{\partial V(h_0,\theta)}{\partial \theta_i} \tag{5.16}$$

where $\eta \in [0,1]$ is the gradient step-size.

Policy gradient methods are proved to converge to a local optimum. The probability of reaching a global optimum depends on the starting point, the step-size, and the complexity (or the shape) of the value function. In the following section, we show that the value function is polynomial, with a smaller degree when the policy is represented by a PPR.

5.5 The Degree of The Value Function

The functions ω^o and $\mu^{o,a}$ return distributions of probabilities, they are usually represented as softmax functions. The analysis of the value function is more difficult with this representation, since it involves exponentials and fractions. For the purpose of comparing to PPRs, and without loss of generality, we consider each $\omega^o(g, g')$ and each $\mu^{o,a}(g)$ as a different parameter, so $\omega^o(g, g', \theta) \stackrel{def}{=} \theta_{g,o,g'}$ and $\mu^{o,a}(g, \theta) \stackrel{def}{=} \theta_{o,g,a}$, we also consider $b_0(g, \theta) \stackrel{def}{=} \theta_g$.

For PPRs, the vectors b_{hoa} are not stochastic, therefore each $b_{hoa}(h')$ is a parameter θ_i, so $b_{hoa}(h', \theta) \stackrel{def}{=} \theta_{hoa,h'}$. As for FSMs, we define $m_{oa}(h, \theta) \stackrel{def}{=} \theta_{oa,h}$.

Notice now that the function returning $Pr(h_t^a|h_t^o, \theta)$ (Equations 5.14 and 5.15) is a multivariate polynomial of degree less than or equal to $2t$. The variables of this polynomial correspond to the parameters $\theta_{ho_ia_i,h'}, \theta_{o_ja_j,h}$ for a PPR, and to $\theta_{g,o_j,g'}, \theta_{o_j,g,a_j}, \theta_g$ for an FSM. Unless a specific structure of the FSM is provided *a priori*, the graph of the FSM is generally completely connected, i.e. $\forall a, o, g, g' : \omega^o(g, g', \theta)\mu^{o,a}(g', \theta) > 0$. Thus, we have $\alpha_j(g, \theta) > 0$ and $\beta_j^t(g', \theta) > 0$ for every internal state g and step $i \leqslant t$, the degree of the polynomial $Pr(h_t^a|h_t^o, \theta)$ for the FSM is then equal to $2t$:

$$Pr(h_t^a|h_t^o, \theta) = \underbrace{b_0^T(., \theta)}_{\text{degree } 0} \underbrace{M_\theta^{o_1a_1} \dots M_\theta^{o_ta_t}}_{\text{degree } 2t} \underbrace{e}_{\text{degree } 0}$$

To reduce the degree of the value function, [Aberdeen and Baxter, 2002] proposed reducing the outdegree of the FSM graph while augmenting the number of internal states. However, the convergence can be delayed in that case, given that more parameters should be learned.

The main advantage of PPRs comes from the fact that, contrary to internal states, the core histories are contained within the sequence of actions and observations, and no transition probabilities are used to calculate the probability of a core history sequence. Namely, when a prefix sequence $o_1a_1 \dots o_ia_i$ of the history $h_t = o_1a_1 \dots o_ia_i \dots o_ta_t$ corresponds to a core history h_i, then the term $b_0^T(., \theta)M_\theta^{o_1a_1} \dots M_\theta^{o_ia_i}$ in Equation (5.15), can be replaced by a vector $\alpha_i(., \theta)$ where $\alpha_i(h_i, \theta) = Pr(h_i^a|h_i^o, \theta)$ and $\alpha_i(h, \theta) = 0$ for $h \in \mathcal{H} - \{h_i\}$ (α_i is the "unnormalized" belief at time i). The probability of the sequence of actions contained in the core history h_i is given by:

$$Pr(h_i^a|h_i^o, \theta) = Pr(a_1|h_0, o_1, \theta) \dots Pr(a_i|h_{i-1}, o_i, \theta)$$

From the construction of PPRs, as we will show in Section 5.7 (see also Figure 5.5), we know that all the prefixes of a core history are also core histories, therefore:

$$Pr(h_i^a|h_i^o, \theta) = \theta_{o_1a_1,h_0}\theta_{o_2a_2,h_1} \dots \theta_{o_ia_i,h_{i-1}}$$

This latter term, $Pr(h_i^a|h_i^o, \theta)$, is a polynomial of degree i. Hence, the degree of the

polynomial $Pr(h_t^a|h_t^o,\theta)$ (Equation 5.15) is at most equal to $2t-i$:

$$Pr(h_t^a|h_t^o,\theta) = \underbrace{b_0^T(.,\theta)}_{\text{degree } 0}\underbrace{M_\theta^{o_1a_1}\dots M_\theta^{o_ia_i}}_{\text{degree } i}\underbrace{M_\theta^{o_{i+1}a_{i+1}}\dots M_\theta^{o_ta_t}}_{\text{degree } 2(t-i)}\underbrace{e}_{\text{degree } 0}$$

Consequently, the degree of the value function of a PPR is equal to $2t-i$, where i is the length of the shortest core history h_i that is not a suffix of another core history. Therefore, the degree of the value function polynomial is generally smaller when the policy is represented by a PSR than when it is represented by an FSM. This gain is due to the simple fact that PPRs proceed as decision trees as long as the encountered history is a core history, then they start acting like Finite-State Machines for the remaining time-steps, as depicted in the example of Figure 5.5.

An interesting advantage of optimizing a function with a smaller degree is that the number of local optima can also be smaller, as we will see in the following toy example.

5.5.1 Example

Figure 5.6 shows a simple POMDP problem with 3 deterministic actions: $\downarrow$ (*Down*), $\leftarrow$ (*Left*) and $\rightarrow$ (*Right*), and two deterministic observations o_1 (*upper three states*) and o_2 (*lower three states*). There are also two final states marked with positive rewards 1 and 1/2.

To control the robot, a parametric FSM with three internal states is used, g_1 is the initial internal state, g_2 and g_3 are final internal states. Although one should use a different parameter for each $\omega^o(g,g')$ and each $\mu^{o,a}(g)$, we will use only two parameters, since representing graphically functions with more than 2 variables is unfeasible. The first parameter θ_1 corresponds to the probability of going from state g_1 to state g_3 after observing o_2, while $1-\theta_1$ is the probability of going to g_2. We assume that the robot already knows that it should first move down, thus $Pr(\downarrow|o_1,g_1)=1$. The only possible observation after having moved down is o_2. The second parameter θ_2 corresponds to the probability of executing action $\leftarrow$ in state g_3 after having observed o_2, and $1-\theta_2$ is the probability of executing action $\downarrow$ in g_3 after having observed o_2 too. The same parameter θ_2 is used for the actions in state g_2. The optimal FSM is given by $\theta_1=\theta_2=0$. The value function of this FSM for horizon 2 is given by:

$$V^{FSM}(h_0,\theta) = \frac{1}{2}\theta_1\theta_2 + (1-\theta_1)(1-\theta_2)$$

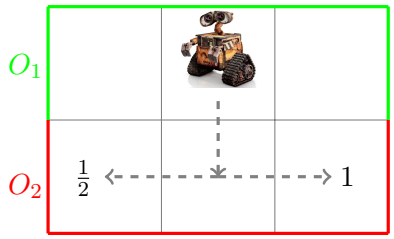
it has one global maximum of value 1 and one local maximum of value $\frac{1}{2}$.

The equivalent PPR can be represented by one core history: ε, and three parameters: ${\theta_1}' = m_{o_2\leftarrow}(\varepsilon)$ and ${\theta_2}' = m_{o_2\rightarrow}(\varepsilon)$ and ${\theta_3}' = m_{o_2\downarrow}(\varepsilon)$. The value function of this policy is given by:

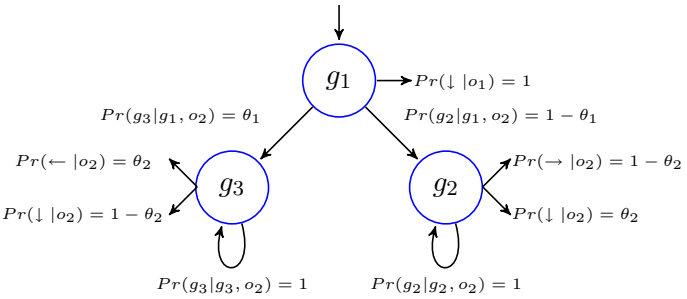
$$V^{PPR}(h_0,\theta') = \frac{1}{2}{\theta_1}' + {\theta_2}'$$

it has only a global maximum within the simplex of valid parameters, i.e. $\theta_1' + \theta_2' \leqslant 1$.

This gain is due to the fact that the history $h_1 = o_1 \downarrow$ is equivalent to ϵ (all the histories are equivalent to ϵ when $\mathcal{H} = \{\epsilon\}$, in which case the policy is simply a reactive one), thus $b_{h_1}(\varepsilon) = 1$ (degree 0) and the probability of action $\leftarrow$ after h_1 and o_2, for instance, is $Pr(\leftarrow | h_1, o_2) = {\theta_1}'$ (degree 1). While for the FSM, we have $b_{h_1}(g_3) = \theta_1$, $b_{h_1}(g_2) = 1 - \theta_1$ and $Pr(\leftarrow | h_1, o_2) = \theta_1\theta_2$ (degree 2).



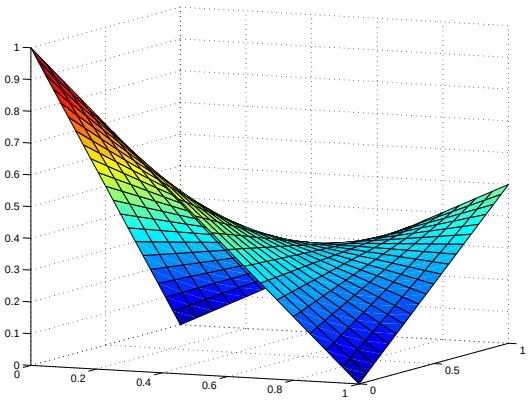
(a)



(b)

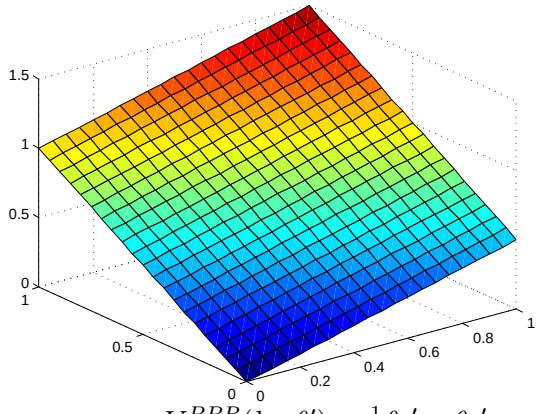
$$h_0 \stackrel{def}{=} \varepsilon,\ \mathcal{H} = \{\varepsilon\},\ b_{o_1\downarrow}(\varepsilon) = 1, m_{o_2\leftarrow}(\varepsilon) = Pr(\leftarrow | o_2) \stackrel{def}{=} {\theta_1}'$$
$$m^{o_2\rightarrow}(\varepsilon) = Pr(\rightarrow | o_2) \stackrel{def}{=} {\theta_2}', m^{o_2\downarrow}(\varepsilon) = Pr(\downarrow | o_2) \stackrel{def}{=} {\theta_3}'$$

(c)



$$V^{FSM}(h_0, \theta) = \tfrac{1}{2}\theta_1\theta_2 + (1 - \theta_1)(1 - \theta_2)$$

(d)



$$V^{PPR}(h_0, \theta') = \tfrac{1}{2}{\theta_1}' + {\theta_2}'$$
$$\text{subject to } {\theta_1}' + {\theta_2}' \leqslant 1$$

(e)

Figure 5.6: The value functions of an FSM is a polynomial of degree 2, while that of its equivalent PPR is a polynomial of degree 1. (a) A toy POMDP problem. (b) A Finite-State Machine (FSM) for controlling this POMDP. (c) An equivalent Predictive State Representation (PPR). (d) The value function of the FSM. (e) The value function of the PPR.

5.6 Constraining The PPR Belief States

The belief state of a PPR with core histories is not a vector of probabilities, it is rather subject to a set of constraints that should be satisfied at any time-step t:

$$\begin{cases} C_t^1 : \forall o \in \mathcal{O}, \forall a \in \mathcal{A} : \sum_{h \in \mathcal{H}} b_t(h, \theta) m_{oa}(h, \theta) \geqslant 0; \\ C_t^2 : \forall o \in \mathcal{O} : \sum_{a \in \mathcal{A}} \sum_{h \in \mathcal{H}} b_t(h, \theta) m_{oa}(h, \theta) = 1; \end{cases}$$

The number of reachable belief states b_t is exponential with respect to the horizon H, while only the vectors m_{oa} and b_{hoa} appear in the parameters of the policy. Thus, we should define a finite set of constraints on m_{oa} and b_{hoa} to ensure that $\{C_t^1, C_t^2\}$ will be satisfied after an arbitrary number of bayesian updates. The existence of such constraints depends on the convexity of the bayesian update function inside the convex hull of valid belief states, and up to our knowledge, this problem is still open [Wolfe, 2010b]. Nevertheless, we will define a set of weaker constraints on m_{oa} and b_{hoa} that can keep the beliefs b_t within the hull of valid weights for short horizons:

$$\begin{cases} C_0^1 : \forall o \in \mathcal{O}, \forall a \in \mathcal{A}, \forall h \in \mathcal{H} : m_{oa}(h, \theta) \geqslant 0; \\ C_0^2 : \forall o \in \mathcal{O}, \forall h \in \mathcal{H} : \sum_{a \in \mathcal{A}} m_{oa}(h, \theta) = 1; \\ C_0^3 : \forall o \in \mathcal{O}, \forall a \in \mathcal{A}, \forall h \in \mathcal{H} : \sum_{h' \in \mathcal{H}} b_{hoa}(h', \theta) = 1; \end{cases}$$

Constraints C_0^1 and C_0^2 can be satisfied by using a soft-max function for each core history h and each observation o. Constraint C_0^3 is justified by Equation 5.5. To satisfy it, we project the gradient $\frac{\partial V(h_0,\theta)}{\partial \theta_{hoa,h'}}$ onto the hyperplane defined by C_0^3. The projected values are given by:

$$\frac{\partial V(h_0, \theta)}{\partial \theta_{hoa,h'}} \leftarrow \frac{1}{2}\left[\frac{\partial V(h_0, \theta)}{\partial \theta_{hoa,h'}} - \frac{1}{|\mathcal{H}|} \sum_{h'' \in \mathcal{H}} \frac{\partial V(h_0, \theta)}{\partial \theta_{hoa,h''}}\right] \tag{5.17}$$

This latter projection is found by solving a system of linear equations, defining the closest point in a plane to another point in the space.

5.7 Discovery of Core Histories

Initially, the set of core histories $\mathcal{H}$ contains only the empty history ε, the vectors m_{oa} are initialized to uniform distributions over actions a for each observation o, and all the histories are considered as equivalent to ε, i.e. $\theta_{oa,\varepsilon} \overset{def}{=} b_{oa}(\varepsilon, \theta) = 1$ (because of the constraint C_0^3). As the parameters θ are updated, new core histories are iteratively discovered and added to $\mathcal{H}$. A history hoa (which is an extension of a core history $h \in \mathcal{H}$) is considered as a core history if there is at least a test q such that $Pr(q^a|hoa, q^o)$ cannot be written as a linear combination of the probabilities $Pr(q^a|h', q^o), h' \in \mathcal{H}$. Since this criteria cannot be verified for every possible test q, we will rather use a set of heuristic measures as an indicator of the predictability of

the history hoa. The first one is based on the distance d_{hoa} between the vector θ_{hoa}, which is updated by using the gradient calculated in Equation (5.13), and the corrected vector θ'_{hoa}, which is updated by using the projected gradient of Equation (5.17) (i.e. d_{hoa} is the correction made on θ_{hoa} to make it satisfy the constraint C_0^3):

$$d_{hoa}^2 = \sum_{h' \in \mathcal{H}} (\theta_{hoa,h'} - \theta'_{hoa,h'})^2$$

The second measure is the entropy e_{ho} of the distribution on actions a conditioned on the history h and the observation o:

$$e_{ho} = -\sum_{a \in \mathcal{A}} Pr(a|h,o) \ln Pr(a|h,o)$$

Finally, a history hoa cannot be added to $\mathcal{H}$ if it has not been tested enough. Thus, our third measure is $\hat{Pr}(hoa|\varepsilon) = \frac{\#hoa}{N}$, where N is the number of trials.
A history hoa is considered as a new core history if and only if:

$$(d_{hoa} \geqslant \epsilon_d) \wedge (e_{ho} \geqslant \epsilon_e) \wedge (\hat{Pr}(hoa|\varepsilon) \geqslant \epsilon_p)$$

where ϵ_d,ϵ_e and ϵ_p are predefined thresholds.

The choice of these thresholds has a great impact on the number of core histories, smaller thresholds lead to a higher number of core histories. This problem, which is close to the problem of choosing the number of internal states in FSMs, has been empirically studied in [Makino and Takagi, 2008].

The probabilities $m_{oa}(h', \theta)$ of a new core history h' are initialized to a uniform distribution, whereas all the extensions of h' are considered as equivalent to h', i.e. $b_{h'o'a'}(h', \theta) = 1$ and $b_{h'o'a'}(h'', \theta) = 0$ for $h'' \in \mathcal{H} - \{h'\}$.

5.8 Experiments

We tested the policy gradient approach described in Section 5.4 on small standard problems taken from [Cassandra, 1998], using PPRs and FSMs as models of the policies. We used softmax functions for representing: (i) the distributions on actions in both PPRs (with the vectors m_{oa}) and FSMs (with the functions μ^{o_j,a_j}) and, (ii) the distributions on the next internal states in FSMs (with the functions ω^{o_j}). We used the same temperature τ for these functions, τ is initialized to 0.1 for 4×4 maze problem, to 1 for Cheese maze and Shuttle problems, and to 10 for Network problem. For all these problems, τ is decreasing by a constant factor of 0.999 at every time-step.

The number of internal states $|\mathcal{G}|$ is the only hyper-parameter for FSMs, it is chosen by a cross-validation for each problem. We found that the best value of $|\mathcal{G}|$ is 6 for 4×4 maze,

3 for Cheese maze and Shuttle problems, and 8 for Network problem. The parameters of the transition functions ω^{o_j}, used by the softmax function, were randomly initialized to values between 0 and 1. The parameters of the action-selection functions μ^{o_j,a_j} were initialized to 0 (a uniform distribution). The functions μ^{o_j,a_j} and ω^{o_j} cannot be all initialized to uniform distributions, this results in uniform beliefs and the initial parameters will be a local optimum [Aberdeen and Baxter, 2002].

The hyper-parameters of PPR policies are the thresholds ϵ_p,ϵ_e, and ϵ_d (see Section 5.7). ϵ_p is set to 0.1 for all the problems, while ϵ_d is set to 0.1 for 4×4 maze, to 6 for Network, and to 0 for Cheese maze and Shuttle. ϵ_e is set to 0 for 4×4 maze and Network, and to 1.5 for Cheese maze and Shuttle. Notice that these thresholds are used to control the size of $\mathcal{H}$: using smaller thresholds leads to more core histories and vice versa. They are found by a cross-validation, with slightly more trials than for $|\mathcal{G}|$ in FSMs, since they are continuous parameters. Finally, we used the same constant step-size η for both PPRs and FSMs, η is set to 1 for 4×4 and Cheese mazes (problems with small rewards), and to 0.1 for Network and Shuttle (problems with high rewards). This scaling problem came from the fact that we did not use natural gradients.

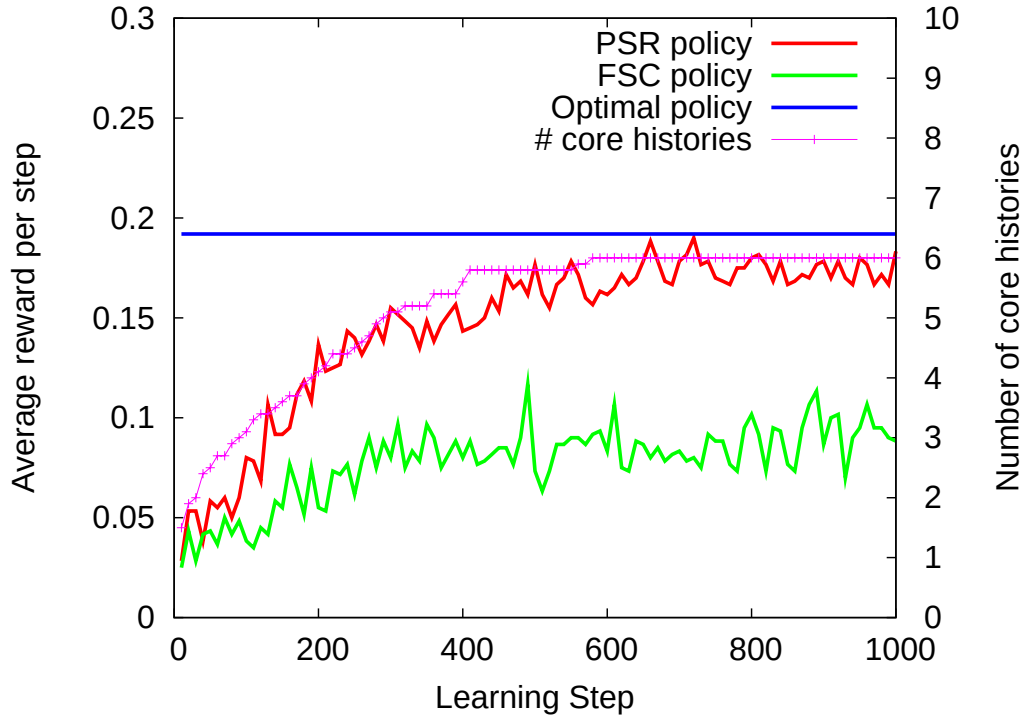


Figure 5.7: Empirical results on learning PPRs (referred as PSR policies) and FSMs using the Policy Gradient approach in the 4x4 maze problem.

Figures 5.7, 5.8, 5.9, and 5.10 show the average reward per step of FSMs and PPRs (referred PSR policies), and the average number of discovered core histories per step for PPRs. The results are averaged over 10 independent runs. For all these problems, the Policy Gradient algorithm converged to locally optimal values for both models of policies. However, we notice that the final value is slightly higher when we use a Predictive Policy Representation, which was expected from the theoretical analysis of Section 5.5. The better performance of PPRs is more pronounced in problems with a smaller number of histories ($|\mathcal{A}||\mathcal{O}| = 8$ in 4×4 maze and Network, 15 in Shuttle, and 28 in Cheese maze). Indeed, discovering the accurate set of core histories becomes more difficult when the branching factor $|\mathcal{A}||\mathcal{O}|$ is higher. Finally, we notice that the learning algorithm for PPRs converged rapidly in Network problem, this is explained by the fact that all the core histories were discovered within the first 30 trials.

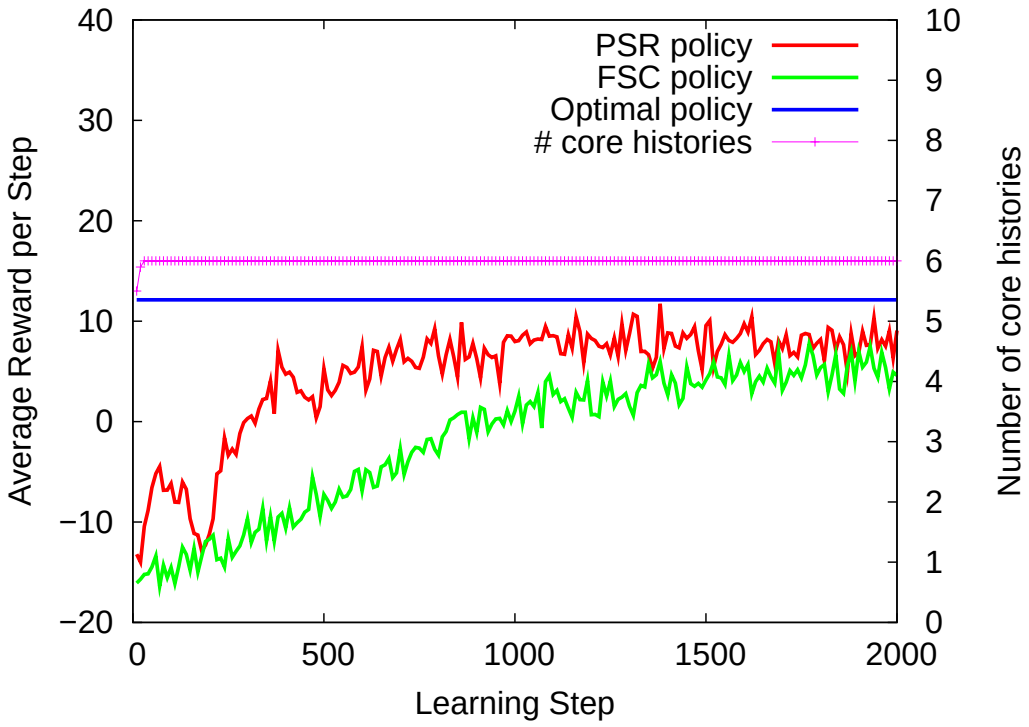


Figure 5.8: Empirical results on learning PPRs (referred as PSR policies) and FSMs using the Policy Gradient approach in the Network problem.

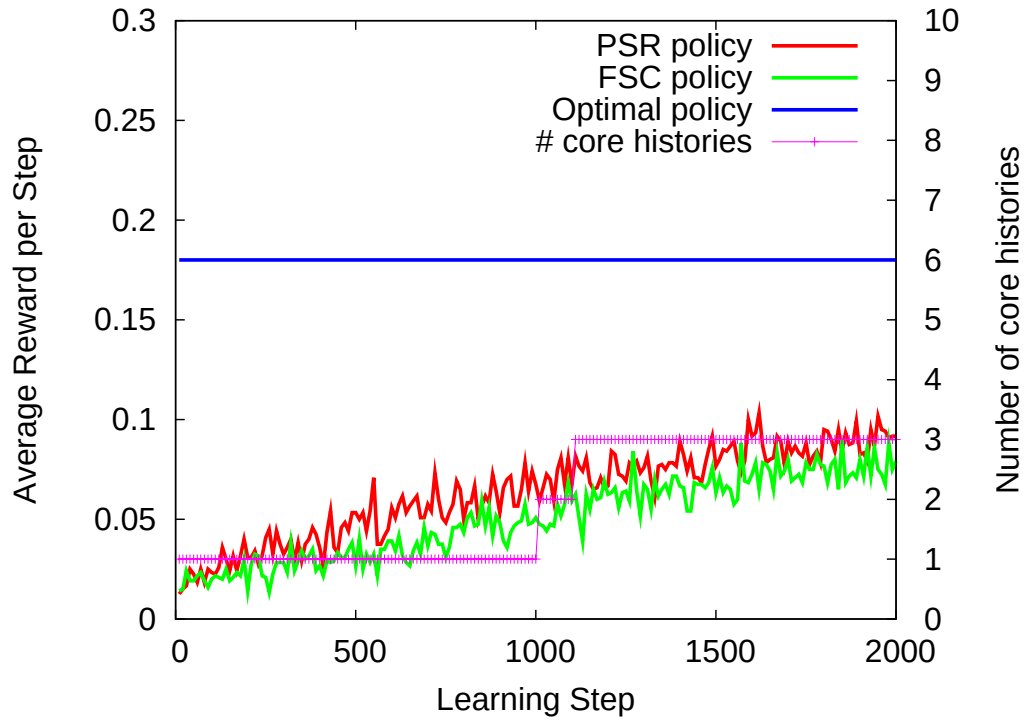


Figure 5.9: Empirical results on learning PPRs (referred as PSR policies) and FSMs using the Policy Gradient approach in the Cheese maze problem.

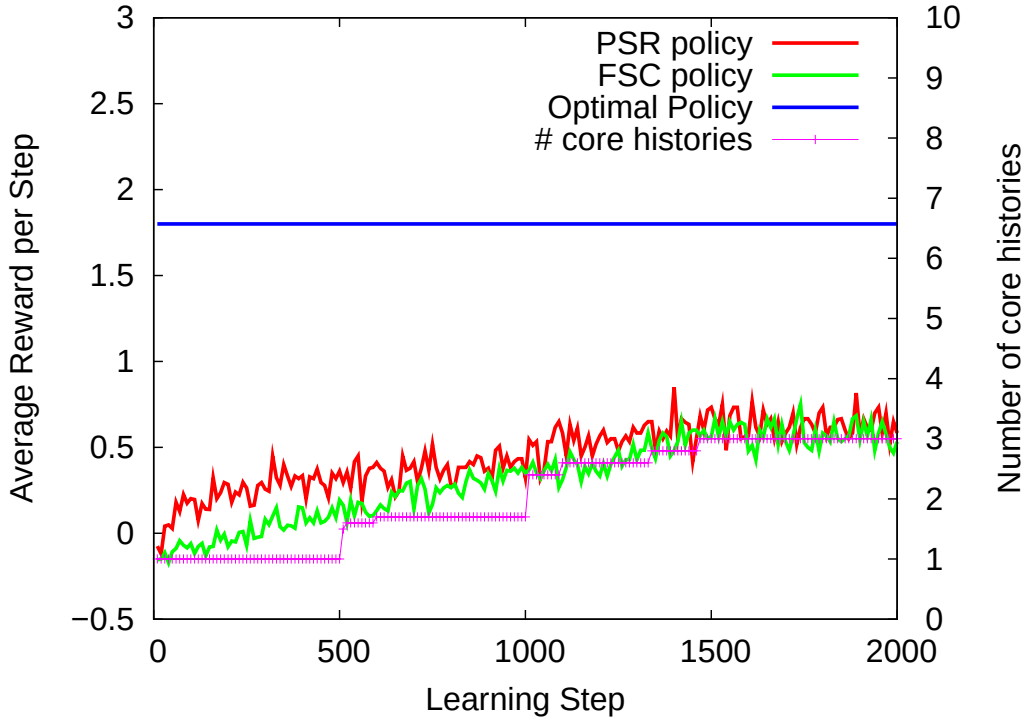


Figure 5.10: Empirical results on learning PPRs (referred as PSR policies) and FSMs using the Policy Gradient approach in the Shuttle problem.

5.9 Conclusion

We have shown that Predictive Policy Representations (PPRs) are alternative to Finite-State Machines in policy gradient methods for POMDPs. Internal states of PPRs are based on observable sequences of interacting with the environment, called core histories. Two main advantages result from this property. The first one is related to problems with short horizons, where the degree of the value function polynomial is reduced by the length of the shortest core history that is not a prefix of another core history. The second advantage is the possibility of discovering new core histories, based on the predictability of the extensions of previous core histories. We used different heuristics, based on the entropy of the actions distribution and the correction of the gradient, as an indicator of the predictability, but more sophisticated techniques should be investigated [Makino and Takagi, 2008].

However, it is unclear how PPRs will perform in problems with infinite, or even just long, horizon where the value function is defined only on the stationary belief state and the core histories cannot be observed. In fact, the preliminary experiments that we performed on larger scale problems showed that PPRs had the same difficulty as FSMs in reaching a satisfactory solution, with an additional effort for cross-validating the continuous hyper-parameters.

This problem can be treated by using a mechanism for detecting *reset points* [James, 2005]. Particularly, one can combine core histories and core tests, and use the first ones to detect the reset points for the second. The other issue related to infinite-horizons is the instability of the belief states. We defined some constraints that can keep the belief within the hull of valid parameters for short horizons, but finding a general efficient solution remains an open problem [Wolfe, 2010b].

Chapter 6

Imitation Learning with Predictive Representations

Learning by imitation has shown to be a powerful paradigm for automated learning in autonomous robots. This paradigm is often preferred to reinforcement learning, presented in Chapter 5, for several reasons. Amongst these reasons, the most important one is the small number of the training data needed for learning how to perform a given task. Moreover, the hazardous exploratory actions that are important in reinforcement learning are no longer necessary when the agent learns under the supervision of an expert. Finally, many real-world tasks, such as those related to humanoid robotics, are so complicated that their corresponding reward functions cannot even be handcrafted. In this chapter, we tackle two problems occurring in imitation learning. The first one is related to the partial observability of states, while the second one is related to the scarcity of the expert's demonstrations in large domains. We present a general model of learning by imitation for stochastic and partially observable systems. The model, which was already introduced in Chapter 4, is a Predictive Policy Representation (PPR) whose goal is to represent the policies of an expert without any reference to states. Then, we return to the problem of learning from an expert in a completely observable stochastic environment where the demonstrations of the expert are scarce and cover only a small part of a large state space. To solve this problem, we propose to use a transfer method, known as soft homomorphism, in order to generalize the expert's policy to unvisited regions of the state space. Finally, we will show that this same problem can be solved by bootstrapping the expert's policy, and using predictive reward representations.

6.1 Introduction

Modern robots are designed to perform complicated planning and control tasks, such as manipulating objects, navigating in outdoor environments, and driving in urban settings. Unfor-

tunately, manually programming these tasks is almost infeasible in practice due to the high number of related states. Markov Decision Processes (MDPs) provide efficient mathematical tools to handle such tasks with a little help from an expert. The expert's help consists in simply specifying a reward function. Based on this reward function, the robot can either directly find an optimal policy if the dynamics of the environment related to the task is known, or learn that policy by trying different actions, and estimating the value of each action based on the received rewards. The first approach is called planning and was presented in Chapters 2 and 3, the second one is called reinforcement learning and was presented in Chapters 2 and 5.

The assumption of knowing the dynamics of the environment cannot be easily verified in practice. For instance, a typical task of a personnel assistive robot always involves an interaction with a human user. Due to its high complexity, the human behavior can hardly be modeled using a finite number of parameters. Reinforcement learning solves this problem, but at the cost of executing random and hazardous actions for long periods before finding the optimal ones. Moreover, in many practical problems, even specifying a reward function is not easy. In fact, it is often easier to demonstrate examples of a desired behavior than to define a reward function [Ng and Russell, 2000].

This latter approach, known as imitation learning (a.k.a. apprenticeship learning) [Atkeson and Schaal, 1997; Billard, 2002], is a well-known technique that provides a fast and efficient way of acquiring new skills without the need for extensive experimentations. It is also justified somewhat by the fact that much of biological learning is done by imitation. In assistive robotics, where the human-machine interaction is omnipresent, imitative learning can be a powerful learning paradigm. In the imitation process, a *learner* agent observes the actions of a *expert* agent, and tries to find the relation between the actions and the different encountered situations. One can generally distinguish between direct and undirect apprenticeship approaches [Ratliff et al., 2009]. In direct methods, the robot learns a function that maps states features into actions by using a supervised learning technique [Atkeson and Schaal, 1997]. The best known example of a system built on this paradigm is ALVINN [Pomerleau, 1989], where a neural network was trained to learn a mapping between a road image and a vehicle steering action. Despite the remarkable success of the ALVINN system and others, direct methods suffer from a serious drawback: they can learn only reactive policies, where the optimal action of a state depends only on its features, regardless of the future states of the system.

To overcome this drawback, Ng and Russell [2000] introduced a new approach of undirect apprenticeship learning known as Inverse Reinforcement Learning (IRL). The aim of IRL is to recover a reward function under which the expert's policy is optimal, rather than to directly mimic the actions of the expert. The learned reward function is then used to find an optimal policy. Contrary to direct methods, IRL takes into account the fact that the different states of the system are related by transition and value functions. Consequently, the expert's actions can be predicted in states that are different from those appearing in the demonstration.

Unfortunately, as already pointed out by [Abbeel and Ng, 2004], recovering a reward function is an ill-posed problem. In fact, the expert's policy can be optimal under an infinite number of reward functions. Most of IRL algorithms that have been proposed were attempting to solve this problem by adding a regularization cost. These algorithms rely on the assumption that the reward function is a linear combination of states features, and the frequency of encountering each feature can be accurately estimated from the demonstration. However, feature frequencies might be poorly estimated when the number of demonstrations is small, as we will show in our experiments in this chapter. Moreover, in the absence of a complete expert's policy, the frequencies of the features that belong to only unvisited states cannot even be estimated.

In the next two sections, we present a brief overview of direct and undirect approaches to imitation learning. The remaining sections are dedicated to describe the solutions that we propose for solving two important problems in imitation learning: learning in a partially observable environment, and generalizing the learned policy to unseen states.

6.2 Direct Imitation Learning

Direct Imitation Learning, also known as *mimicking* or *Behavioral cloning*, is one of the oldest paradigms studied in robotics and machine learning [Schaal, 1999], and before that, in cognitive science and ethology (the scientific study of animal behavior) [Piaget, 1951]. Historically, this topic of study suffered from a prejudice that "imitating, or mimicking, is not an expression of higher intelligence". This attitude started to change in the 70's with a growing evidence reported by many scientists about the importance of imitation in both human and animal learning. As an example of that, Meltzoff and Moore [1977] reported on the ability of 21 day old infants to imitate both facial and manual gestures. Thus, the ability to map a perceived facial gesture to their own's gesture was showed to be innate. It was also discovered that many animals were unable to learn by imitation [Meltzoff, 1995; Byrne, 1995], these finding contributed to today's view of imitation as an important expression of higher intelligence [Schaal, 1999; Whiten, 2002]. Recently, the discovery of *mirror neuron system*, a network of human brain areas specialized in imitation, opened the way to a better understanding of the neural foundations of the complex mechanisms of imitation [Decety et al., 2002]. Inspired by this new evidence, imitation learning has become again a core topic of research in robotics and machine learning [Billard, 2002; Calinon and Billard, 2005; Verma and Rao, 2006; Hoey and Little, 2007], making use of sophisticated probabilistic models.

In this chapter, we focus on imitation learning for solving tasks that can be casted as a Markov Decision Process (MDP). A detailed overview of direct imitation learning methods for solving sequential decision-making problems is given by [Schaal et al., 2003].

Direct imitation learning is generally reduced to a supervised learning problem, where the

expert's policy is directly learned by mapping states to actions. To do so, states and actions should be identifiable, which is a serious issue in imitation learning, since the expert's internal (mental) states are hidden from the learner, and the executed actions can be perceived only through their effects. Thus, the primitive actions need to be defined in terms of variables that can be perceived, for instance, the acceleration of the finger tip in the task of pole balancing instead of the commands sent to the motor neurons. This problem, called *task-level learning*, was discussed in details by Aboaf et al. [1989]. Task-level imitation requires prior knowledge of how a task-level command (the desired action) can be converted into an actuator-level command (the realization of the action).

The best known example of a system built on this paradigm is ALVINN [Pomerleau, 1989], where a neural network was trained to learn a mapping between a road image and a vehicle steering action. Supervised learning was also used for the task of pole balancing with a simulated pole [Nechyba and Xu, 1995]. For this purpose, a neural network has been trained on the task of balancing a pole, the training data was provided by a human demonstrator. This method was also adopted by several mobile robotic groups, were an agent learns to navigate in an indoor environment by following a human and repeating her actions [Dautenhahn, 1995; Nechyba and Xu, 1995].

Direct imitation learning can also be reduced to a problem of learning the parameters of a dynamical system. The learner agent represents the expert's policy with a probabilistic model such as HMMs or (PO)MDPs. The dynamics of this model can be learned from the demonstrated behavior. Atkeson and Schaal [1997] showed how a robot arm agent can use a probabilistic model to represent the behavior of a human balancing a pole, and learn it after a few trials. This approach allows the learner agent to overcome the problem of generalization in supervised learning methods, but it may need a great number of training trajectories before converging to accurate parameters if the environment is highly stochastic.

Despite the growing body of work that has been done over the past few years on imitation learning, there has been less focus on applying these ideas to partially observable environments. Some of the proposed solutions [Verma and Rao, 2006; Hoey and Little, 2007] use the POMDP model to represent the policy of the expert, and then learn the parameters of this POMDP by using the Baum-Welch method [Koenig and Simmons, 1996], which needs considerable learning data, and is subject to local minima unless a good initialization of the parameters is provided. Initializing the parameters of a policy is not so easy because the policy states correspond to mental states rather than to physical situations.

In this chapter, we propose a new method for learning the expert's policy in a partially observable environment. Our approach is based on using predictive representations [Littman et al., 2001], which are generally easier to learn than the state-based representations in partially observable domains [James, 2005]. This approach is presented in Section 6.4.1.

6.3 Inverse Reinforcement Learning

6.3.1 Overview

The key idea behind inverse reinforcement learning is that the reward function is the most succinct hypothesis for explaining a behavior. Consequently, an observed behavior can be better generalized to other situations when the reward function is known. In this section, we start by presenting a background on the theoretical tools used in IRL. Then, we present the general IRL formulation, followed by two algorithms for imitation learning based on IRL, namely Linear Programming Apprenticeship Learning (LPAL) [Syed et al., 2008], and Maximum Margin Planning (MMP) [Ratliff et al., 2006].

6.3.2 Preliminaries

We denote by MDP\R a Markov Decision Process without a reward function, i.e. a tuple $(\mathcal{S}, \mathcal{A}, T, \alpha, \gamma)$, where: $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, T is a transition function, α is an initial state distribution defined as $\alpha(s) = Pr(s_0 = s)$, and γ is a discount factor (see Chapter 2). We assume that there exists a set of k feature vectors ϕ_i, and the reward function R is given by a linear combination of these features with weights w_i:

$$\forall s \in \mathcal{S}, \forall a \in \mathcal{A} : R(s,a) = \sum_{i=0}^{k} w_i \phi_i(s,a) \tag{6.1}$$

The agent decides which action to execute according to a policy π, defined as:

$$\pi(s,a) = Pr(a_t = a | s_t = s)$$

The value $V(\pi)$ of a policy π is the expected sum of rewards that the agent will receive if it follows the policy π, i.e.

$$V(\pi) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | \alpha, \pi, T] \tag{6.2}$$

An optimal policy π is one satisfying $\pi = \arg\max_\pi V(\pi)$.

The occupancy μ_π of a policy π, also known as the discounted state-action visit distribution or the state-action visit frequencies, is defined as:

$$\mu_\pi(s,a) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t \delta_{s_t,s} \delta_{a_t,a} | \alpha, \pi, T]$$

where δ is the Kronecker delta: $\delta_{s_t,s} = 1$ if $s_t = s$ and $\delta_{s_t,s} = 0$ if $s_t \neq s$.

The following linear constraints, known as Bellman-flow constraints, are necessary and sufficient to define the occupancy of a policy π:

$$\mu_\pi(s) = \alpha(s) + \gamma \sum_{s' \in \mathcal{S}} \sum_{a \in \mathcal{A}} \mu_\pi(s', a) T(s', a, s) \tag{6.3}$$
$$\sum_{a \in \mathcal{A}} \mu_\pi(s, a) = \mu_\pi(s)$$
$$\mu_\pi(s, a) \geqslant 0$$

A policy π is well-defined by its occupancy μ_π, we will interchangeably use π and μ_π to denote a policy. The set of feasible occupancy measures is denoted by $\mathcal{G}$.

The frequency of a feature ϕ_i using a policy π is given by:

$$\begin{aligned} v_{i,\pi} &= \mathbb{E}[\sum_{t=0}^{\infty} \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \gamma^t \delta_{s_t,s} \delta_{a_t,a} \phi_i(s, a) | \alpha, \pi, T] \\ &= \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t \delta_{s_t,s} \delta_{a_t,a} \phi_i(s, a) | \alpha, \pi, T] \\ &= F(i, .) \mu_\pi \end{aligned} \tag{6.4}$$

where F is a $k \times |\mathcal{S}||\mathcal{A}|$ feature matrix, such that $F(i, (s, a)) = \phi_i(s, a)$.

Using the definitions given by Equations 6.2 and 6.4, the value of a policy π can be written as a linear function of the feature frequencies:

$$\begin{aligned} V(\pi) &= \sum_{i=0}^{k} w_i v_{i,\pi} \\ &= w^T F \mu_\pi \end{aligned} \tag{6.5}$$

Therefore, the value of a policy is completely determined by the expected frequencies of the features ϕ_i.

6.3.3 Apprenticeship Learning via Inverse Reinforcement Learning

The aim of apprenticeship learning is to find a policy π that is at least as good as the expert's policy π^E. The value functions of π and π^E cannot be directly compared, since the true reward function is *unknown*. As a solution to this problem, Ng and Russell [2000] proposed to first learn a reward function, assuming that the expert is optimal, and then use it to recover the expert's generalized policy. This can be achieved by solving the following equation $\forall \pi' \in \Pi : V(\pi^E) \geqslant V(\pi')$ where the unknown variables are the rewards and Π denotes the set of all the possible policies. By using the definition of Equation 6.5, this problem can be reduced to the following system of linear equations, where the variables are w:

$$\forall \pi' \in \Pi : w^T F \mu_{\pi^E} \geqslant w^T F \mu_{\pi'} \tag{6.6}$$

However, the problem of learning a reward function given an optimal policy is ill-posed. Indeed, a large class of reward functions, including all constant functions for instance, may lead to the same optimal policy and solve Equation 6.6.

To overcome this problem, Abbeel and Ng [2004] did not consider recovering a reward function, instead, their algorithm returns a policy π with a bounded loss in the value function, i.e. $\| V(\pi) - V(\pi^E) \| \leqslant \epsilon$, where the value is measured with respect to the expert's unknown reward function. This property is derived from the fact that when the feature frequencies of two policies match, their cumulative rewards match as well, assuming the reward is a linear function of the features. However, this algorithm iteratively calls an MDP planner as a subroutine, which considerably affects its computational efficiency. More importantly, this algorithm cannot efficiently generalize the learned policy from one region of the state space to another, or when the start and goal states change. This is due to the fact that the frequencies of the features depend on the executed policy, as well as the transition probabilities, which may vary from one region to another.

Ratliff et al. [2006] noted that the regret, corresponding to the difference in value function compared to the expert's policy, is convex but nondifferentiable, and they successfully applied the subgradient technique to learn policies minimizing the regret. This method will be presented in Section 6.3.4. Along these lines, Syed and Schapire [2008] proposed to use efficient game-theoretic methods for minimizing the regret. In this method, the reward weights correspond to the weights minimizing the loss in value function in the worst case. Based on the same technique, Syed et al. [2008] derived a simpler algorithm for the case where all the reward weights are positive. This algorithm, known as Linear Programming Apprenticeship Learning (LPAL) will be presented in Section 6.3.5.

For their part, Neu and Szepesvári [2007] proposed to use a different loss function, based on the difference between the expert's policy and the learned policy rather than on the difference between the corresponding value functions. They utilized a natural gradient method (see Section 5.3.4 for a brief presentation of natural gradients) to learn policies that are close to the expert's one. The authors tested the proposed method on artificial domains and found it to be more reliable and efficient than some other methods.

Bayesian approaches for selecting a reward function also have been investigated by Ramachandran and Amir [2007]. In this work, the authors assumed that the action-selection policy is a softmax function, and used a Monte Carlo Markov Chain (MCMC) method to sample posterior reward distributions. This approach requires that a prior distribution on the reward functions is known, which is not always obvious if no domain knowledge is available.

Maximum entropy methods for generalizing the expert's policy have been considered by Ziebart et al. [2008]. The principle of maximum entropy is that the hypothesis that generalizes the best is the one that does not make any further assumptions beyond the observed examples. Ziebart et al. [2008] used a parametric softmax distribution on all the possible

optimal policies that match the expert's policy. The parameters of this distribution are the weights of the reward function, and the optimal weights are those that maximize the entropy of this distribution.

Lopes et al. [2009] used an approach combining active learning and bayesian inverse reinforcement learning for recovering the reward function. Given the examples provided by the expert and a prior distribution on the reward functions, a posterior distribution on the rewards is calculated just as in [Ramachandran and Amir, 2007]. However, the authors assumed that the learner agent can inquire information about missing examples from the expert. To minimize the number of these queries, the learner agent inquires information about only the states and actions that may reduce the entropy of the distribution on the rewards. This approach is well-suited for problems where the learner agent is constantly interacting with the expert, but not for the general context.

In the next two subsections (6.3.4 and 6.3.5), we provide a detailed description of two IRL algorithms that will be used for comparison later in this chapter.

6.3.4 Maximum Margin Planning

Maximum Margin Planning (MMP), proposed by Ratliff et al. [2006], returns a vector of reward weights w, such that the value of the expert's policy $w^T F \mu_{\pi^E}$ is higher than the value of an alternative policy $w^T F \mu$ by a margin that scales with the number of expert's actions that are different from the actions of the alternative policy. This criterion is directly specified in the cost function minimized in the algorithm:

$$c_q(w) = \Big(\max_{\mu \in \mathcal{G}} (w^T F + l)\mu - w^T F \mu_{\pi^E} \Big)^q + \frac{\lambda}{2} \parallel w \parallel^2 \tag{6.7}$$

This latter equation corresponds to the cost used in MMP for one MDP domain, where $q \in \{1, 2\}$ defines the slack penalization, λ is a regularization parameter, and l is a deviation cost vector, defined as: $l(s, a) = 1 - \pi^E(s, a)$. Intuitively, a policy maximizing the cost-augmented reward vector $(w^T F + l)$ is completely different from the expert's policy, since it is given an additional reward $l(s, a)$ for the actions that are different from those of the expert. The algorithm directly minimizes the difference between the value divergence $w^T F \mu_{\pi^E} - w^T F \mu$ and the policy divergence $l\mu$.

The cost function c_q is convex, but nondifferentiable. Ratliff et al. [2006] showed that c_q can be minimized by using a generalization of gradient ascent called the subgradient method. For a given reward w, a subgradient g_w^q is given by:

$$g_w^q = q \Big((w^T F + l)\mu^+ - w^T F \mu_{\pi^E} \Big)^{q-1} . F \Delta^w \mu^E + \lambda w \tag{6.8}$$

where $\mu^+ = \arg\max_{\mu \in \mathcal{G}} (w^T F + l)\mu$, $\Delta^w \mu^E = \mu^+ - \mu^E$.

As for most of apprenticeship learning algorithms, MMP requires the knowledge of the feature frequencies $F(i,.)\mu_{\pi^E} = v_{i,\pi^E}$. These frequencies can be analytically calculated (using Bellman-flow constraints) only if a complete expert policy is provided. However, the expert provides only a sequence of M demonstration trajectories $t_m = (s_1^m, a_1^m, \ldots, s_H^m, a_H^m,)$. The estimated feature frequencies $\hat{v}_{i,\pi^E}$, which are used in lieu of v_{i,π^E} in MMP, are given by:

$$\hat{v}_{i,\pi^E} = \frac{1}{M} \sum_{m=1}^{M} \sum_{t=1}^{H} \gamma^t \phi_i(s_t^m, a_t^m) \tag{6.9}$$

There are nevertheless many problems related to this approximation. First, the estimated frequencies $\hat{v}_{i,\pi^E}$ can be very different from the true ones when the demonstration trajectories are few. Second, the frequencies $\hat{v}_{i,\pi^E}$ are estimated for a finite horizon H, whereas the frequencies $v_{i,\pi}$ used in the objective function (Equation 6.7), are calculated for an infinite horizon (Equation 6.4). In practice, these two values are too different and cannot be compared as done in the cost function. The frequencies v_{i,π^E} are a function of both a policy and the transition probabilities, the empirical estimation of v_{i,π^E} does not take advantage of the fact that the transition probabilities are known. Finally, a frequency $\hat{v}_{i,\pi^E}$ cannot even be estimated if ϕ_i takes non null values only in states that did not appear in the demonstration. These issues of estimating the frequencies of the features are treated in Sections 6.5, 6.6 and 6.7.

6.3.5 Linear Programming Apprenticeship Learning

Linear Programming Apprenticeship Learning (LPAL), proposed by Syed et al. [2008], is a fast IRL algorithm that is based on the following observation: if all the weights w_i of the reward features are positive, and for some policy π we have $v^E = \min_{i=0,\ldots,k-1}[v_{i,\pi} - v_{i,\pi^E}]$ then $V(\pi) \geqslant V(\pi^E) + \sum_{i=0}^{k-1} w_i v^E$, where k is the number of features.

LPAL consists in maximizing the margin v^E, aiming to find policies that might outperform the expert's one by a margin of $\sum_{i=0}^{k-1} w_i v^E$. The maximal value of v^E is found by solving the following linear program:

$$\begin{aligned}
&\max_{v,\mu_\pi} \quad v \\
&\text{subject to} \\
&\qquad \forall i \in \{0,\ldots,k-1\} : v \leqslant \underbrace{\sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} \mu_\pi(s,a)\phi_i(s,a)}_{v_{i,\pi}} - v_{i,\pi^E} \\
&\qquad \mu_\pi(s) = \alpha(s) + \gamma \sum_{s'\in\mathcal{S}} \sum_{a\in\mathcal{A}} \mu_\pi(s',a) T(s',a,s) \\
&\qquad \sum_{a\in\mathcal{A}} \mu_\pi(s,a) = \mu_\pi(s) \\
&\qquad \mu_\pi(s,a) \geqslant 0
\end{aligned} \tag{6.10}$$

The last three inequalities in this linear program correspond to the Bellman-flow constraints (Equation 6.4) defining the feasible set of μ_π. The learned policy π is given by:

$$\pi(s,a) = \frac{\mu_\pi(s,a)}{\sum_{a' \in \mathcal{A}} \mu_\pi(s,a')} \tag{6.11}$$

As for Maximum Margin Planning (see the discussion in the previous section), LPAL requires the knowledge of the feature frequencies v_{i,π^E} (in Equation 6.11). This issue is treated in Sections 6.5, 6.6 and 6.7.

In the next section, we return to direct imitation learning approaches (behavioral cloning), and we propose to use Predictive Policy Representations (PPRs) for the case where the environment is partially observable.

6.4 Imitation Learning In Partially Observable Environments

6.4.1 Motivating problems

Intelligent wheelchairs are one instance of assistive robots where the human user is constantly interacting with the machine [Pineau and Atrash, 2007]. They are generally designed to assist individuals with mobility impairments who have difficulty with fine motor skills, and therefore cannot easily or safely operate a standard motorized wheelchair. The intelligent controller is designed to reduce their physical (and sometimes cognitive) load. Many of these individuals however are not willing to relinquish full control to an intelligent controller, thus a shared control solution is preferred.

As part of this shared control, the intelligent wheelchair should be able to detect the anomalous behavior of the user and to switch into automatic control mode (or alert the user). An example of anomalous (or undesirable) behavior may be as simple as the wheelchair hitting a wall, which is a frequent occurrence for many wheelchair users. Detecting more complex deviations from usual behavior is a very hard problem, since it depends on having a notion of the normal behavior. This problem can be cast as a problem of learning a policy for the human controller in a partially observable environment, as the sensors used by the wheelchair, such as laser rangefinders, do not precisely determine the location of the wheelchair.

The second problem we are interested in addressing is the classical pathfinding problem. In general, this problem can be solved efficiently if the environment dynamics are known in advance (e.g. using A*-type methods, or the more complex POMDP framework). But these approaches work best when the optimization criterion is simple (e.g. minimum distance). Note however that the shortest path is not always the user's preferred path, and in some

cases the user may wish to choose between several short paths. This problem can also be cast in the imitation learning framework, and solved with the methods outlined in this chapter.

To solve these problems, we propose to model the human's policy with a Predictive Policy Representation. The techniques used for learning predictive representations are reviewed in the next section.

6.4.2 Learning Predictive Representations

Most of the PSR literature is devoted to the problem of learning a model from a sequence of actions and observations. This problem can be divided into two parts: *discovery* of a set of core tests, and *learning* the parameters corresponding to the core tests (the vectors m_{ao} and m_{aoq}, see Section 2.4).

The first algorithm for learning PSRs [Singh et al., 2003] focused only on learning the parameters, and assumed that the set of core tests was given. The authors defined an error function corresponding to the squared difference between the probabilities of future tests according to the PSR model and the actual probabilities of the same tests. A stochastic gradient-descent was then used to update the parameters at each time-step and to gradually converge to a local optimum. However, the authors argued that the gradient of the error function is complex to compute because the derivative of the belief state requires taking into account the entire history. Instead, they used a *myopic* gradient where it was assumed that the derivative of the belief states is zero. This algorithm obtained mixed results on a set of POMDP domains taken from [Cassandra, 1998], and a high prediction error was reported for relatively large domains.

Analytical Discovery and Learning (ADL) [James and Singh, 2004] was the first algorithm to consider the problem of discovering a set of core tests. To do so, the authors constructed a large matrix D, known as the dynamics matrix [Singh et al., 2004], that captures all the dynamics of the system. The rows of the dynamics matrix correspond to the histories and the columns correspond to the future tests, and the entries of the matrix are the conditional success probabilities of the tests after each possible history, i.e. $D(h, q) = Pr(q|h)$. This matrix can be estimated by sampling sequences of actions and observations and resetting the system to an initial distribution at the end of each trial. At the end of this process, the core tests will be the tests corresponding to linearly independent columns. The weight vectors m_{ao} and m_{aoq} can then be found by simply solving a system of linear equations. The empirical results showed that ADL was able to discover an accurate set of core tests, and to find good model parameters of dynamical systems where the previous myopic algorithm had difficulty. However, this algorithm is limited by the size of available memory since the size of the dynamics matrix grows double exponentially with respect to the horizon of planning.

Rosencrantz et al. [2004] proposed a learning algorithm close to ADL, where the dimensionality of the model was reduced even more by performing a Singular Value Decomposition (SVD) on the dynamics matrix. The resulting PSR model, called Transformed PSR (TPSR), was used in a real world robot tracking task, and performed significantly better than a Hidden Markov Model. An analysis of ADL-style algorithms is given in [Wiewiora, 2005].

Resetting the system into an initial state distribution is not always possible in practice. To solve this problem, Wolfe [2009] proposed an algorithm for learning the parameters of a PSR model from a single trajectory of actions and observations. Assuming that a set of core tests is provided, the algorithm updates the parameters m_{ao} and m_{aoq} by gradually adding the temporal difference (TD) between the prediction of the model before perceiving an observation, and its prediction after. This algorithm outperformed all the previous learning algorithms for PSRs, as well as *Expectation Maximization* (EM) algorithm, which is considered as the standard method for learning POMDPs.

Another solution to the problem of learning PSRs in dynamical systems without reset was given by McCracken and Bowling [2006]. As in [Singh et al., 2003], a gradient-descent method was used to update the PSR parameters, while a dynamics matrix was learned and used for discovering core tests, as done in ADL [James and Singh, 2004]. Moreover, a set of constraints was used to restrict the entries of the dynamics matrix to be valid distributions. The empirical results showed that this algorithm is competitive with the previous ones.

Finally, the problem of learning with *non-blind* (non-uniform) policies was considered in [Bowling et al., 2006], where the learner agent selects the actions that improve the accuracy of the PSR parameters. Bowling et al. [2006] principally provided Monte Carlo estimators of the probabilities of tests, where the bias introduced by the exploration policy was removed.

6.4.3 Imitation Learning with Predictive Policy Representations

In partially observable environments, the learner agent perceives only the actions of the expert and its own observations. Based on this data, it should be able to extract the expert's policy that is generating this control. Generally, a human's behavior is non-stationary, i.e. it does not depend only on the state of the system, but also on the time-step in which an action is performed. However, in order to avoid additional complexity to the problem, we will make the assumption that the expert's policy is stationary.

In the case where the observation models of the expert and the agent match, imitation learning can be reduced to a problem of learning the parameters of a PPR model, which is basically equivalent to the problem of learning the parameters of a PSR. We will return to the general case, where the expert and the agent have heterogenous observations, latter in this section.

6.4.3.1 Learning PPRs Versus FSMs

Contrary to Finite-State Machines (FSMs), PPRs can be learned by simply counting the different tests appearing in a trajectory. This is due to the fact that the core tests of PPRs can be observed during the learning process, whereas the hidden internal states in FSMs cannot be observed. In fact, to estimate the probability that a core test q appears after a history h, we should only repeat h several times and count the number of times q occurs after h. But to estimate the probability of being in a hidden internal state g after a history h, even if one repeats h several times, one should use the parameters of the FSM to estimate $Pr(g|h)$ since g can be observed only through the selected actions. However, FSM parameters cannot be known during the process of learning them. Hence, FSM learning algorithms, such as the Baum-Welch algorithm [Koenig and Simmons, 1996], start with randomly chosen parameters, and then alternate between the phase of estimating the internal belief states b_t (distributions on the internal states) for each time-step t of the training sequence, and the phase of optimizing the FSM parameters, until no further optimization can be achieved. The final result heavily depends on the initial parameters, and can be trapped in a local optimum.

6.4.3.2 Analytical Discovery and Learning of Predictive Policy Representations

In pathfinding problems, the goal of the task is to reach a final state, so the system can always be restarted once the goal was reached. If there is no final state, one can cut the trajectories at distant time-steps, and consider that the system has been restarted at these points. Therefore, we propose to use an efficient algorithm for learning PSRs in dynamical systems with reset, and adapt it in order to learn PPRs. The proposed algorithm, presented in Algorithm 9, is a simple adaptation of the Analytical Discovery and Learning (ADL) [James and Singh, 2004] described in Section 6.4.2.

The learner agent collects several samples of trajectories $t^i = a_1^i o_1^i \dots a_m^i o_m^i$ by observing the expert's behavior. The first step of the algorithm consists in constructing a matrix $\mathcal{D}$, defined as:

$$\mathcal{D}(h, q) = \hat{Pr}(q|h)$$

where h is a history and q is a test occurring in the expert's trajectories.
The estimated success probability $\hat{Pr}(q|\varepsilon)$ of a test $q = a_1 o_1 a_2 o_2 \dots a_k o_k$ is calculated by using the following estimator:

$$\hat{Pr}(q|\varepsilon) = \prod_{t=0}^{k-1} \hat{Pr}(a_{t+1}|a_1 o_1 a_2 o_2 \dots a_t o_t) = \prod_{t=0}^{k-1} \frac{\#a_1 o_1 a_2 o_2 \dots a_t o_t a_{t+1}}{\sum_{a \in \mathcal{A}} \#a_1 o_1 a_2 o_2 \dots a_t o_t a}$$

where $o_0 \stackrel{def}{=} \varepsilon$, and $\#q$ is the number of the expert's trajectories that started with q.

This latter estimator was proposed by Bowling et al. [2006] in order to eliminate the bias introduced by the system dynamics. It was originally used to learn the parameters of a dynamical system when the exploration policy is not *blind*, i.e. the executed actions depend on the perceived observations. This issue is inherent to the problem of learning a policy from a demonstration, because the observations depend on the executed actions, unless the system is uncontrolled.

The following Bayes' Rule is used to find other entries of the matrix $\mathcal{D}$:

$$\hat{Pr}(q|h) = \frac{\hat{Pr}(hq|\varepsilon)}{\hat{Pr}(h|\varepsilon)}$$

Input: Samples of the expert's trajectories $t^i = a_1^i o_1^i a_2^i o_2^i \ldots$;
Output: A PPR model of the expert's policy $\langle \mathcal{A}, \mathcal{O}, Q, Pr(Q|\varepsilon), \{m_{oa}\}, \{m_{oaq}\} \rangle$;

1 Construct the dynamics matrix $\mathcal{D}$ by estimating probabilities $\mathcal{D}(h,q) = \hat{Pr}(q|h)$ for all the sequences q and h occurring in the trajectories t^i;
2 Initialize the set of core tests Q with the void test ε, and set $Pr(\varepsilon|\varepsilon) = 1$;
3 **foreach** $q \in Q, a \in \mathcal{A}, o \in \mathcal{O}$ **do**
4 **if** *the column of the test oaq in* $\mathcal{D}$ *is linearly dependent upon the columns of the core tests* $q' \in Q$*;*
5 **then**
6 Find the vector m_{oaq} by solving the system of linear equations:

$$\forall h : \mathcal{D}(h, oaq) = \sum_{q_i \in Q} \mathcal{D}(h, q) m_{oaq}(q_i)$$

  if $q = \varepsilon$ **then** $m_{oa} = m_{oaq}$
7 **else**
8 Add the test oaq to the set of core tests Q;
9 $\forall a' \in \mathcal{A}, o' \in \mathcal{O} : m_{o'a'}(oaq) = 0$;
10 $Pr(oaq|\varepsilon) = \mathcal{D}(\varepsilon, oaq)$;
11 **end**
12 **end**

Algorithm 9: Analytical Discovery and Learning of Predictive Policy Representations.

Given the matrix $\mathcal{D}$, the next step is to find the set of core tests. This set is initialized with the void test ε (step 2). If q is a core test, then for every action a and every observation o, we check if the column of extended test oaq depends linearly upon the columns of the already discovered core tests, and calculate the approximate weight vector m_{oaq} in that case (steps 3-7). To this end, we perform a Gauss-Jordan elimination on the matrix $\mathcal{D}$, the eigenvalues will then correspond to diagonal elements, and the number of eigenvalues λ_i such that $\lambda_i \geqslant \epsilon$ approximately corresponds to the rank of the matrix, the cutoff threshold ϵ is used to reduce the number of discovered core tests as the entries of the matrix are approximate estimates of the true probabilities. The algorithm stops when all the extensions of all the core tests of

Q are linearly dependent upon the core tests of Q. In fact, one can prove that if a test is linearly dependent upon other tests, then all its extensions will also be dependent upon the same tests [Littman et al., 2001].

The main drawback of this algorithm is the computational cost of estimating the matrix $\mathcal{D}$, since we need to calculate the probability of each possible test after each possible history, which corresponds to $O(H|\mathcal{O}|^{2H}|\mathcal{A}|^{2H})$ operations for trajectories of maximal length H. But in practice, most of these tests are not experienced since they are generated by a particular dynamics and an expert's policy rather than uniformly.

6.4.3.3 Learning the full system

Alternately, one can also avoid the issue of biased samples discussed in the previous section, and significantly simplify the problem of imitative learning by considering the whole system (the expert's policy and the dynamics of the environment) as one uncontrolled system, with no actions. The couples (observation, action) will be the observations of a hidden Markov model. The parameters of this system can be used to predict at any time-step the probability of any couple (action, observation), and then the probability of any action by summing on the different observations. However, the dimension of this system is the product of the dimensions of the environment and the expert's policy. In fact, the states of this joint system are the couples (internal state of the expert, state of the environment), so the learner agent will need more samples to learn an accurate model, because it will be learning more than what is really needed. Thus, we do not pursue this approach.

6.4.3.4 Learning with heterogenous observations

In general, the observations can be different from the expert to the learner (the actions also can be different, but we do not consider this case in this thesis). The other important issue is that the observations of the expert are unknown to the learner, assuming that the learning is completely passive. In fact, the learner observes only the expert's actions and her own observations. Nevertheless, we will show that even in this setting the agent will be able to learn a policy with a finite number of states (or core tests for PPRs) if the expert's is indeed an FSM. For the sake of simplicity, we will adopt FSM notations instead of PPRs, but the result can be generalized to this latter model, since any FSM can be transformed into a PPR (Theorem 1).

Let $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{O}_E, \mathcal{O}_L, T, Z_E, Z_L \rangle$ be a POMDP model of the environment without a reward function, where $\mathcal{O}_E$ and Z_E indicate the expert's observation set and function, while $\mathcal{O}_L$ and Z_L indicate the learner's observation set and function. We assume that both the expert and the learner agents have the same actions.

Let us also assume that the expert's policy is stationary and described by a Finite-State Machine $\mathcal{P}_E = \langle \mathcal{G}_E, \mu_E, \omega_E \rangle$ (see Section 4.3 for the definition of an FSM). We will show that even from the learner agent's view, the expert is following a stationary policy described by another Finite-State Machine $\mathcal{P}_L = \langle \mathcal{G}_L, \mu_L, \omega_L \rangle$. In fact, the system dynamics $\mathcal{M}$ and the expert's policy $\mathcal{P}_E$ can be regrouped together in one hidden Markov model $\mathcal{M} \times \mathcal{P}_E = \langle \mathcal{S} \times \mathcal{G}_E \times \mathcal{O}_E, \mathcal{A} \times \mathcal{O}_L, T' \rangle$, where:

$$\begin{aligned} T'(\langle s, g_E, o_E \rangle, \langle a, o_L \rangle, \langle s', {g_E}', {o_E}' \rangle) &\stackrel{def}{=} Pr(\langle s', {g_E}', {o_E}' \rangle, \langle a, o_L \rangle | \langle s, g_E, o_E \rangle) \\ &= \mu^{o_E, a}(g_E) T(s, a, s') \omega^{o_E}(g_E, {g_E}') Z_E({o_E}', s', a) Z_L(o_L, s', a) \end{aligned}$$

From the joint model $\mathcal{M} \times \mathcal{P}_E$, one can extract an FSM $\mathcal{P}_L = \langle \mathcal{S} \times \mathcal{G}_E \times \mathcal{O}_E, \mu_L, \omega_L \rangle$, which describes only the policy of the expert as seen by the learner agent, and makes abstraction of the environment dynamics. The action selection function μ_L and the internal transition function ω_L are defined as:

$$\begin{cases} \mu_L^{o_L, a}(\langle s, g_E, o_E \rangle) = \mu_E^{o_E, a}(g_E) \\ \omega_L^{o_L}(\langle s, g_E, o_E \rangle, \langle s', {g_E}', {o_E}' \rangle) = \frac{\sum_{a \in \mathcal{A}} \mu_E^{o_E, a}(g_E) T'(\langle s, g_E, o_E \rangle, \langle a, o_L \rangle, \langle s', {g_E}', {o_E}' \rangle)}{Z_L(o_L, s', a)} \end{cases}$$

This latter Finite-State Machine contains $|\mathcal{S}| \times |\mathcal{G}_E| \times |\mathcal{O}_E|$ states. Any learning algorithm for FSM can be used to find the parameters of the policy described by $\mathcal{P}_L$, which represents the expert's policy from the learner's point of view. The learned policy contains at most $|\mathcal{S}| \times |\mathcal{G}_E| \times |\mathcal{O}_E|$ states (or core policy tests), the divergence between the learned policy and the actual policy depends on the divergence between the two observation models Z_E and Z_L.

6.4.4 Empirical Analysis

6.4.4.1 Environment

A simulated environment has been used to test our Algorithm 9 on the two problems presented in Section 6.4.1. The environment is a 20×20 grid (Figure 6.1), the state corresponds to the location of the wheelchair in the grid (275 possible positions) and its orientation (8 possible orientations). Therefore, there are 2200 different states. We consider only three actions: move forward, turn left, and turn right. The turn action results in a rotation of $\pi/4$.

For the obstacle avoidance problem, we consider only three observations: no obstacle, obstacle after one move, and obstacle ahead. These observations are the same for both the user (expert) and the learner. For the pathfinding problem, we consider that the wheelchair observes the presence or the absence of obstacles in the 8 adjacent positions, we have then 256 observations, whereas the user completely observes the state. The observations are aliasing, but they are deterministic, and the actions have stochastic effects: they succeed with probability 0.9, and have no effect with probability 0.1.

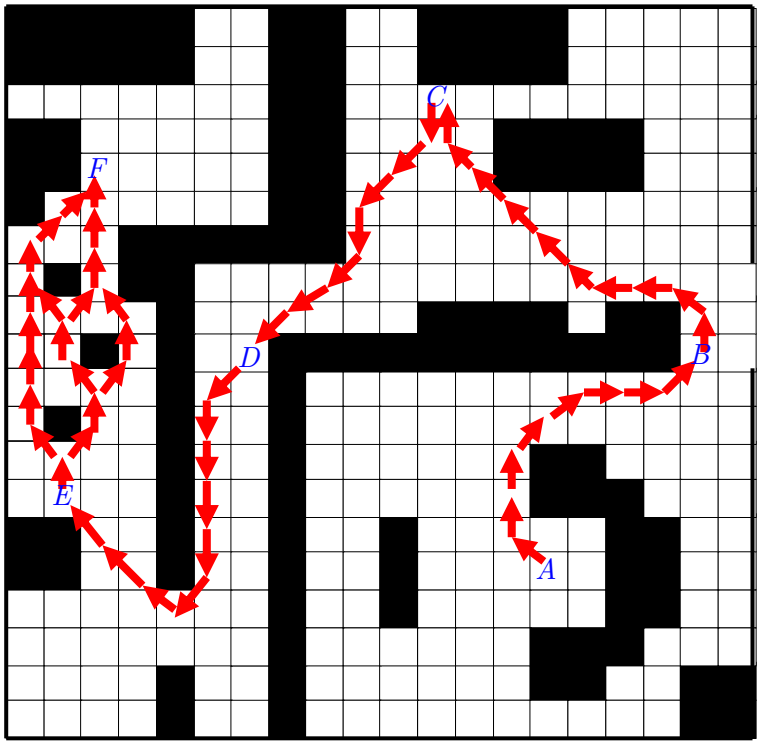


Figure 6.1: Simulated indoor environment of the robotic wheelchair.

6.4.4.2 The obstacle avoidance problem

The user's behavior was simulated by means of the Finite-State Machine described in Figure 6.2. In a normal situation (*internal state 1*), the user moves forward with probability 0.8, and turns left or right, with probability 0.2. When an obstacle is one step away (*internal state 2*), the user tries to avoid it by turning left or right with probability 0.9, but there is still a small probability, 0.1, that the user moves forward, because it can be looking for something on the obstacle (the obstacle is its goal). Last, if the obstacle is ahead (*internal state 3*), the user will turn left or right with probability 1, and will never move in the direction of the obstacle. In an abnormal behavior, we suppose that the user uniformly chooses its actions in internal states 1 and 2, but she still avoids obstacles in state 3 by turning left or right with probability 1. If we consider that the user can hit obstacles, the problem becomes no longer interesting since abnormal behavior could be detected each time the user moves towards an obstacle.

Note that both the environment and the user's models were oversimplified, because the principal goal of these experiments is only to compare the predictive representations to finite-state machines. In order to effectively solve the problems described in Section 6.4.1, one should consider more elaborated models, tacking into account the continuous nature of the state, action and observations spaces, as well as many hidden variables considered by the user in its decisions.

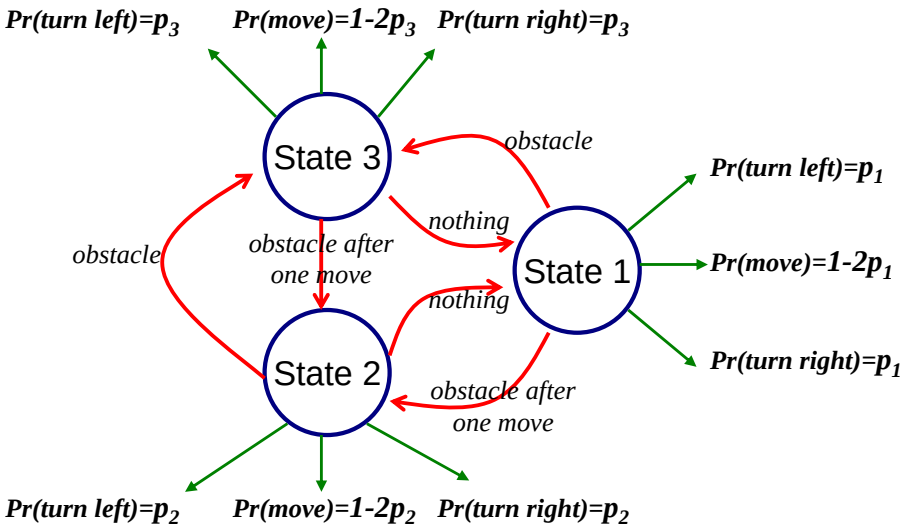


Figure 6.2: The finite-state machine used to simulate the wheelchair user's behavior. In a normal behavior, we have $p_1 = 1/10, p_2 = 9/20, p_3 = 1/2$. In an abnormal behavior, we have $p_1 = 1/3, p_2 = 1/3, p_3 = 1/2$, the user tends to move closer to obstacles.

To learn the user's normal behavior, we randomly generate the initial location of the wheelchair in the grid, then we use the models of the environment and the user's policy in order to generate one action and one observation at each time-step, we repeat this process H times. We also consider a reset point after every 3 time-steps. In fact, most of the decisions related to avoiding obstacles are made at most 3 time-steps before reaching the obstacle.

Given that the rank of the dynamics matrix $\mathcal{D}$ (see Algorithm 9) is very sensitive to the variations in the estimated probabilities, the number of discovered core tests can be very high for the smallest values of H. So, in order to keep just a few core tests, the cutoff threshold ϵ was set to 0.9. Algorithm 9 does not require any other hyper-parameters.

The same trajectory was used for learning the parameters of a Finite-State Machine with the Baum-Welch algorithm [Koenig and Simmons, 1996], which is a variant of the expectation-maximization method for learning POMDPs, and can be easily adapted for learning FSMs. Given an initial model, this algorithm calculates beliefs on the hidden states at each time-step of the trajectory, and uses this expectation as a basis for finding new parameters that maximize the likelihood of the observed actions. This operation is repeated until a fixed point, corresponding to a local maximum, is reached. In our experiments, the initial model is a 3-states FSM with random transition and action-selection probabilities.

To test the accuracy of our learned policies, we simulated a trajectory of $H = 100$ time-steps, and at each time-step t, we calculated for each action a the probability $\hat{Pr}(a|h_t)$ that the user will execute a according to the learned policy model, and compare it to $Pr(a|h_t)$, the true probability given by the FSM of Figure 6.2. The *Root Mean Square Deviation (RMSD)*

is given by:

$$\text{RMSD} = \sqrt{\frac{1}{H \times |\mathcal{A}|} \sum_{t=1}^{H} \sum_{a \in \mathcal{A}} [\hat{Pr}(a|h_t) - Pr(a|h_t)]^2}$$

Figure 6.3 shows the different values of the RMSD, averaged over 20 trajectories of length 100, as a function of the number of time-steps in the training trajectory. As expected, an accurate PPR model can be learned after only 40 episodes of 3 time-steps, with an average error of 0.04. The error continues to decrease slowly as more training steps are used. The FSM parameters are also quickly learned, but one should wait until the episode 100 to get an error close to zero. Figure 6.4 shows the *Kullback-Leibler* (KL) divergence between the learned policies and the true one. Notice the correlation between the KL divergence and the RMSD deviation, this means that the PPR distributions on the user's actions are also more accurate than the FSM distributions.

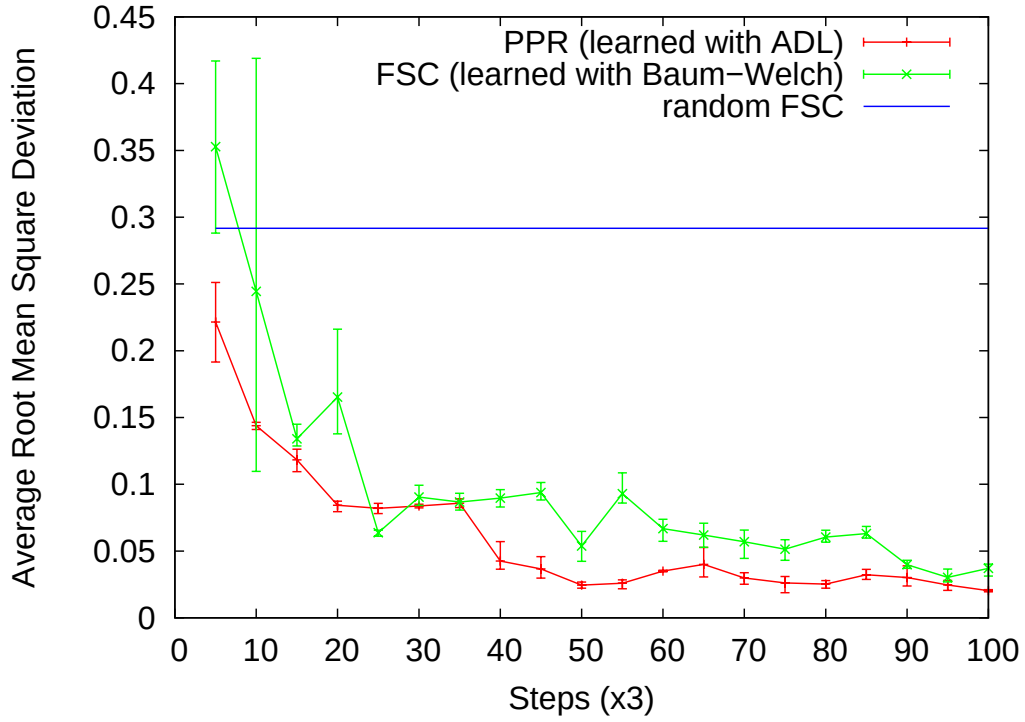


Figure 6.3: The average RMSD in predicting the user's normal actions in the obstacle avoidance problem, as a function of the training steps.

Finally, the learned policies were used to distinguish the normal behavior context from the abnormal one. For this end, we defined a meta FSM containing these two states of the user's behavior. In every state, the actions are generated by the corresponding FSM described in Figure 6.2. The belief state on the user's behavior is updated with the Bayes' Rule (Equation 5.1).

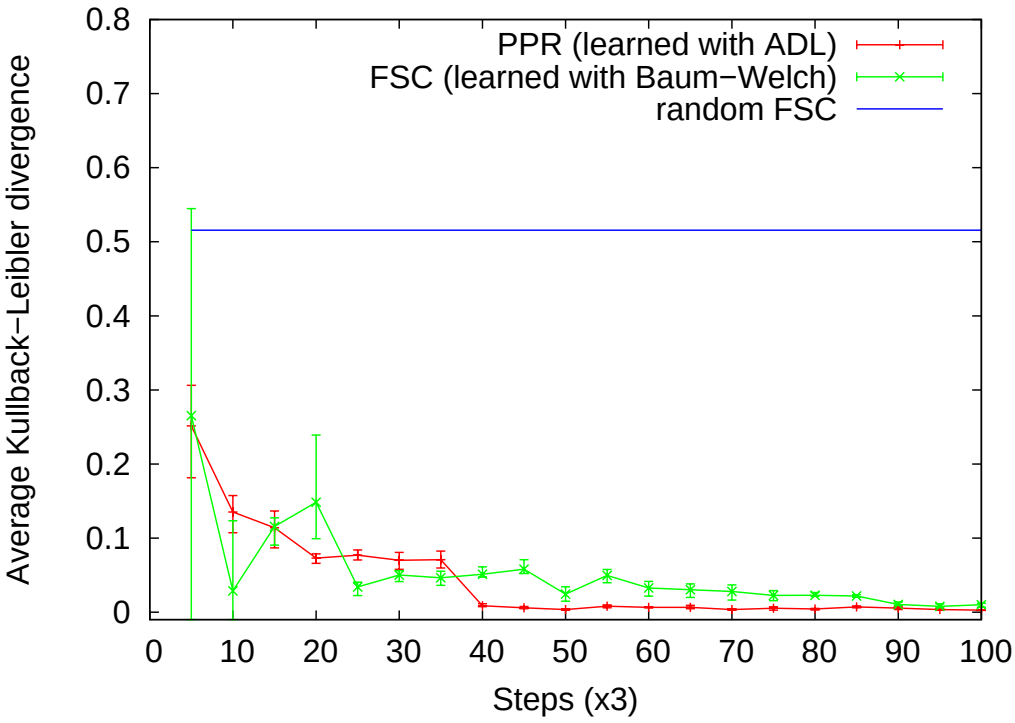


Figure 6.4: The average KL-divergence between the predictions of the learned model and those of the accurate model.

The initial belief state is set to $(0.5, 0.5)$, and the belief state at a given time-step indicates the most likely type of behavior that has been followed during the last few steps. We sampled 10 normal scenarios and 10 abnormal scenarios, and reported in Figure 6.5 the error rate in classifying the scenarios as a function of the number of time-steps used for training the PPR and the FSM models. Both PPR and FSM were able to properly classify the behavior. We observed that for the smallest training samples, PPR misclassifies the anomalous behaviors because of a missing core policy test that does not occur in shorter trajectories and its probability was automatically set to 0. Finally, Figure 6.6 shows how the deviations from a normal behavior to an abnormal behavior can be quickly detected by using the learned PPR policy. The user starts an anomalous behavior after step 75, and it is detected just a few steps later. After step 125, the user returns to its normal policy, and the belief on the normal state of the user increases gradually and becomes almost equal to 1 after step 160. Notice that this deviation is not reflected by hitting obstacles, but just by changing the probabilities of selecting actions (Figure 6.2).

6.4.4.3 The pathfinding problem

This problem is much more complex than the first one, because of the larger set of observations, the length of the trajectory, and more importantly, the user's observations do not match with

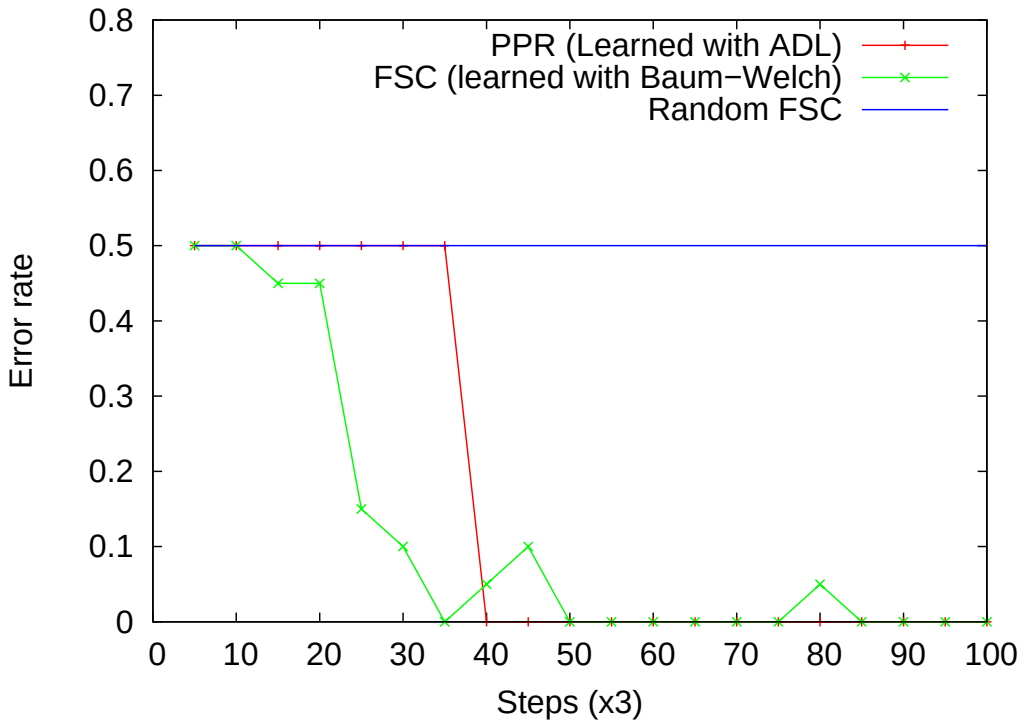


Figure 6.5: The average error rate in recognizing the user's anomalous behavior.

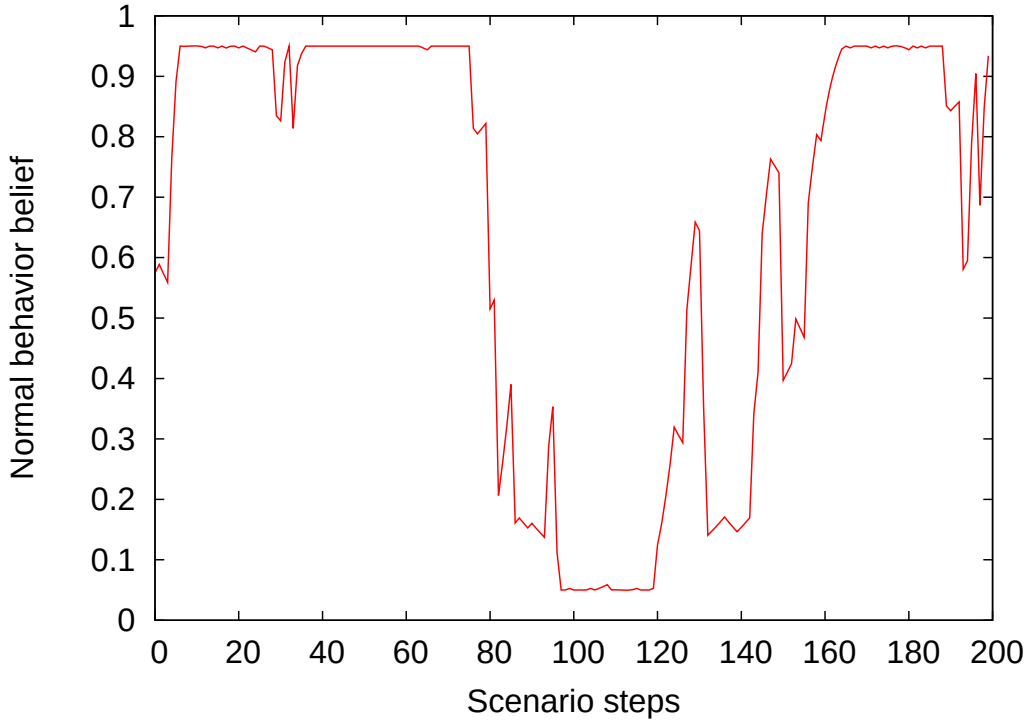


Figure 6.6: Detecting an abnormal behavior sequence within a normal behavior sequence, by using the learned PPR model. The anomalous behavior starts at step 75 and ends at step 125.

the wheelchair's observations. The user completely observes the state, whereas the wheelchair can observe only the obstacles surrounding it. To make the problem more challenging, we consider that the wheelchair perceives only its location and not its orientation, so it can never realize if a turn action has succeeded or not because the observation will be the same in both cases. The user's policy is described by the trajectory shown in Figure 6.1. There are 5 reset points in the global path, each one corresponds to a subgoal (a frequent destination of the user such as doors, tables ... etc.). Recall that the point is not to find the shortest paths, but to learn the user's preferred path. We repeatedly generate the trajectory from the start to the arrival positions, and used it for learning the underlying user's policy, even though the user's behavior can never be exactly reproduced in that case. The parameters of the learned FSM are randomly initialized, whereas the number of its internal states corresponds to the number of internal states in the optimal FSM, which is 20.

Figure 6.7 indicates the calculated RMSD, averaged over 20 trajectories. Notice that in this problem too, the PPR was learned with fewer training data compared to the FSM, and it converged quickly to a minimal error around 0.10. This latter error, indicated in Figure 6.7 by the error of the exact FSM, corresponds to the divergence between the true policy of the user, and this policy as seen by the learner, because of the difference between the two observation models (see Section 6.4.3.4).

For the last path ($E \rightarrow F$), we considered that both of the user and the learner use the same observation model. In this case, the observation corresponds to the presence or absence of an obstacle ahead the wheelchair, and the direction of the wheelchair. We have then 2×8 observations. Based on these partial observations, the user can follow different paths for moving from E to F. Here again, we notice that the divergence of the PPR learning algorithm decreases faster than the divergence of the FSM. Contrary to the previous paths, the final error (0.05) in this path is almost null because both of the user and the wheelchair have the same model of observation. The error of the PPR model fell down quickly after 15 repetitions because of a second core policy test ($\langle$obstacle ahead, direction north$\rangle$, $\langle$turn left$\rangle$) that was recognized as independent, with the cutoff ϵ set to 0.1, only after the 15^{th} repetition.

6.4.5 Discussion

Most of the previous work on imitation learning was restrained to the context of completely observable environments. In this section, we showed how to use Predictive Policy Representations (PPRs), the newly introduced method in Chapter 4, in order to represent the expert's policy in a partially observable environment. The advantage of PPRs, compared to Finite-State Machines (FSMs) for example, is the fact that their parameters are a function of only observable data (actions and observations). We tested this model on two problems related to assistive robotics. The first one is to learn the behavior of an automated wheelchair's

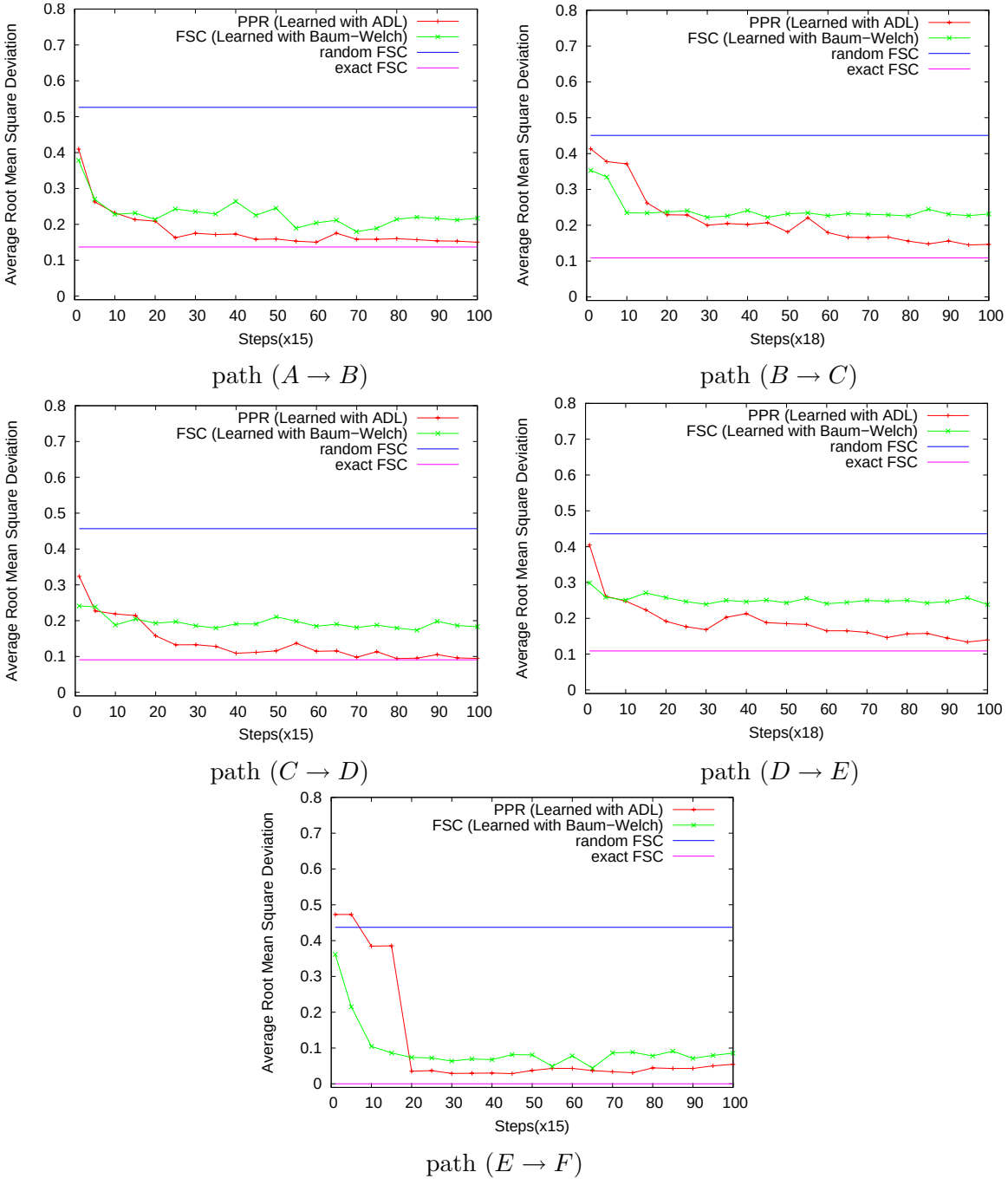


Figure 6.7: The root mean squared deviation on predicting the user's actions, as a function of the number of training steps.

user regarding obstacles avoidance, and to recognize deviations from this behavior in order to alert the user, or to switch to an automated control. The second problem is to imitate the user's motion policy in an indoor environment. In both problems, PPR outperforms the Baum-Welch algorithm for FSM, and the corresponding policies could be learned after a few repetitions.

In the next section, we will consider the more robust framework for imitation learning, namely Inverse Reinforcement Learning (IRL), and we will show how a special type of predictive representations, known as Homeomorphism of MDPs, can be used to solve a serious drawback of imitation learning algorithms that are based on this framework.

6.5 Imitation Learning with Local Homomorphisms

6.5.1 Overview

Most of Inverse Reinforcement Learning (IRL) algorithms rely on the assumption that the reward function is a linear combination of states features, and the frequency of encountering each feature can be accurately estimated from the demonstration. This assumption is considered in most of recent apprenticeship methods, despite the fact that the feature frequencies might be poorly estimated when the number of demonstrations is small, as we will show in our next experiments (Section 6.5.4). Moreover, in the absence of a complete expert's policy, the frequencies of the features that belong to only unvisited states cannot even be estimated.

In this section, we propose to use a transfer learning technique, known as soft homomorphism [Sorg and Singh, 2009], in order to generalize the expert's actions to unvisited regions of the state space. The generalized policy can then be used to analytically calculate the expected frequencies of the features. Contrary to previous direct methods, homomorphisms take into account the long term dependencies between different states. We will show that combining this transfer method with other apprenticeship algorithms provides a significant improvement in the quality of the learned policies.

6.5.2 Transfer Learning Through Homomorphism of MDPs

Transfer learning refers to the problem of using a policy learned for performing some task in order to perform a related, but different, task. The related task may be defined in a new domain, or in the same domain but in a different region of the state space. This problem has been widely studied in the context of reinforcement learning. An overview of the literature on transfer learning is out of the scope of this thesis, the interested reader might find an extended overview in [Taylor and Stone, 2009].

We will focus on a transfer method known as MDP homomorphism [Ravindran, 2004]. A homomorphism from an MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, R, \alpha, \gamma)$ to another MDP $\mathcal{M}' = (\mathcal{S}', \mathcal{A}, T', R', \alpha, \gamma)$ is a surjective function f, called *the transfer function*, defined as:

$$\begin{aligned} f: \quad & \mathcal{S} \to \mathcal{S}' \\ & \forall s \in \mathcal{S}, s' \in \mathcal{S}', \forall a \in \mathcal{A} : T'(f(s), a, s') = \sum_{\substack{s'' \in \mathcal{S} \\ f(s'')=s'}} T(s, a, s'') \end{aligned}$$

In other terms, a homomorphism of an MDP can be seen as a transformation that preserves its dynamics. Figure 6.8 shows an example of a homomorphism f from an MDP with 4 states $\{s_0, s_1, s_2, s_3\}$ to another one with 3 states $\{s'_0, s'_1, s'_2\}$, where $f(s_0) = s'_0$, $f(s_1) = s'_1$, and $f(s_2) = f(s_3) = s'_2$. In order to make the example simpler, we considered only one action in this example. Note that $Pr(s'_1|s'_0) = Pr(s_1|s_0)$, $Pr(s'_2|s'_1) = Pr(s_2|s_1) + Pr(s_3|s_1)$ (both s_2 and s_3 have as image s'_2), and $Pr(s'_2|s'_2) = Pr(s_2|s_2) + \underbrace{Pr(s_3|s_2)}_{0} = Pr(s_3|s_3) + \underbrace{Pr(s_2|s_3)}_{0}$.

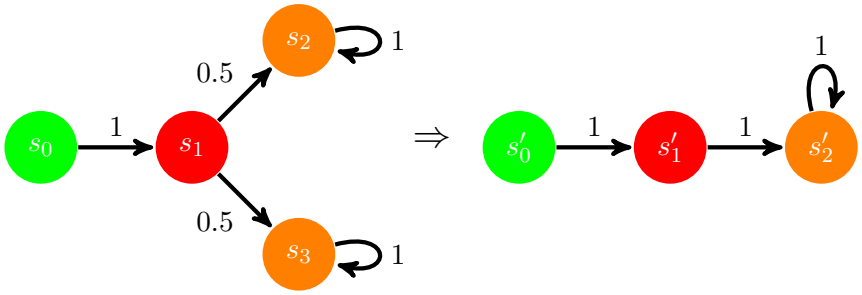


Figure 6.8: Homomorphism of Markov Decision Processes.

Unfortunately, finding a homomorphism is an NP-complete combinatory search problem. In this thesis, we will consider another variant of this approach known as Soft MDP homomorphism [Sorg and Singh, 2009]. The core idea of this latter method consists in finding a transfer function f that maps each state of an MDP model $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, R, \alpha, \gamma)$ to a probability distribution on the states of another MDP model $\mathcal{M}' = (\mathcal{S}', \mathcal{A}, T', R', \alpha, \gamma)$. Additionally, the mapping between the states of $\mathcal{S}$ and $\mathcal{S}'$ should preserve the transition probabilities:

$$\begin{aligned} f: \quad & \mathcal{S} \times \mathcal{S}' \to [0, 1] \\ & \forall s \in \mathcal{S}, s' \in \mathcal{S}', \forall a \in \mathcal{A} : \sum_{s'' \in \mathcal{S}} T(s, a, s'') f(s'', s') = \sum_{s'' \in \mathcal{S}'} f(s, s'') T'(s'', a, s') \end{aligned} \quad (6.12)$$

The reward function also should be preserved, but we will not consider this constraint since, in the context of apprenticeship learning, the reward function is unknown.

Notice the similarity between this latter equation and the equation $T^{ao}F = F\tilde{T}^{ao}$ used for compressing a POMDP in Section 3.6 (see also Figure 3.2). In fact, by defining the matrix T^a as $T^a(s, s') = T(s, a, s')$, the matrix $\tilde{T}^a$ as $\tilde{T}^a(s, s') = T'(s, a, s')$, and the compression matrix F as $F(s, s') = f(s, s')$, one can see that Equation 6.12 reduces to the equation $T^a F = F\tilde{T}^a$.

However, the distribution on the states of the MDP $\mathcal{M}'$ defined by the transfer function $f(s,.)$ cannot be used to track back the corresponding state s in the MDP $\mathcal{M}$. An obvious solution to this problem is to ensure that the matrix F is reversible. Instead, one can just add the constraints $Fm_s = e_s$ in order to ensure that the states $s \in \mathcal{S}$ can be tracked back after the compression, where m_s is an $|\mathcal{S}'|$-dimensional vector and e_s is an $|\mathcal{S}|$-dimensional vector defined as $e_s(s) = 1$ and $\forall s' \in \mathcal{S} - \{s\} : e_s(s') = 0$. By adding these constraints, the vectors of the distributions $f(s,.)$ belong to a vectorial space that corresponds to the Krylov subspace $Kr(\{T^a\}, \{e_s\})$, and the matrix F is given by the linearly independent vectors of that space. Therefore, the columns of the matrix F will correspond to vectors of the form $T^a \dots T^{a,o} e_s$, which is a predictive representation of the states in an MDP. A discussion of the link between PSRs and homomorphisms is given in [Wolfe, 2010a].

Sorg and Singh [2009] showed that soft homomorphisms can be used to transfer the values of the policies from an MDP model to another. In the next section, we will show how one can use soft homomorphisms in order to transfer actions from a subset of a state space to another subset of the same space.

6.5.3 Apprenticeship Learning with Local Homomorphisms

Given an MDP model without reward $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, \alpha, \gamma)$ (see subsection 6.3.2 for a recall) and a set of M trajectories provided by an expert, the state space $\mathcal{S}$ can be divided into two subsets: $\mathcal{S}^E$, the set of states that appear in the provided trajectories, and $\mathcal{S} \backslash \mathcal{S}^E$. For the states of $\mathcal{S}^E$, the expert's policy π^E can be directly inferred from the trajectories if it is deterministic, or estimated by calculating the frequencies of actions if it is stochastic. We will consider the general case and use $\hat{\pi}^E$ to denote the estimated expert's policy.

In order to generalize the policy $\hat{\pi}^E$ to $\mathcal{S} \backslash \mathcal{S}^E$, we first create a restrained MDP without a reward function $\mathcal{M}^E = (\mathcal{S}^E, \mathcal{A}, T^E, \alpha, \gamma)$, where the transition function T^E is defined as:

$$\forall s, s' \in \mathcal{S}^E, \forall a \in \mathcal{A} : \begin{cases} T^E(s,a,s') = T(s,a,s') & \text{if } s' \neq s \\ T^E(s,a,s) = T(s,a,s) + \sum_{s'' \in \mathcal{S} \backslash \mathcal{S}^E} T(s,a,s'') \end{cases}$$

This function ensures that all the transitions remain within the states of $\mathcal{S}^E$ by assuming that any transition that leads to a state outside of $\mathcal{S}^E$ has no effect.

The next step consists in finding a lossy soft homomorphism between $\mathcal{M}$ and $\mathcal{M}^E$, where the loss function corresponds to the error in preserving the transition probabilities according to Equation 6.12. The transfer function f of this homomorphism is found by solving the

following linear program:

$$\begin{aligned}
&\min_{f} \quad e && (9)\\
&\text{subject to}\\
&\forall s \in \mathcal{S}, s' \in \mathcal{S}^E, \forall a \in \mathcal{A} :\\
&|\sum_{s'' \in \mathcal{S}} T(s,a,s'')f(s'',s') - \sum_{s'' \in \mathcal{S}^E} f(s,s'')T^E(s'',a,s')| \leqslant e\\
&f(s,s') \geqslant 0\\
&\sum_{s' \in \mathcal{S}^E} f(s,s') = 1
\end{aligned}$$

The transfer function f can be seen as a measure of similarity between two states. One can use this measure in order to define the generalized policy $\hat{\pi}^E$ as follows:

$$\forall s \in \mathcal{S} \backslash \mathcal{S}^E, \forall a \in \mathcal{A} : \hat{\pi}^E(s,a) = \sum_{s' \in \mathcal{S}^E} f(s,s')\hat{\pi}^E(s',a)$$

Unfortunately, this method scales up poorly with respect to the number of states visited by the expert and the number of states in the corresponding domain. This is due to the fact that $|\mathcal{S}^E| \times |\mathcal{S}|$ variables are used in this linear program. To improve the computational efficiency of this approach, we redefine the function f as a measure of local similarity between two states. We denote by s^d the set of states that can be reached from state s within a distance of d steps, and by $\mathcal{M}^{s,d}(s^d, \mathcal{A}, T^{s,d}, \alpha, \gamma)$ the MDP\R model defined on these states. The transition function $T^{s,d}$ is then defined as:

$$\forall s, s' \in s^d, \forall a \in \mathcal{A} : \begin{cases} T^{s,d} = T(s,a,s') & \text{if } s' \neq s \\ T^{s,d} = T(s,a,s) + \sum_{s'' \in \mathcal{S} \backslash s^d} T(s,a,s'') \end{cases}$$

Given a distance d and a threshold ϵ, two states s and s' are considered as locally similar if there exists a soft homeomorphism between $\mathcal{M}^{s,d}$ and $\mathcal{M}^{s',d}$ with a transfer error not greater than ϵ. This property is checked by solving the following linear program:

$$\begin{aligned}
&\min_{f} \quad e && (6.13)\\
&\text{subject to}\\
&\forall s_i \in s^d, s_k \in s'^d, \forall a \in \mathcal{A} :\\
&|\sum_{s_j \in s^d} T^{s,d}(s_i,a,s_j)f(s_j,s_k) - \sum_{s_j \in s'^d} f(s_i,s_j)T^{s',d}(s_j,a,s_k)| \leqslant e\\
&f(s_i,s_k) \geqslant 0\\
&\sum_{s_k \in s'^d} f(s_i,s_k) = 1
\end{aligned}$$

The principal steps of our approach are summarized in Algorithm 10, with a high-level view given in the diagram of Figure 6.9. For every state $s \in \mathcal{S} \backslash \mathcal{S}^E$, we create the list s^t

of neighbor states that can be reached from s within t steps (steps 1-6). The distance t is gradually increased until we find a state $s' \in s^t \cap \mathcal{S}^E$ that is locally similar to s (step 7). If $s^t = s^{t-1}$, i.e. all the states that can be reached from s are already contained in s^{t-1}, and no one is locally similar to s (step 8), then we set $\hat{\pi}^E(s,a)$ to a uniform distribution (step 9), and a stopping boolean variable c is set to true (step 21). Otherwise, for each action a, $\hat{\pi}^E(s,a)$ is proportional to the weighted votes for a of the states that are locally similar to s (steps 16 and 23).

Input: An MDP model without reward $(\mathcal{S}, \mathcal{A}, T, \alpha, \gamma)$, a set of demonstration trajectories, an error threshold ϵ, and a similarity distance d;
1 Let $\mathcal{S}^E$ be the set of states contained in the demonstration trajectories;
2 Use the demonstration trajectories to estimate the policy $\hat{\pi}^E$ for the states of $\mathcal{S}^E$;
3 Let s^t be the set of states that can be reached from a state s within t steps, *votes* a vector containing the number of votes per action, and c the stopping condition;
4 **foreach** $s \in \mathcal{S} \backslash \mathcal{S}^E$ **do**
5 $t \leftarrow 0, s^0 \leftarrow \{s\}, votes \leftarrow (0, \ldots, 0), c \leftarrow$ false;
6 **repeat**
7 $t \leftarrow t + 1$;
8 **if** $s^t = s^{t-1}$ **then**
9 $c \leftarrow$ true, $votes \leftarrow (1, \ldots, 1)$;
10 **else**
11 **foreach** $s' \in s^t \cap \mathcal{S}^E$ **do**
12 Let e be the error returned by the linear program (10) on $(\mathcal{M}^{s,d}, \mathcal{M}^{s',d})$;
13 **if** $e \leqslant \epsilon$ **then**
14 $c \leftarrow$ true;
15 **foreach** $a \in \mathcal{A}$ **do**
16 $votes(a) \leftarrow votes(a) + \hat{\pi}^E(s', a)$
17 **end**
18 **end**
19 **end**
20 **end**
21 **until** $c = true$;
22 **foreach** $a \in \mathcal{A}$ **do**
23 $\hat{\pi}^E(s,a) = \frac{votes(a)}{\sum_{a' \in \mathcal{A}} votes(a')}$;
24 **end**
25 **end**
Output: A generalized policy $\hat{\pi}^E$;

Algorithm 10: Apprenticeship Learning via Soft Local Homomorphisms.

The generalized policy $\hat{\pi}^E$ can be either considered as the robot's policy, or used to calculate the feature frequencies $\hat{V}_i(\pi^E)$ for another algorithm, as LPAL.

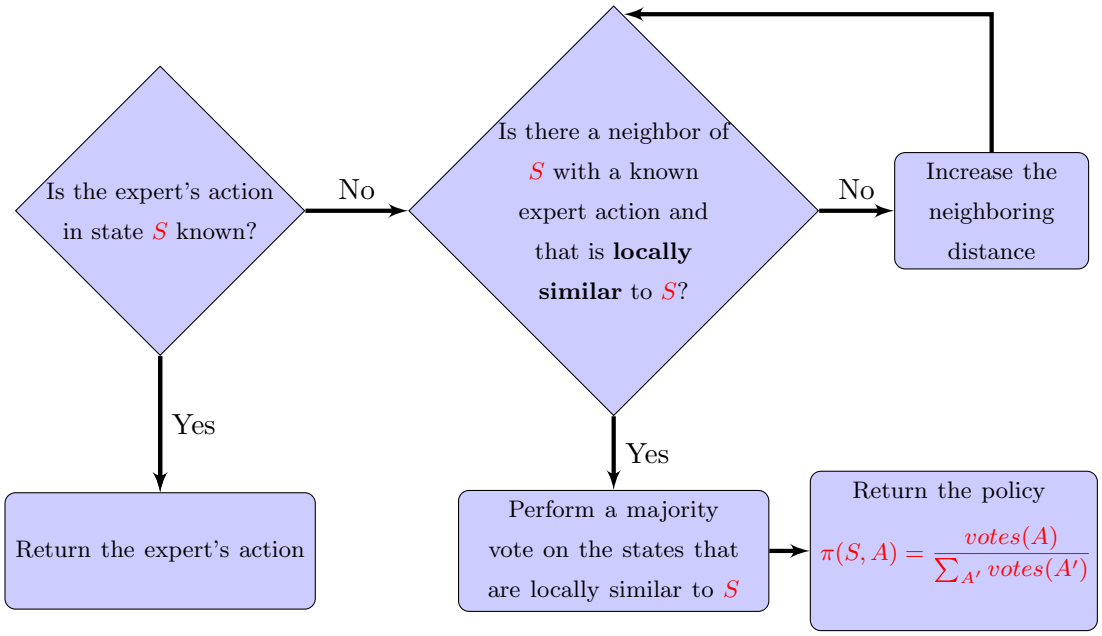


Figure 6.9: Apprenticeship Learning via Soft Local Homomorphisms.

6.5.4 Experiments

To validate our approach, we performed experiments on two simulated navigation domains. The first one is a gridword problem taken from [Abbeel and Ng, 2004]. While this is not meant to be a challenging task, it allows us to compare our approach to other methods of generalizing the expert's policy when the number of demonstrations is small. The second domain corresponds to a racetrack.

6.5.4.1 Gridworld

We consider multiple $x \times x$ girdworld domains, with x taking the following values: 16, 24, 32, and 48. The state of the robot corresponds to its location on the grid, therefore, the dimension $|\mathcal{S}|$ of the state space takes the values 256, 576, 1024, and 2304. The robot has four actions for moving in one of the four directions of the compass, but with a probability of 0.3 the action fails and results is a random move. The initial state corresponds to the position $(0, 0)$, and the discount factor γ is set to 0.99. The gridworld is divided into non-overlapping regions, and the reward function varies depending on the region in which the robot is located. For each region i, there is one feature ϕ_i, where $\phi_i(s)$ indicates whether state s is in region i. The robot knows the features ϕ_i, but not the weights w_i defining the reward function of the expert. The weights w_i are set to 0 with probability 0.9, and to a random value between 0 and 1 with probability 0.1.

The expert's policy π^E corresponds to the optimal deterministic policy found by value iteration. In all our experiments on gridworlds, we used only 10 demonstration trajectories, which is a small number compared to other methods ([Neu and Szepesvári, 2007] for example). The length of the trajectories are 50 for the 16×16 and 24×24 grids, 100 for the 32×32 grid, and 200 for the 48×48 grid.

The robot is trained by using the LPAL algorithm. However, as already mentioned, this algorithm requires the knowledge of the frequencies v_{i,π^E}, which is not the case in our experiments since the demonstration covers only a small number of states. Instead, we used the following methods for learning a generalized policy $\hat{\pi}^E$, and provided the feature frequencies of $\hat{\pi}^E$ to LPAL. Except for Monte Carlo, the frequencies $v_{i,\hat{\pi}^E}$ are calculated by solving the flow Equations 6.4.

Full policy: the complete expert's policy π^E is provided to LPAL.

Soft local homomorphism: The generalized policy $\hat{\pi}^E$ is learned by Algorithm 10, the threshold ϵ is set to 0 and the distance d is set to 1.

Maximum entropy: The generalized policy $\hat{\pi}^E$ is set to a uniform distribution in the states that did not appear in the demonstration.

Euclidian k-NN: The generalized policy $\hat{\pi}^E$ is learned by the k-nearest neighbors algorithm using the euclidian distance measured over the position of the robot. The distance k is gradually increased until encountering at least one state that appears in the demonstration trajectories, as done in Algorithm 10.

Manhattan k-NN: k-NN with a Manhattan distance.

Nonlinear regression: The occupancy measure $\mu_{\hat{\pi}}$ is considered as a linear function of a polynomial kernel defined on the horizontal and vertical coordinates of the robot's position. In other terms, for each state $s = (s_i, s_j)$ we have:

$$\sum_{a \in \mathcal{A}} x^{\mu_\pi}(s,a) = \alpha_0 + \alpha_1 s_i + \alpha_2 s_j + \alpha_3 s_i^2 + \alpha_4 s_j^2 + \alpha_5 s_i s_j + \epsilon(s), \alpha_i \in \mathbb{R}$$

We use a linear program to minimize the sum of errors $\sum_s |\epsilon(s)|$ under Bellman flow constraints (Equation 6.4), while the states that appear in the demonstration are constrained to have the same action as the expert. Finally, $\hat{\pi}^E$ is extracted from μ_π according to Equation 6.11.

Monte Carlo: This is the method used in the literature, the frequencies $\hat{v_{i,\pi^E}}$ are estimated directly from the trajectories, according to Equation 6.9.

Table 6.1 shows the average reward per step of the robot's policy, averaged over 1000 independent trials of the same length as the demonstration trajectories. Our first observation is that LPAL algorithm learned policies are just as good as the expert's policy when the feature frequencies are calculated by using the expert's full policy, but remarkably failed to do so when

Gridworld Size	Number of Regions	**Expert policy**	**Full policy**	**Soft local homomorphism**	**Monte Carlo**	**Maximum Entropy**	**Euclidian k-NN**	**Regression**	**Manhattan k-NN**
16×16	16	0.4672	0.4692	**0.4663**	0.0380	0.3825	0.4672	0.4370	0.4635
	64	0.5281	0.5310	**0.5210**	0.0255	0.4607	0.5218	0.5038	0.5198
	256	0.3988	0.4029	**0.4053**	0.0555	0.3672	0.3915	0.3180	0.4062
24×24	64	0.6407	0.6386	**0.6394**	0.0149	0.5855	0.6394	0.5530	0.6334
	144	0.5916	0.5892	**0.5827**	0.0400	0.5206	0.5890	0.5069	0.5876
	576	0.3568	0.3553	**0.3489**	0.0439	0.2814	0.3114	0.2701	0.2814
32×32	64	0.6204	0.6179	**0.6188**	0.0145	0.5694	0.6198	0.5735	0.6177
	256	0.5773	0.5779	**0.5726**	0.0556	0.5118	0.5730	0.4372	0.5729
	1024	0.4756	0.4778	**0.4751**	0.0394	0.4482	0.4751	0.4090	0.4706
48×48	64	0.6751	0.6751	**0.6732**	0.0141	0.6234	0.6732	0.6052	0.6653
	256	0.6992	0.7006	**0.6909**	0.0603	0.6587	0.6999	0.6437	0.6997
	2304	0.4950	0.4972	**0.4876**	0.0528	0.4640	0.4913	0.4437	0.4330

Table 6.1: Gridworld results.

the frequencies are learned from the demonstration by using a Monte Carlo estimator. This is due to the fact that we used a very small number of demonstrations compared to the size of these problems. Second, LPAL returns better policies when the frequencies are analytically calculated by using maximum entropy technique than when they are estimated by Monte Carlo. This is because Monte Carlo estimates the frequencies for a finite horizon. Given that the expert's actions cannot be explained by only the vertical and horizontal coordinates, the regression method also failed to outperform the maximum entropy method. We also remark that Euclidian and Manhattan k-NN performed similarly due to the similarity between these two distances in the context of flat grids. They both succeeded to learn policies with values close to the optimal value. Finally, our approach performed just as k-NN in this experiment. In fact, since there are no obstacles on the grid, most of the states are locally similar, even for neighbors of a long distance.

6.5.4.2 Racetrack

We implemented a simplified car race simulator, the corresponding racetracks are showed in Figure 6.10. The states correspond to the position of the car in the racetrack and its speed. We considered two discretized speeds, low and high, in each direction of the vertical and horizontal axis, in addition to the zero speed in each axis, leading to a total of 25 possible combinations of speeds, 5900 states for racetrack (a), and 5100 states for racetrack (b). The car controller can accelerate or decelerate in each axis, or do nothing. The controller cannot however combine a horizontal and a vertical action, the number of actions then is five. When the speed is low, acceleration\deceleration actions succeed with probability 0.9, and fail with probability 0.1, leaving the speed unchanged. The success probability of the actions falls

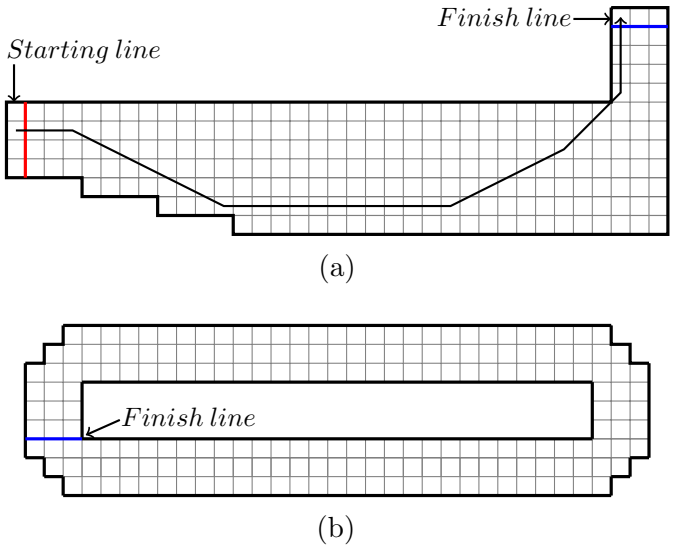


Figure 6.10: Racetrack configurations and a demonstration of the expert's policy. In racetrack (b), the car starts at a random position.

down to 0.5 when the speed is high, making the vehicle harder to control. When the vehicle tries to move off-road, it remains in the same position and its speed is reset to zero. The car controller receives a reward of 5 for each step except for off-roads, where it receives 0, and for reaching the finish line, where the reward is 200. A discount factor of 0.99 is used in order to favor shorter trajectories.

In this experiment, we compared only the methods that performed well in the gridworld domaine, which are LPAL with a full policy, LPAL with soft local homomorphisms, and LPAL with k-NN using the Manhattan distance, since the euclidian distance does not take into account the speed of the vehicle. We also compared k-NN and soft local homomorphisms without LPAL.

Figures 6.11, 6.12, 6.13, 6.14, 6.15, and 6.16 show the average reward per step of the robot's policy, the average number of off-roads per step, and the average number of steps before reaching the finish line, as a function of the number of trajectories in the demonstration. For racetrack (a), the car always starts from the same initial position, and the length of each demonstration trajectory is 20. For racetrack (b) however, the car starts at a random position, and the length of each trajectory is 40. The results are averaged over 1000 independent trials of length 30 for racetrack (a) and 50 for racetrack (b). Contrary to the gridworld experiments, LPAL achieved good performances only when the features are calculated by using the complete policy of the expert. For clarity, we removed the results of LPAL with k-NN and with soft local homomorphisms, which were below the performances of the other methods.

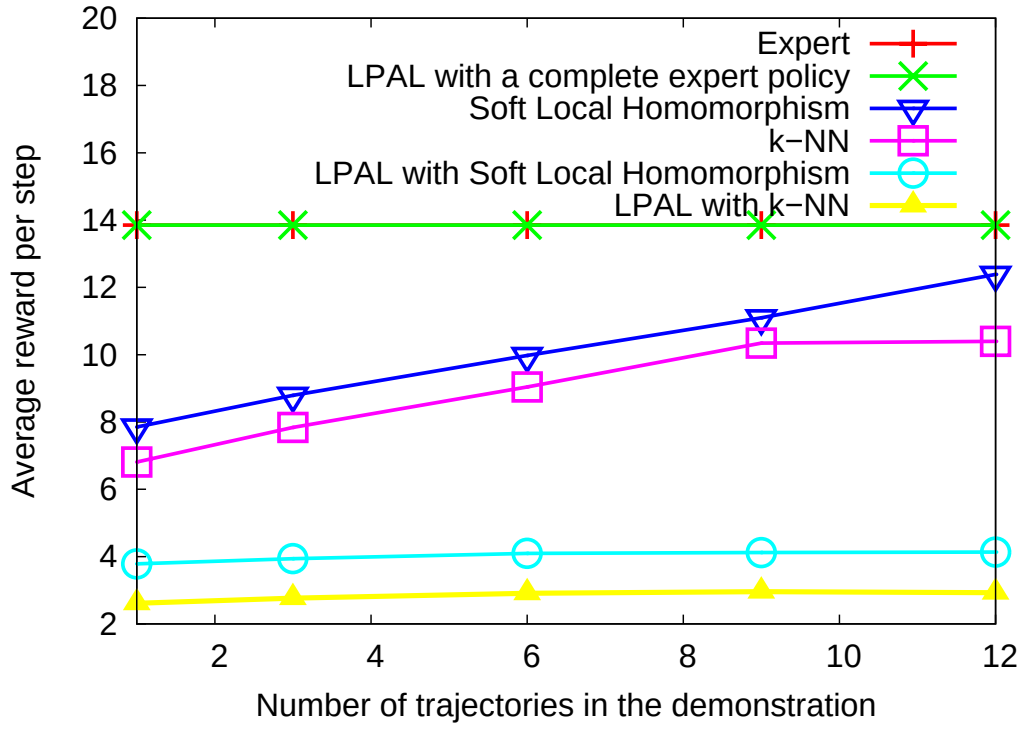


Figure 6.11: Average reward per step in racetrack (a).

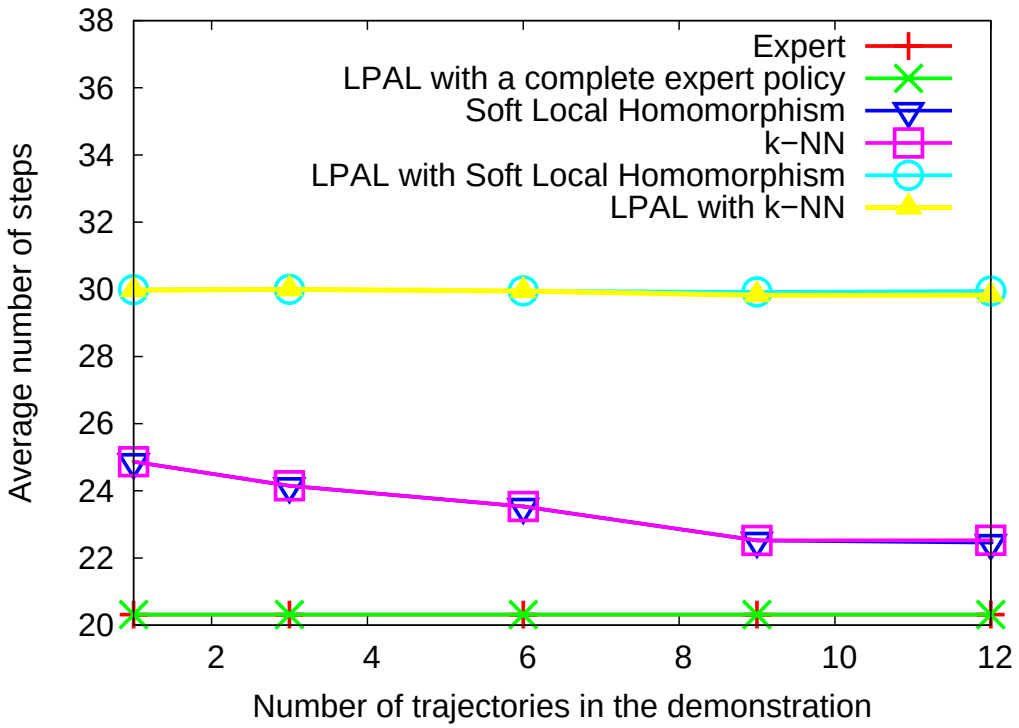


Figure 6.12: Average number of steps before reaching the finish line in racetrack (a).

As expected, we notice the significant improvement of our algorithm over k-NN in terms of average reward, average number of off-roads per step, and average number of steps to the finish line. This is due to the fact that, contrary to k-NN, homomorphisms do take into account the dynamics of the system. For example, when the car faces an obstacle, the local MDP defined around its current position is similar to all the local MDPs defined around the positions of facing an obstacle, the deceleration actions can then be efficiently transferred, as depicted in the example of Figure 6.17.

6.5.5 Discussion

The main question of apprenticeship learning is how to generalize the expert's policy to states that have not been encountered during the demonstration. Previous works have attempted to solve this problem by considering the state as a vector of features of a smaller dimension, and classifying the states accordingly. However, an expert's policy is a much more complicated function than it can be explained by only immediate features.

Inspired by the intuition that the states that are locally similar have the same optimal action in general, we introduced a new technique for generalizing the expert's policy based on soft local homomorphisms. Unlike other methods, our approach considers the long term dependencies between different states, rather than just immediate features. We also showed

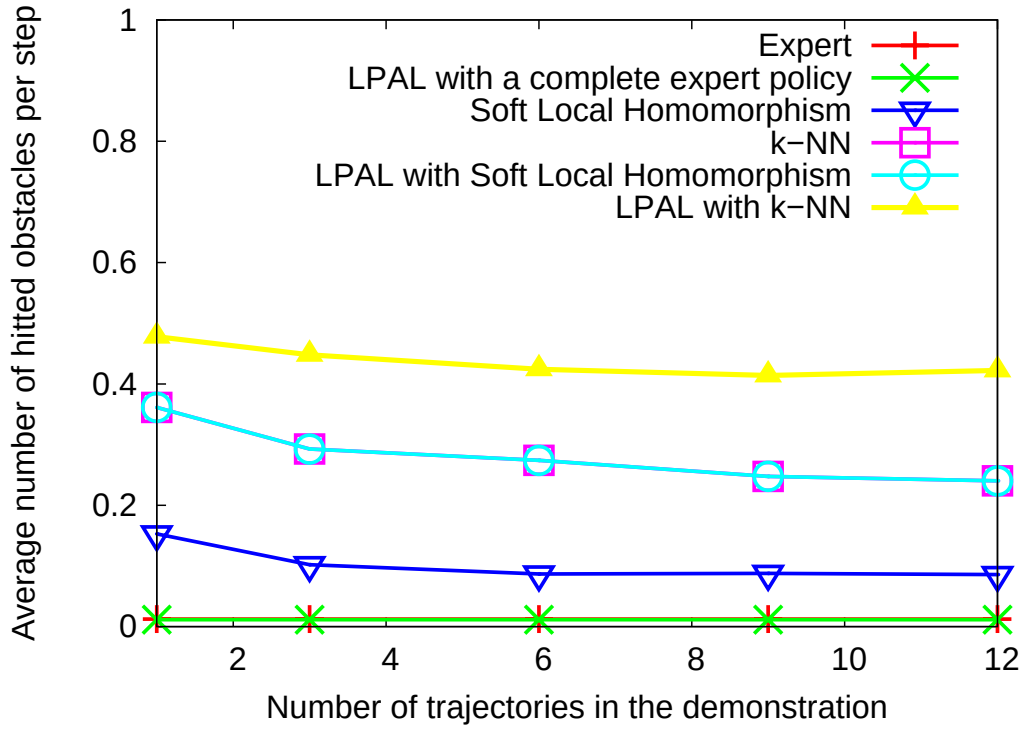


Figure 6.13: Average number of off-roads per step in racetrack (a).

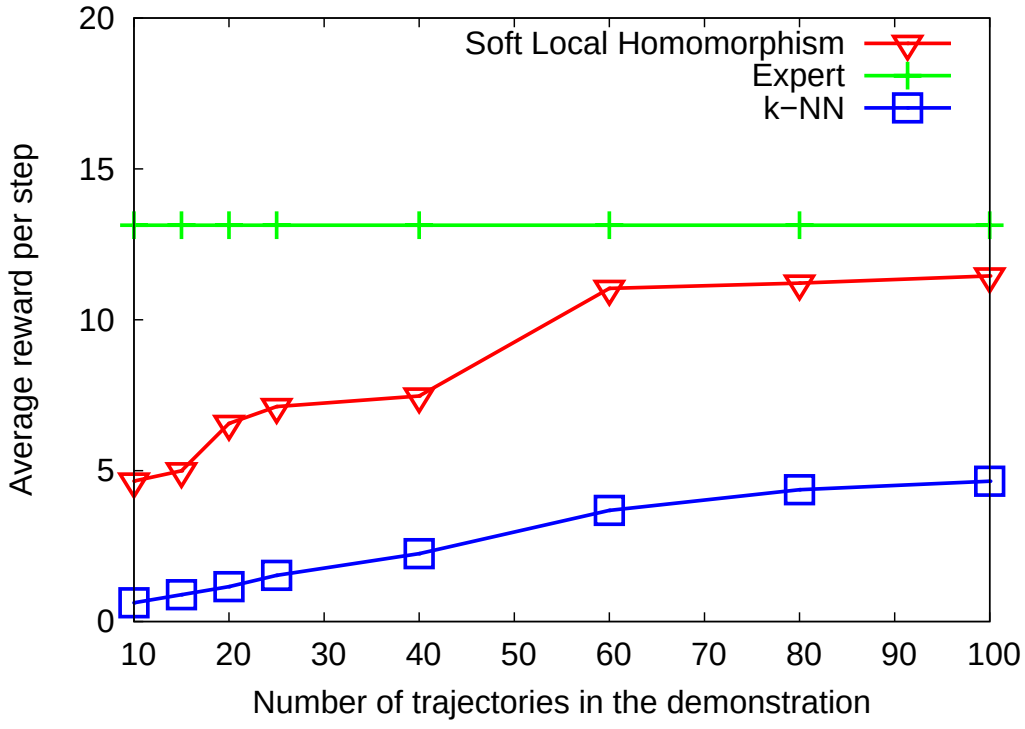


Figure 6.14: Average reward per step in racetrack (b).

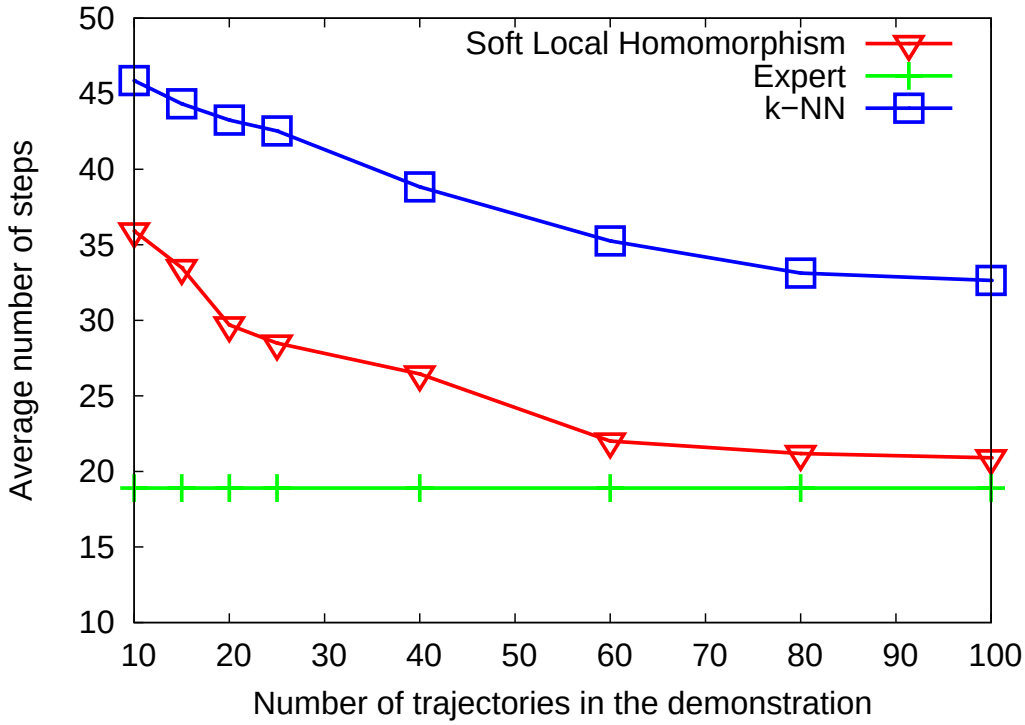


Figure 6.15: Average number of steps before reaching the finish line in racetrack (b).

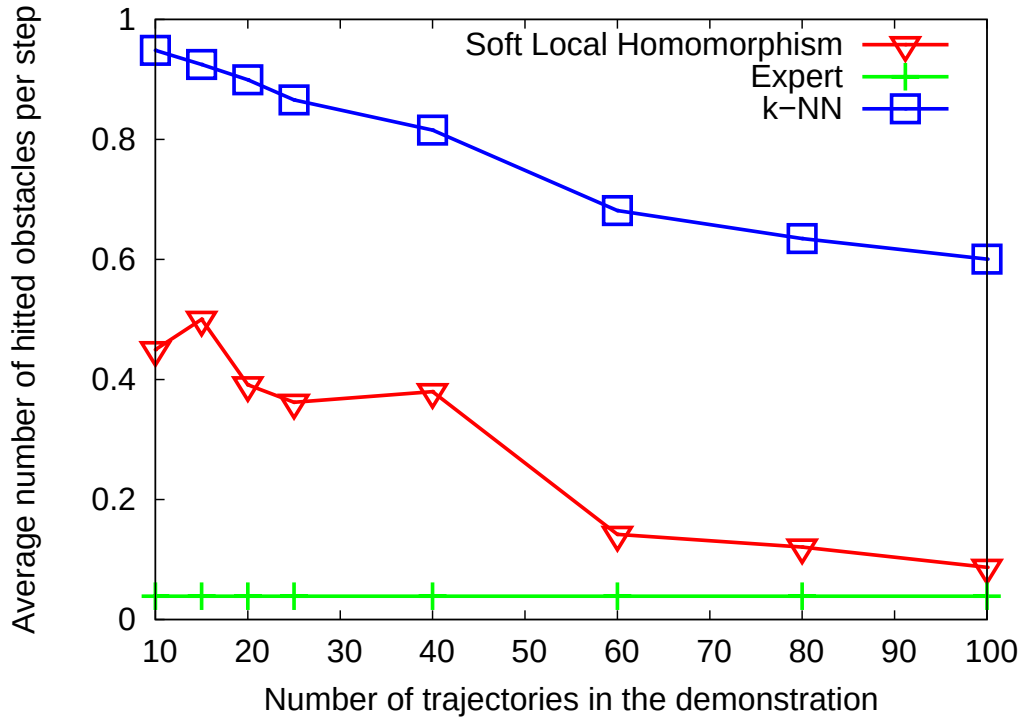


Figure 6.16: Average number of off-roads per step in racetrack (b).

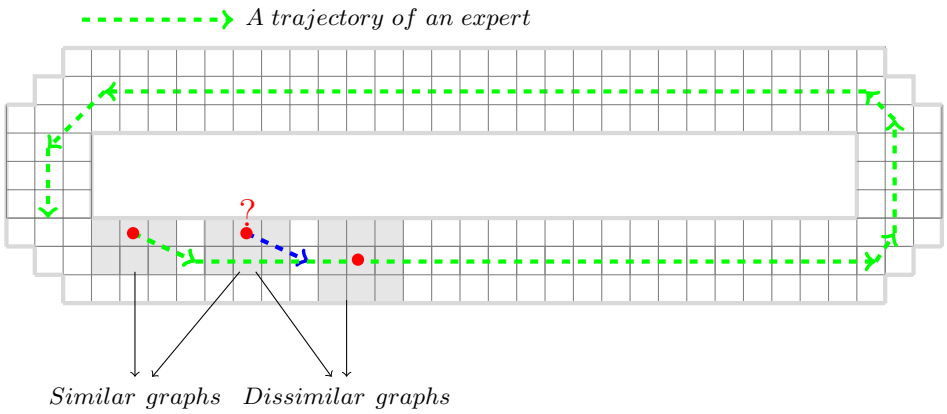


Figure 6.17: An example of two states that are locally similar by homomorphism.

that using homomorphisms leads to a significant improvement in the quality of the learned policies.

However, our approach lacks a theoretical guarantee beyond the intuition. In fact, control policies are local and reactive in most states, such as avoiding obstacles during a navigation task, but there are always some critic states where the optimal actions cannot be explained by only the local dynamics of the system. Distinguishing between these states is crucial for providing a theoretical guarantee of our approach in a future work.

In the next section, we introduce another solution for the same problem of generalizing the expert's policy. Our solution is based on using the learned reward function in order to generalize the expert's policy, which will be used for learning an improved reward function.

6.6 Bootstrapping The Expert's Examples

6.6.1 Overview

In this section, we consider the same problem of generalizing the expert's demonstration that was discussed in the previous section. However, we propose a different approach for solving it, based on learning a reward function by using an IRL algorithm with the partial examples provided by the expert. The learned rewards are used for finding an optimal policy, which is then fed back to the IRL algorithm. This operation is repeated until a locally optimal solution is found. Empirical results on gridworlds and car racing problems show that our approach is able to learn good policies from a small number of demonstrations.

6.6.2 Reward Loss in Maximum Margin Planning

In the remainder of this chapter, we will adopt Maximum Margin Planning (MMP) as framework for IRL, and build our methods on this. This choice is motivated by the good empirical results obtained by this algorithm compared to others, in addition to its simplicity.

To show the effect of the frequency error on the quality of learned policies, we provide in this section an analysis of the distance between the vector of reward weights $\hat{w}$ returned by MMP with estimated frequencies $\hat{v}_{\pi^E} = F\hat{\mu}_{\pi^E}$, calculated from the examples by using Equation 6.9, and the vector w^E returned by MMP with accurate frequencies $v_{\pi^E} = F\mu_{\pi^E}$, calculated by using Equations 6.4 with the full policy π^E. We will use the following notations: $\Delta v_\pi = \hat{v}_{\pi^E} - v_{\pi^E}$, $\Delta w = \hat{w} - w^E$, and $V_l(w) = \max_{\mu \in \mathcal{G}}(w^T F + l)\mu$, and consider $q = 1$. The following theorem shows how the reward error Δw is related to the frequency error Δv_π.

Theorem 3. *Let $\epsilon \in \mathbb{R}^+$, if $\forall w \in \mathbb{R}^k$, such that $\| w - \hat{w} \|_2 = \epsilon$, the following condition is verified:*

$$\| \Delta v_\pi \|_2 < \frac{V_l(w) - V_l(\hat{w}) + (\hat{w} - w)^T \hat{v}_{\pi^E} + \frac{\lambda}{2}(\| w \|_2 - \| \hat{w} \|_2)}{\epsilon}$$

then $\| \Delta w \|_2 \leqslant \epsilon$.

Proof. The condition stated in the theorem above implies:

$$\begin{aligned}
&\| \hat{w} - w \|_2 \| \Delta v_\pi \|_2 < V_l(w) - V_l(\hat{w}) + (\hat{w} - w)^T \hat{v}_{\pi^E} + \frac{\lambda(\| w \|_2 - \| \hat{w} \|_2)}{2} \\
&\Rightarrow (\hat{w} - w)^T \Delta v_\pi < V_l(w) - V_l(\hat{w}) + (\hat{w} - w)^T \hat{v}_{\pi^E} + \frac{\lambda(\| w \|_2 - \| \hat{w} \|_2)}{2} \text{ (Hölder)} \\
&\Rightarrow V_l(\hat{w}) - \left(\hat{w}^T v_{\pi^E} - \frac{\lambda}{2} \| \hat{w} \|_2\right) < V_l(w) - \left(w^T v_{\pi^E} - \frac{\lambda}{2} \| w \|_2\right)
\end{aligned}$$

In other terms, the point $(\hat{w}^T v_{\pi^E} - \frac{\lambda}{2} \| \hat{w} \|_2)$ is closer to the surface V_l than any other point $(w^T v_{\pi^E} - \frac{\lambda}{2} \| w \|_2)$, where w is a point on the sphere centered around $\hat{w}$ with a radius of ϵ. Since the function V_l is convex and $({w^E}^T v_{\pi^E} - \frac{\lambda}{2} \| w^E \|_2)$ is by definition the closest point to the surface V_l, then w^E should be inside the ball centered around $\hat{w}$ with a radius of ϵ. Therefore, $\| w^E - \hat{w} \|_2 \leqslant \epsilon$ and thus $\| \Delta w \|_2 \leqslant \epsilon$. □ □

Consequently, the reward loss $\| \Delta w \|_2$ approaches zero as the error of the estimated feature frequencies $\| \Delta v_\pi \|_2$ approaches zero. A simpler bound can be easily derived given admissible heuristics of V_l.

Corollary: Let $\underline{V_l}$ and $\overline{V_l}$ be respectively a lower and an upper bound on V_l, then Theorem 3 holds if $V_l(w) - V_l(\hat{w})$ is replaced by $\underline{V_l}(w) - \overline{V_l}(\hat{w})$.

Figure 6.18 illustrates the divergence from the optimal reward weight w^E when approximate frequencies are used. The error is not a continuous function of Δv_π when the cost

function is not regularized, because the vector returned by MMP is always a fringe point. In general, the error is proportional to the subgradient of the function $V_l - v_{\pi^E}$ at the fringe point w^E.

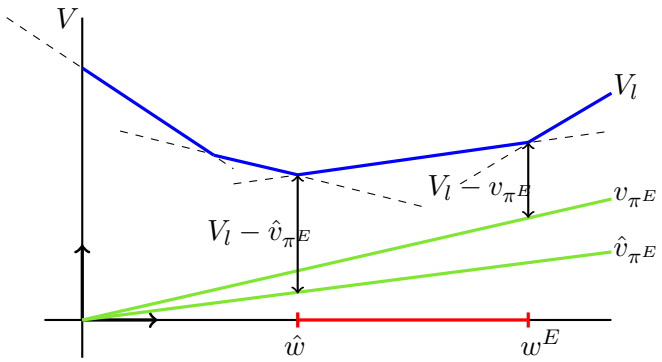


Figure 6.18: Reward loss in MMP with approximate frequencies $\hat{v}_{\pi^E}$.

6.6.3 Efficient Frequency Approximation

The feature frequency error Δv_π can be significantly reduced by using the known transition function for calculating $\hat{v}_{i,\pi^E}$ instead of the Monte Carlo estimator (Equation 6.9). However, this cannot be done unless the complete expert's policy π^E is provided.

In Maximum Margin Planning, as in any iterative IRL algorithm, the reward weights are learned incrementally. These algorithms start with initial weights, and then iteratively improve them until a local optimal solution is reached. One can take advantage of this property of iterative algorithms and use the intermediate reward weights for both generalizing the provided examples, and for approximating the expert's full policy. Assuming the expert is optimal, the policy π^E can be approximated by another one, which selects the same actions as the expert whenever the expert's action is known, and the optimal actions, according to the current reward weight w, when the expert's action is unknown, as shown in the example of Figure 6.19.

We will refer to this approach, where the expert's policy is calculated from the provided examples and the current reward function, as Improved Maximum Margin Planning (I-MMP). The cost function of I-MMP is given by:

$$c_q(w) = \left(\max_{\mu \in \mathcal{G}} (w^T F + l)\mu - \max_{\mu \in \mathcal{G}_{\pi^E}} w^T F \mu \right)^q + \frac{\lambda}{2} \parallel w \parallel^2 \tag{6.14}$$

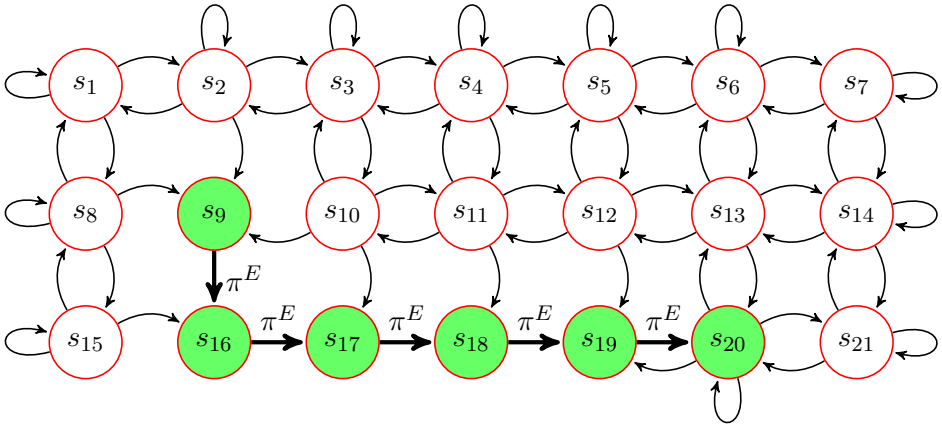


Figure 6.19: A modified Markov Decision Process. The states (in green) where the expert's action is known form a Markov chain, whereas the other states form a Markov Decision Process.

where $\mathcal{G}_{\pi^E}$ is the set of occupancy vectors μ_π, subject to the constraints:

$$\mu_\pi(s) = \alpha(s) + \gamma \sum_{s' \in \mathcal{S}^E} \mu_\pi(s') \sum_{a \in \mathcal{A}} \pi^E(s', a) T^a(s', s) + \gamma \sum_{s' \in \mathcal{S} \setminus \mathcal{S}^E} \sum_{a \in \mathcal{A}} \mu_\pi(s', a) T^a(s', s) \quad (6.15)$$

$$\sum_{a \in \mathcal{A}} \mu_\pi(s, a) = \mu_\pi(s)$$

$$\mu_\pi(s, a) \geqslant 0$$

where $\mathcal{S}^E$ is the set of states where the expert's policy is known.

Notice the difference between Equation 6.14 and the original one, Equation 6.7, which we rewrite here:

$$c_q(w) = \Big(\max_{\mu \in \mathcal{G}} (w^T F + l)\mu - w^T F \mu_{\pi^E} \Big)^q + \frac{\lambda}{2} \parallel w \parallel^2$$

The only difference is that, in the original cost function, μ_{π^E} is empirically estimated from the examples, whereas in the new cost function, μ_{π^E} is replaced by the occupancy vector $\mu \in \mathcal{G}_{\pi^E}$ of the optimal policy according to the reward w, that selects the same actions as the expert's policy π^E in the demonstrated states.

Unfortunately, the new cost function (Equation 6.14) is not necessarily convex. In fact, it corresponds to a margin between two convex functions: the value of the generalized expert's policy $\max_{\mu \in \mathcal{G}_{\pi^E}} w^T F \mu$ and the value of the best alternative policy $\max_{\mu \in \mathcal{G}} (w^T F + l)\mu$. Yet, a local optimal solution of this modified cost function can be found by using the same sub-gradient as in Equation 6.8, and putting $\mu^E = \arg\max_{\mu \in \mathcal{G}_{\pi^E}} w^T F \mu$. In practice, as we will show in the experimental analysis, the solution returned by the improved MMP outperforms the solution of MMP where the expert's frequency is calculated without taking into account the known transition probabilities. This improvement is particularly pronounced in highly stochastic environments.

The empirical analysis of this approach will be presented in Section 6.7.3, along with the analysis of the next approach that we propose for improving IRL algorithms, which is based on predictive reward representations.

6.7 Predictive Reward Representations

6.7.1 Motivation

Figure 6.20 shows a car driving domain that can be represented as an MDP. The actions correspond to turning left or right, accelerating or decelerating, or doing nothing. The states corresponds to the direction of the car (west-east: $\Rightarrow$ or east-west: $\Leftarrow$), and its position on one of the three lanes (north: 1, middle: 2, or south: 3) or on the grass. The state may also include other information, such as the speed of the car and the positions of other cars.

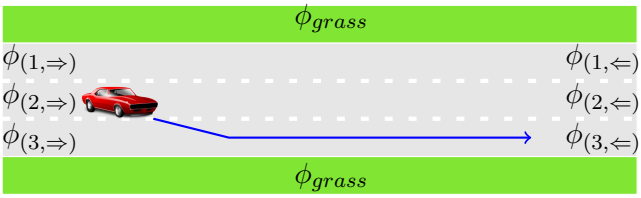


Figure 6.20: A car driving simulation and the corresponding features.

Now assume that we want to train an agent to drive like an expert, and one of the expert's preferences is to drive on the right of the road. In this case, the reward is simply a linear function of the binary features $\phi_{(lane,direction)}$ corresponding to the driver's local observations. Figure 6.20 also shows the trajectory of an expert's demonstration, where the right of the road is the south lane. From that demonstration, the learner agent will associate a high weight to the feature $\phi_{(3,\Rightarrow)}$ and no weight to the others, whereas it should associate a high weight to the feature $\phi_{(1,\Leftarrow)}$ too. This failure to generalize the expert's demonstration from one direction to another is due to the fact that the feature $\phi_{(1,\Leftarrow)}$ is sparse (too specific). Consequently, the empirical frequency of feature $\phi_{(1,\Leftarrow)}$, estimated from this demonstration, is null.

To solve this problem, the learner agent may use a simpler (more general) hypothesis to explain the expert's behavior. For instance, the latent feature $\phi_{turn\ right,grass}$, which is the probability of observing the grass after executing the action "turn right", is non null if and only if one of the features $\phi_{(1,\Leftarrow)}$ and $\phi_{(3,\Rightarrow)}$ is true. The empirical frequency of this latent variable, estimated from the same example, is non null. Therefore, the learner agent will associate a high weight to this latent feature, and will be able to generalize the example to the other direction. Therefore, the reward function will be presented by the predictions of the values of the different features in the future. These latter features will be called *extended* (or *latent*) features.

6.7.2 Extended Features

As shown in the previous example, one way to reduce the frequency error is to consider latent features $\tilde{\phi}_i$, which take non null values in more states than the base features ϕ_i. We use $\tilde{F}$ to denote the $\tilde{k}$ by $|\mathcal{S}||\mathcal{A}|$ latent feature matrix, where $\tilde{F}(i,(s,a)) = \tilde{\phi}_i(s,a)$ and $\tilde{k}$ is the number of latent features. The reward vector must remain in the span of the latent features, i.e. $R = w^T\tilde{F}$. Ideally, one should find a minimal set of latent features that satisfy this condition. However, this cannot be done since the reward vector is unknown. Instead, if we can ensure that all the basic features ϕ_i are in the span of the latent features, i.e. $F = m^T\tilde{F}$, then the reward vector will necessarily be in the span of $\tilde{F}$.

Furthermore, to reduce the sparsity of the matrix $\tilde{F}$, we propose to use latent features that are sufficient to predict the expected value of the basic features after an arbitrary sequence of actions. We will refer to this type of features as extended features (or tests on features). The problem of finding a minimal set of extended features can be solved by finding $\tilde{k}$ by $|\mathcal{S}|$ matrix $\tilde{F} = [\tilde{\phi}_i^{a_1}, \ldots, \tilde{\phi}_i^{a_n}]^T$, $\tilde{k}$ by $\tilde{k}$ matrices $\tilde{T}^a$, and weight vectors $m, \tilde{m}$ that satisfy the quadratic constraints: $f_i^a = m_a^T\tilde{F}$ and $T^a\tilde{F}^T = \tilde{F}^T\tilde{T}^a, \forall a \in \mathcal{A}$, where $\phi_i^a(s) = \phi_i(s,a)$.

As with Predictive State Representations (Section 3.6), a fast solution to this problem is given by using the Krylov method, where the vectors of the latent features correspond to the images of the basic features ϕ_i under the product of transition matrices $T^a, a \in \mathcal{A}$. The set Q of vectors $\tilde{\phi}_i^a$ is iteratively constructed as follows: Q^0 is initialized with the basic features ϕ_i that are linearly independent, and for $t > 0$, $Q^{t+1} = Q^t \cup \{T^a\tilde{\phi} | a \in \mathcal{A}, \tilde{\phi} \in Q^t\}$. At the end of each iteration t, only independent vectors are kept in Q^t. This process stops when $Q^{t+1} = Q^t$, notice that this condition is satisfied after at most $|\mathcal{S}|$ expansion steps.

This method can be improved by keeping at the end of each step only features with a high information gain. Informally, the information gain of a feature $\tilde{\phi}_i$ is inversely proportional to the margin $\hat{w}_i^{max} - \hat{w}_i^{min}$, where $\hat{w}_i^{max}$ and $\hat{w}_i^{min}$ are respectively the maximum and minimum values of $\hat{w}_i$, and $\hat{w}$ is the vector that minimizes the cost function. This margin is loosely related to the subgradient with respect to w_i at the hinge point $\hat{w}$.

6.7.3 Experimental Results

To validate our approaches, we experimented on the two simulated navigation problems described in Section 6.5.4. We report here the results of these experiments, where it is refereed to MMP with Bootstrapping as *Improved MMP* or I-MMP, and to MMP with latent extended features as L-MMP.

6.7.3.1 Gridworld

Similarly to the experiments in Section 6.5.4, we used only 10 demonstration trajectories. Table 6.2 shows the average reward per step of the agent's policy, averaged over 10^3 independent trials of the same length as the demonstration trajectories. Our first observation is that Improved MMP algorithm (I-MMP) learned policies just as good as the expert's policy, while MMP remarkably failed to collect any reward. This is due to the fact that we used a very small number of demonstrations compared to the size of these problems, and the frequencies in MMP are learned from the demonstration by using a Monte Carlo estimator. This problem is not specific to MMP, and any other algorithm using the same estimation method would produce the same results. Second, I-MMP with latent features (L-MMP) performed almost the same as I-MMP without latent features. This result is not surprising, given that most of the features are null and the demonstration covered all the features with high values. Finally, we remark that k-NN performed as an expert in this experiment. In fact, since there are no obstacles on the grid, neighboring states often have similar optimal actions.

Size	Features	**Expert**	**k-NN**	**MMP**	**I-MMP**	**L-MMP**
16×16	16	0.4672	0.4635	0.0000	0.4678	**0.4424**
16×16	64	0.5281	0.5198	0.0000	0.5252	**0.5275**
16×16	256	0.3988	0.4062	0.0537	0.3828	**0.3879**
24×24	64	0.5210	0.6334	0.0000	0.5217	**0.4929**
24×24	144	0.5916	0.5876	0.0122	0.5252	**0.5270**
24×24	576	0.0849	0.2814	0.0974	0.0514	**0.1380**

Table 6.2: Gridworld average reward results.

6.7.3.2 Racetrack

We implemented a simplified car race simulator, the corresponding racetracks are showed in Figures 6.10 and 6.21. The states correspond to the position of the car in the racetrack and its speed. We considered two discretized speeds, low and high, in each direction of the vertical and horizontal axis, in addition to the zero speed in each axis, leading to a total of 25 possible combinations of speeds, 5900 states for racetrack (a), 5100 states for racetrack (b), and 10^4 states for racetrack (c). The controller can accelerate or decelerate in each axis, or do nothing. The controller cannot however combine a horizontal and a vertical action, the number of actions then is five. When the speed is low, acceleration\deceleration actions succeed with probability 0.9, and fail with probability 0.1, leaving the speed unchanged. The success probability falls down to 0.5 when the speed is high, making the vehicle harder to control. When the vehicle tries to move off-road, it remains in the same position and its speed

is reset to zero. In racetracks (a) and (b), the controller receives a reward of 5 for each step except for off-roads, where it receives 0, and for reaching the finish line, where the reward is 200. In racetrack (c), the controller receives a reward of 0 for each step except for reaching the finish line, where it receives 10, and for driving on the right side of the road, where it receives 1, and for driving on the grass, where it receives a penalty of -1. For the three racetrack, a discount factor of 0.99 is used in order to favor shorter trajectories.

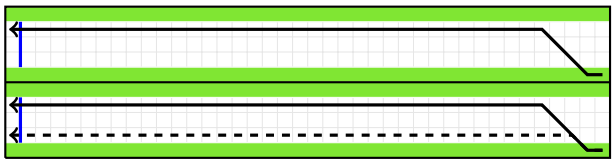


Figure 6.21: Racetrack (c). The trajectories of the expert cover only the upper half of the road. In the lower half, the policy learned using extended features was able to keep the car on the right side, contrary to the policy learned using basic features (dashed line).

For racetrack (a), the car always starts from the same initial position, and the length of each demonstration trajectory is 20. For racetrack (b), the car starts at a random position, and the length of each trajectory is 40. For racetrack (c), the car starts at the same initial position during the demonstration, and at a different position, located at the lower half of the road, during the testing. The length of each trajectory is 40. The results are averaged over 10^3 independent trials of length 30 for racetrack (a) and 50 for racetracks (b) and (c). The basic features correspond to the finish line and driving off-road, in addition to the index of the lane (from 1 to 6) and the grass for the racetrack (b).

Figures 6.22-6.30 show the average reward per step of the controller, the average number of off-roads per step, and the average number of steps before reaching the finish line, as a function of the number of trajectories in the demonstration. We first notice that k-NN performed poorly, this is principally caused by the effect of driving off-road on both the cumulated reward and the speed of the car. In this context, neighbor states do not necessarily share the same optimal action. Contrary to the gridworld experiments, MMP achieved good performances on racetrack (a). In fact, by fixing the initial state, the demonstration covers most of the reachable states, and the feature frequencies are accurately estimated from the demonstration. In racetrack (b) however, MMP was unable to learn a good policy because all the states were reachable from the initial distribution.

As expected, we notice the significant improvement of L-MMP (I-MMP using latent features) in racetrack (c) (Figure 6.31). In fact, without using latent features, I-MMP associates a high reward for reaching the finish line, for avoiding off-roads and avoiding driving on the grass, and driving on lane 1 (the upper lane). Since the feature associated to lane 1 is absent in the lower half of the road, the controller has no preference among the three possible lanes, and rather chooses the shortest path, whereas with latent features, I-MMP associates a high weight to the latent feature (action: Accelerate on the y axis, base feature: grass), this extended feature is present in both the upper and lower roads (Figure 6.21).

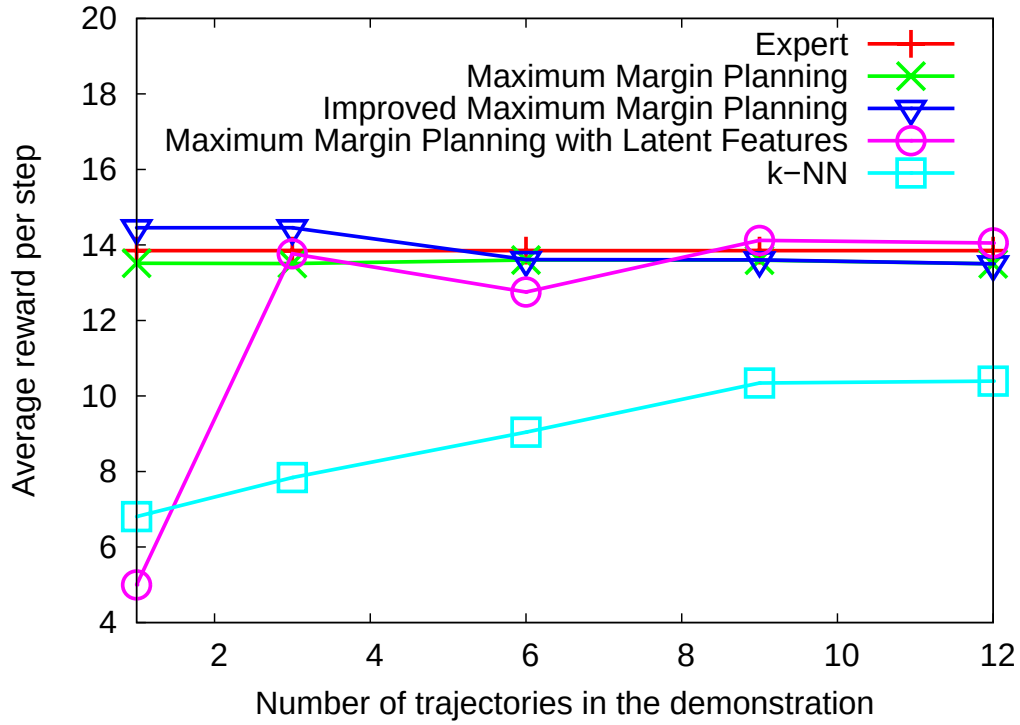


Figure 6.22: Average reward per step in racetrack (a).

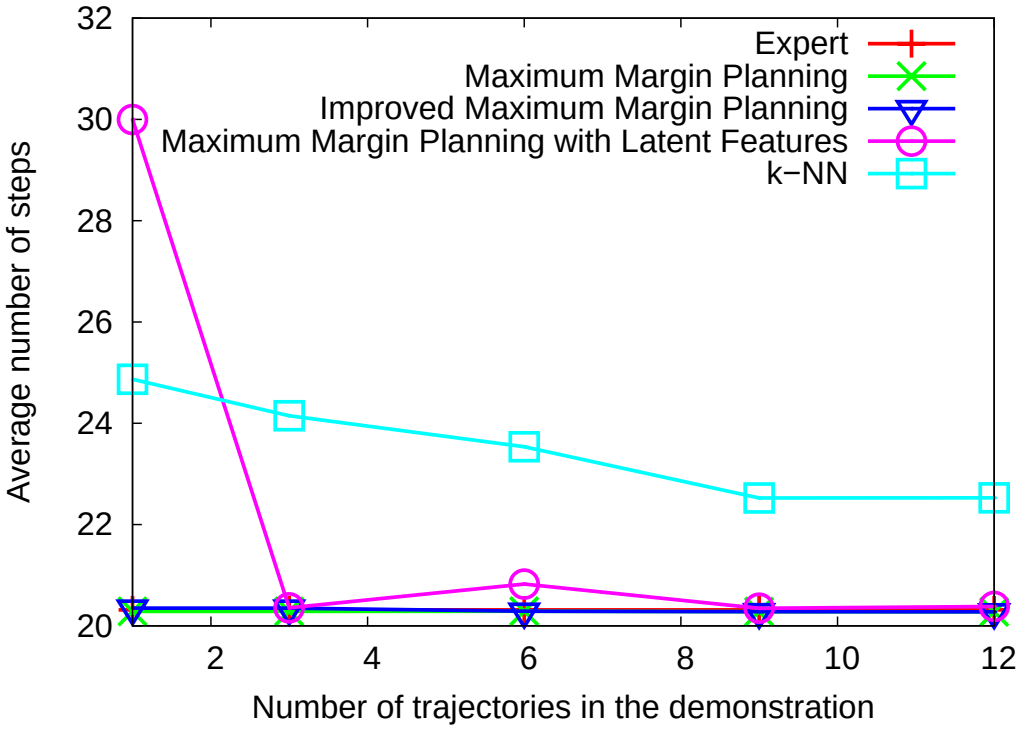


Figure 6.23: Average number of steps before reaching the finish line in racetrack (a).

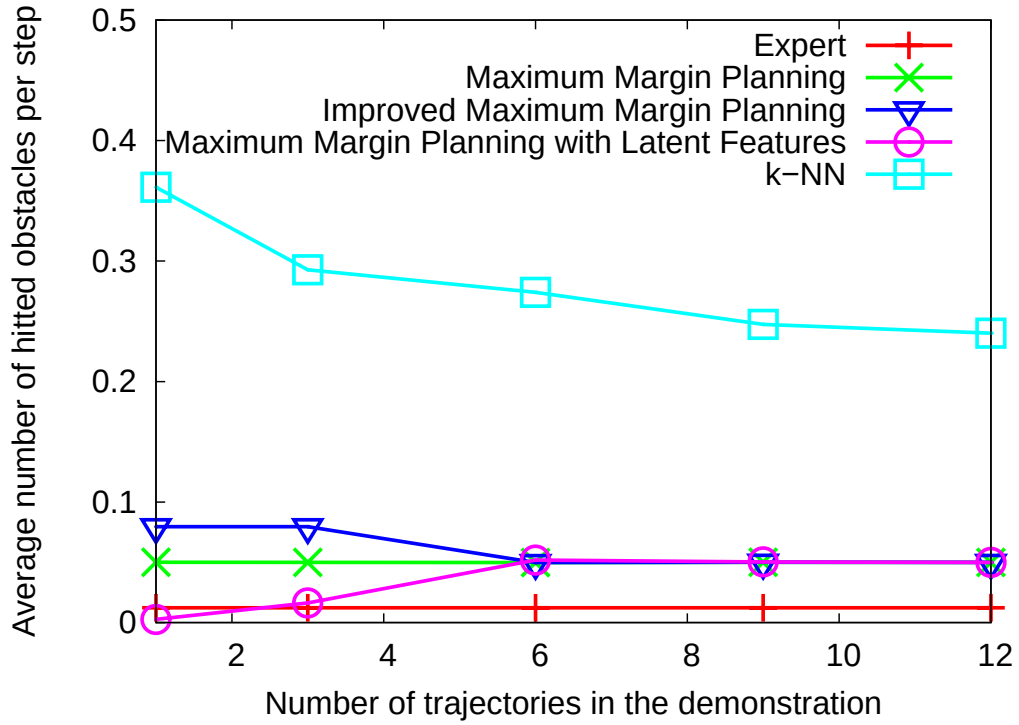


Figure 6.24: Average number of off-roads per step in racetrack (a).

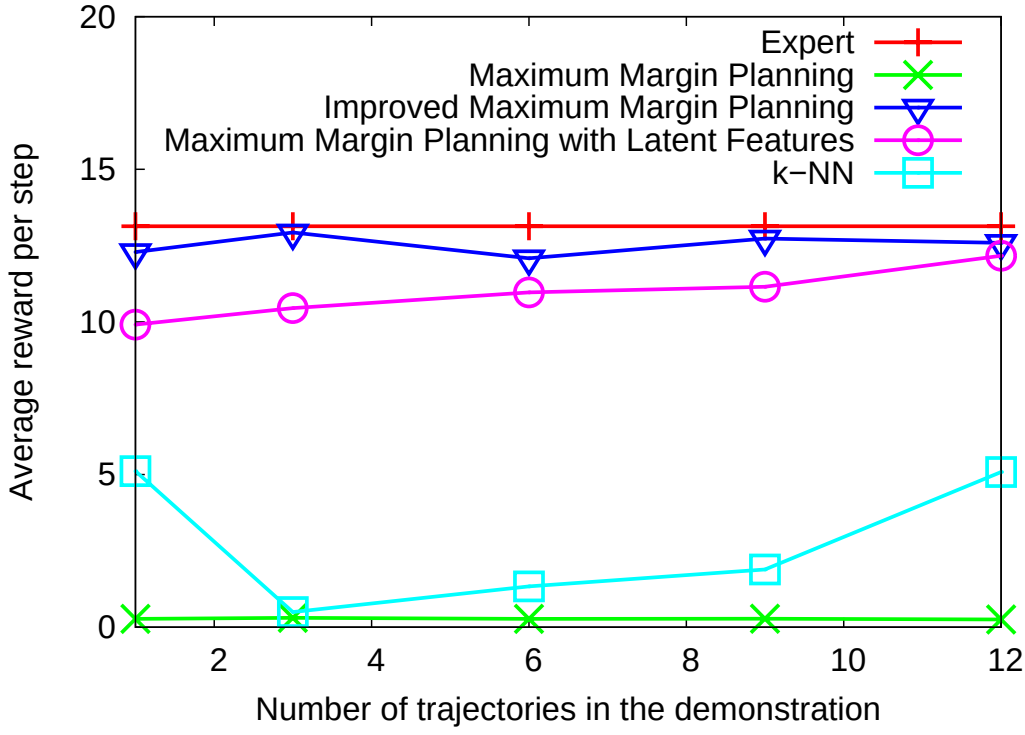


Figure 6.25: Average reward per step in racetrack (b).

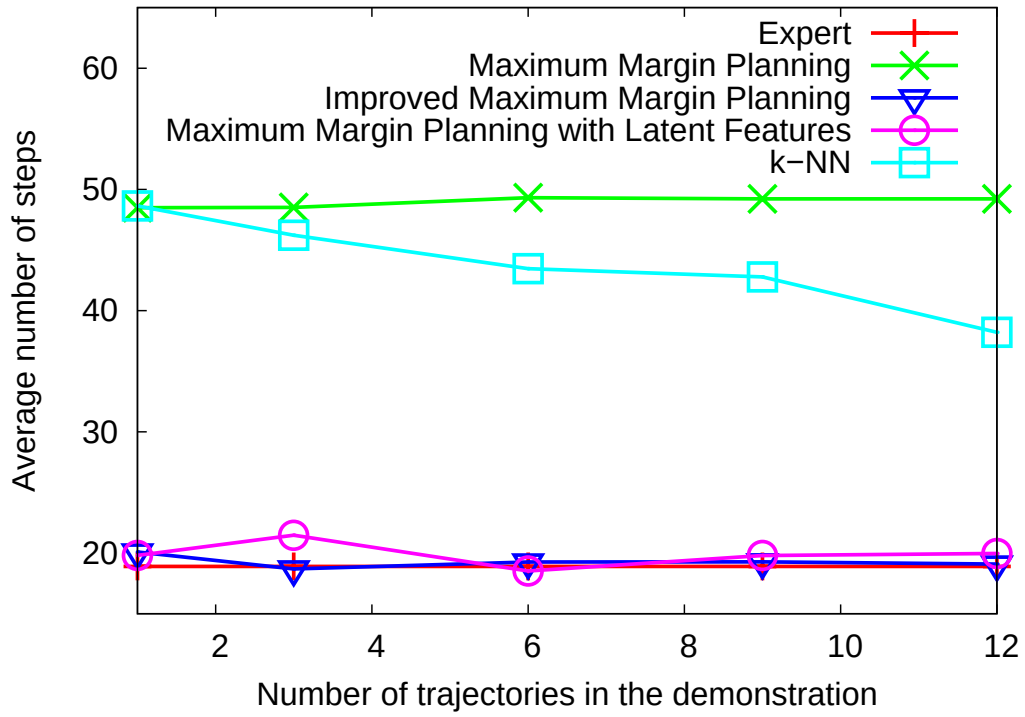


Figure 6.26: Average number of steps before reaching the finish line in racetrack (b).

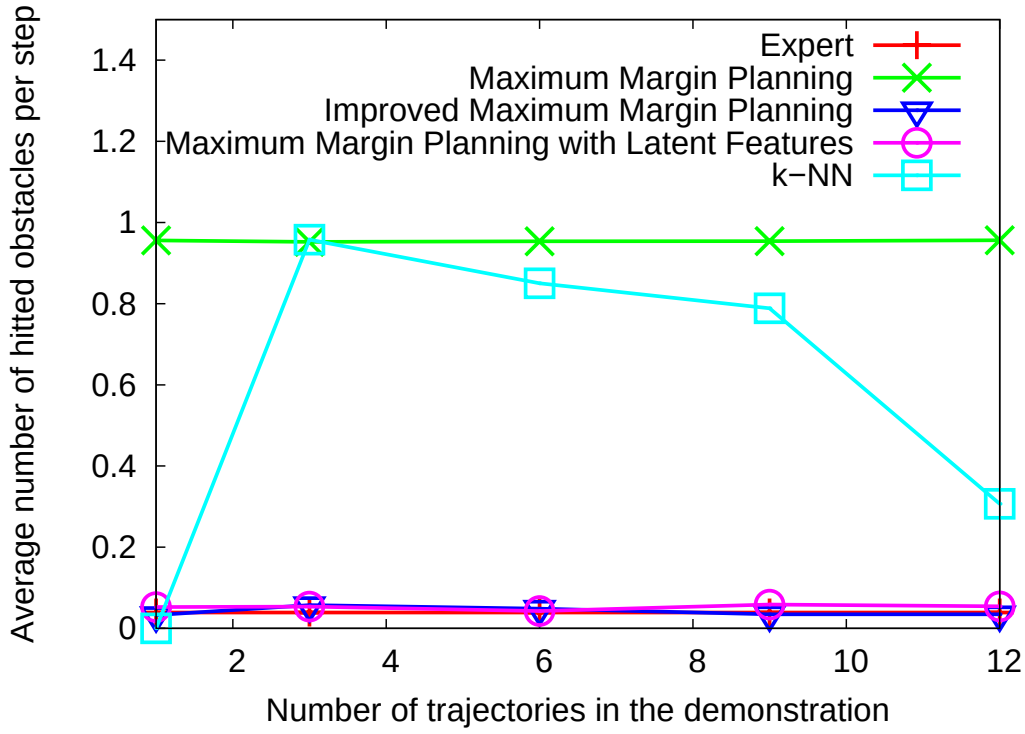


Figure 6.27: Average number of off-roads per step in racetrack (b).

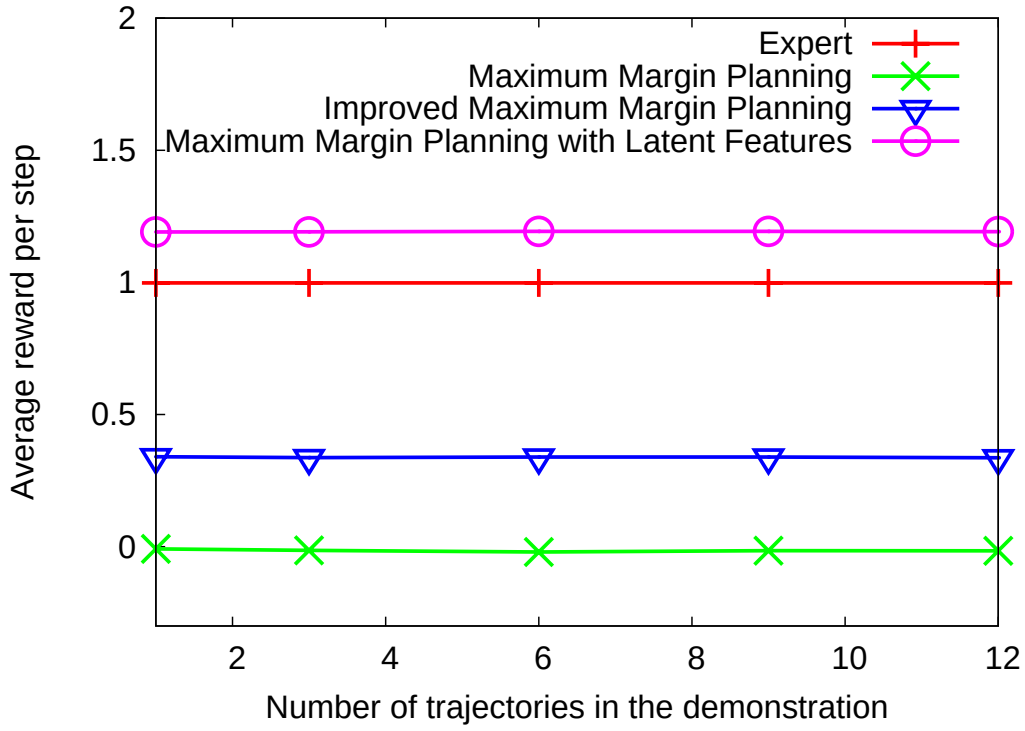


Figure 6.28: Average reward per step in racetrack (b).

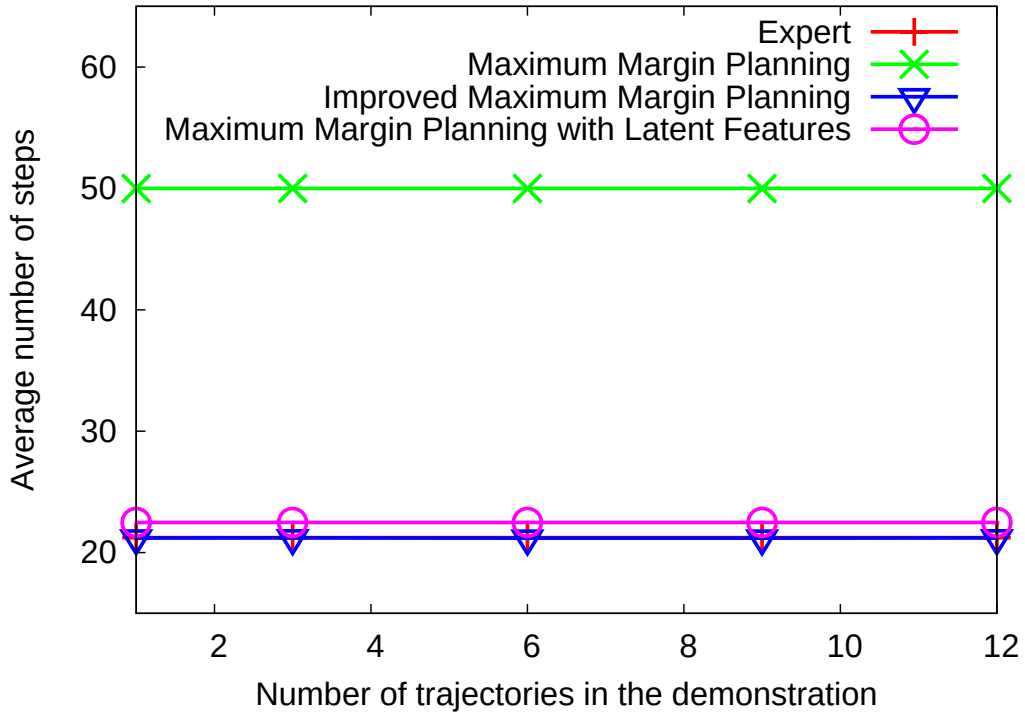


Figure 6.29: Average number of steps before reaching the finish line in racetrack (b).

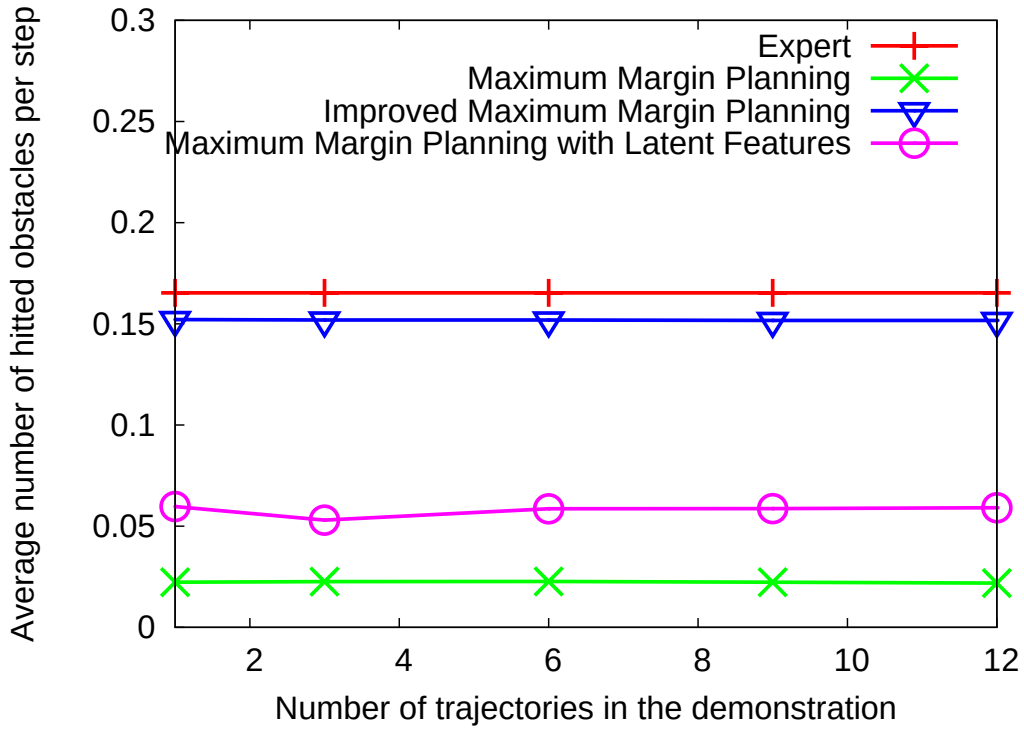


Figure 6.30: Average number of off-roads per step in racetrack (b).

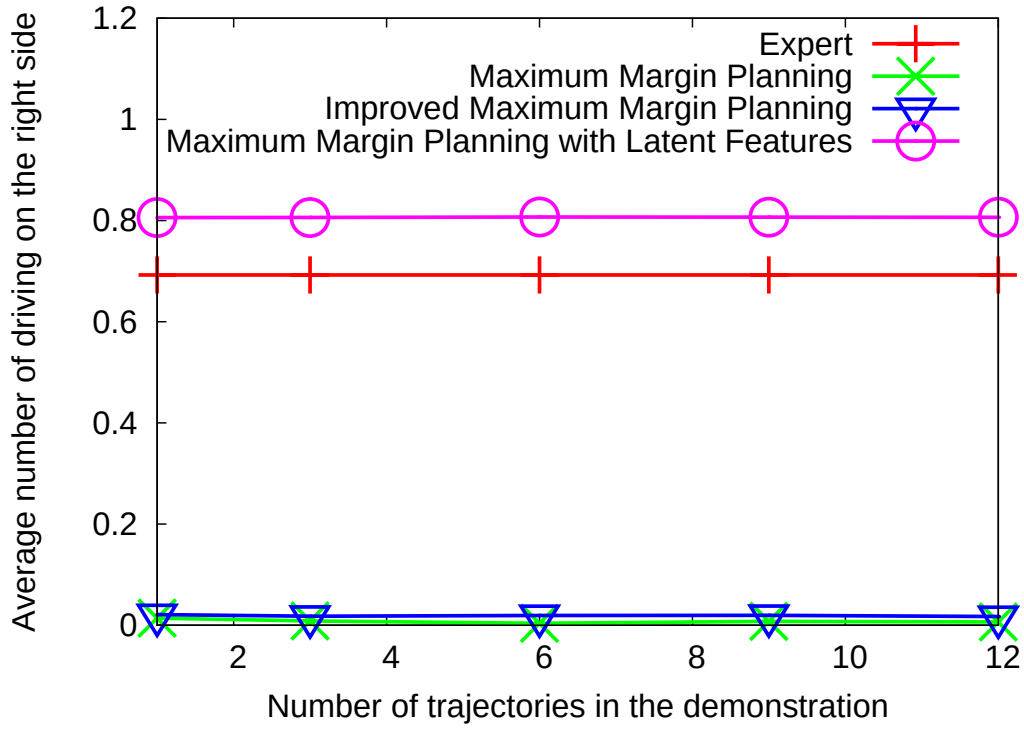


Figure 6.31: Average frequency of driving right in racetrack (c).

6.7.4 Discussion

In the previous section, we showed that the scarcity of the expert's demonstrations can be overcome by using the learned reward function for generating examples that cover the complete state space. The empirical analysis conducted on simulated systems suggests that the learned policies are indeed better when this approach is used. However, this approach requires solving an optimization problem where the objective function is nonconvex with potentially many local optima. In this section, we showed that the same problem can be solved by considering latent features that are a sufficient statistic of the basic features. Latent features are more likely to be encountered in larger regions of the state space, consequently, the frequency of latent features may be estimated more accurately.

The main drawback of this approach is the selection of useful latent features. A possible indicator of the utility of a latent feature is the subgradient of the cost function with respect to this feature, but more elaborated techniques should be studied. As a future work, we mainly target to compare this approach with the one proposed by Ratliff [2009], where the basic features are boosted by using a classifier.

6.8 Conclusion

Learning by imitation is very useful in situations where the task of the agent can be demonstrated by a human expert. This method significantly accelerates the learning process, compared to the fully autonomous methods, such as reinforcement learning.

However, most of the previous work in this topic was focused on completely observable environments. In this chapter, we showed how to use Predictive Policy Representations (PPRs), the method newly introduced in Chapter 4, in order to represent the expert's policy in a partially observable environment. The advantage of PPRs, compared to Finite-State Machines (FSMs), is the fact that their parameters are a function of only observable variables. We tested this model on two problems related to assistive robotics. In both problems, PPR outperformed the Baum-Welch algorithm for FSM, and the corresponding policies could be learned after a few repetitions.

However, there are at least three main drawbacks related to this method. The first one is the large size of the memory used for representing the dynamics of the policy (the dynamics matrix); this issue can be solved by using sparse matrix representations. The second drawback is the sensitivity of the dynamics matrix's rank to the errors in estimating the probabilities of tests. This problem can be solved by using recent methods in estimating the rank of matrices with noisy entries [Keshavan et al., 2009]. Finally, as for PSRs, the main problem of PPRs is the stability of the belief states. In fact, since the parameters of the learned PPRs are

underconstrained, the belief states of PPRs may become inaccurate after a large number of bayesian updates (see the conclusion of Chapter 5).

In fully observable environments, this type of direct imitation suffers from a serious drawback: the expert's demonstrated policy cannot be efficiently generalized to the remaining states, unless sufficiently expressive features are used in the learning. Inverse Reinforcement Learning (IRL) is a framework that solves this problem by restraining the space of hypothesis to the space of reward functions. The aim of IRL is to recover a reward function under which the expert's policy is optimal, rather than to directly mimic the actions of the expert. The learned reward function is then used to find an optimal policy.

Most IRL algorithms rely on the assumption that the reward function is a linear combination of states features, and the frequency of encountering each feature can be accurately estimated from the demonstration. However, the frequencies of the features might be poorly estimated when the number of demonstrations is small, as we showed in our experiments.

Inspired by the intuition that the states that are locally similar have the same optimal action in general, we introduced a new technique for generalizing the expert's policy based on soft local homomorphisms. Unlike other methods, our approach considers the long term dependencies between different states, rather than just immediate features. We also showed that using homomorphisms leads to a significant improvement in the quality of the learned policies. However, our approach lacks of a theoretical guarantee beyond the intuition. In fact, control policies are local and reactive in most states, such as avoiding obstacles during a navigation task, but there are always some critical states where the optimal actions cannot be explained by only the local dynamics of the system. Distinguishing between these states is crucial for providing a theoretical guarantee of our approach in future work.

We also showed that the problem of generalizing the expert's policy can be solved by using the learned reward function for generating examples that cover the complete state space. The same problem can also be solved by augmenting the set of basic features with the predictions of these features after a sequence of actions. These new features, called latent features, are more likely to be encountered in larger regions of the state space than the basic features. Consequently, the frequency of latent features may be estimated more accurately.

The main drawback of this latter approach is the selection of useful latent features. A possible indicator of the utility of a latent feature is the subgradient of the cost function with respect to this feature, but more elaborate techniques should be studied. In fact, this problem is equivalent to the problem of discovering core tests (or core histories) in predictive state (or policy) representations.

Chapter 7

Conclusion

This thesis considers the problem of sequential decision-making under uncertainty. We demonstrated that by using predictive representations, directly defined by the observed data, the planning complexity can be greatly reduced and the learning process significantly improved.

In this chapter, we conclude with a summary of the principal contributions of the work, along with a discussion of its limitations and an overview of future directions and open problems.

7.1 Summary of the Contributions

Conceptually, this dissertation can be broken into three main parts. The first part, consisting of Chapter 3, discusses the problem of reducing a Partially Observable Markov Decision Process (POMDP) into a compact and approximate Predictive State Representation (PSR). The second part, which corresponds to Chapters 4, 5 and the first part of Chapter 6, deals with using Predictive Policy Representations (PPRs) in a variety of problems related to sequential decision-making under uncertainty. The third part, which corresponds to the last part of Chapter 6, presents three techniques for improving the performance of Inverse Reinforcement Learning (IRL) algorithms. The first technique is based on homomorphism of MDPs for generalizing the examples provided by an expert. The second technique consists in bootstrapping the examples. The last one is a predictive reward representation. Below, we briefly summarize the main results and contributions of this work, with an emphasis on the connections between the parts.

7.1.1 Compression of POMDPs using approximate PSRs

The high dimensionality of the belief space in POMDPs is one of the major causes that severely restricts the applicability of this model in planning and prediction tasks. Predictive State Representations (PSRs), have been proposed by Littman et al. [2001] with the hope that they can contribute to solving this problem. However, the equivalent PSR of a POMDP model has generally almost the same dimensionality as the original model [James et al., 2004; Izadi, 2007]. Consequently, it is natural to look for approximation algorithms that generate a subset of core tests more appropriate for prediction and planning purposes.

In this thesis, we showed that this problem can be casted as a linear optimization problem. The proposed algorithm minimizes the potential prediction error caused by missing some core tests. The intuition behind this method is that the parameters of a PSR model can be adjusted in order to compensate the absence of some core tests. This algorithm was inspired by the work of Poupart [2005] on compressing POMDPs, known as Value-Directed Compression (VDC).

The main difference between our algorithm and VDC, besides the use of the PSR framework, is that in VDC, the reduced belief state is a vector containing two types of variables: the probabilities of specific sequences of actions and observations, and the expected reward of other sequences. At a given time-step, both the predicted reward and the probability of any observation are linear functions of the same belief state. This can result in a high prediction error if the rewards are not rescaled as probabilities. In our method, we used separate models for predicting expected rewards and observations. In this context, we presented an empirical evaluation on benchmark problems, illustrating the performance of this approach on prediction and planning tasks. The empirical results show that this approach can indeed be used for reducing the dimensionality of the state space while preserving an acceptable quality of the predictions and the learned policies.

However, there are many limitations to this work. First, the method used for selecting core tests is a simple heuristic favoring shorter core tests to longer ones. This heuristic is based on the assumption that core tests with higher probabilities are more important, and the fact that the probabilities of tests decrease as their length increases. This assumption is not true in general, and a further theoretical analysis should be considered at this level. Second, linear reduction of dimensionality is not as powerful as other techniques, such as Exponential Principal Components Analysis. Finally and most importantly, the experiments were performed on small-sized POMDP problems that are no longer challenging for today solvers. This is particularly important for the planning experiments, since the main motivation of this work is the intractability of planning with POMDPs.

7.1.2 Predictive Policy Representations

Predictive Policy Representation (PPR) is the main contribution of this thesis. Most of the previous work on planning and learning in partially observable environments focused on reducing the dimensionality of the state space, and only a little interest was given to reducing the dimensionality of the internal state space of an agent. The idea of using predictive representations for representing policies has been originally suggested by Wiewiora [2005], where this type of policies was called *regular policies*. In this thesis, we showed how to adapt PSRs in order to represent policies, and derived a Bayes' update rule for maintaining internal belief states. We also demonstrated the efficiency of this method through different problems related to sequential decision-making under uncertainty.

Amongst these problems, the first one was planning in decentralized POMDPs. This problem is NEXP-hard and remains one of the most difficult challenges in sequential decision-making [Bernstein et al., 2002]. Finding good solutions to decentralized POMDPs is so difficult because there is no optimality criterion for the policies of a single agent alone: whether a given policy is better or worse than another depends on the behavior of the remaining agents. Moreover, no agent is aware of the actual behavior of the remaining agents unless the agents can exchange their local information through communication, which is not always possible. Instead, each agent keeps a belief state on the policies of the other agents based on its local actions and observations received so far.

Previous work on this problem used decision trees for representing policies of the other agents. However, the number of decision trees increases double-exponentially with respect to the planning horizon. To solve this problem, we introduced a point-based planning algorithm where the belief states correspond to the probabilities of specific sequences of actions and observations instead of a distribution on all the possible decision trees. An empirical comparison of the same algorithm using the two types of representations shows that predictive policy representations significantly outperform decision tree representations.

We also introduced an exact planning algorithm for decentralized POMDPs based on predictive policy representations. Empirical evaluation showed that this algorithm outperforms the equivalent exact planning approach using decision trees. This performance is due to the fact that the size of belief states is significantly smaller when predictive representations are used. However, although the final algorithm is fairly simple, the derivation of Bellman-like update equations was too long and not straightforward. Another issue related to the use of predictive representations is that the belief space is not a simple simplex as in POMDPs. In fact, it is not yet clear which constraints are sufficient and necessary to define the space of beliefs with predictive representations.

We used Predictive Policy Representations also for reinforcement learning in partially observable environments. We showed that PPRs based on core histories share common char-

acteristics with both history-based models, such as decision trees, and models with internal beliefs, such as Finite-State Machines (FSMs). An empirical comparison of PPRs and FSMs showed that PPRs generally achieve better performances than FSMs. Nevertheless, PPRs suffer from the same difficult problem of selecting core tests (or core histories) discussed in the previous section. In fact, we needed to repeat the experiments several times before finding the best hyper-parameters that will be used for selecting the core histories. The other issue of PPRs and PSRs is the instability of the belief state. Indeed, the belief state starts making inaccurate predictions after a long number of bayesian updates.

Finally, we showed how PPRs can be used for modeling the policy of an expert agent, and how this policy can be learned from demonstrations. We also compared PPRs to FSMs on tasks related to learning the behavior of a robotic wheelchair user. The empirical results showed that PPRs were more accurate than FSMs. However, the instability of the belief state was a major drawback in this problem too.

7.1.3 Improving Inverse Reinforcement Learning Algorithms

We proposed three methods for learning from an expert in a completely observable stochastic environment where the demonstrations of the expert are scarce and cover only a small part of a large state space. To solve this problem, we first used a transfer method, known as soft homomorphism, in order to generalize the expert's policy to unvisited regions of the state space. This method was inspired by the intuition that the states that are locally similar have the same optimal action in general. Unlike other methods, this approach considers the long term dependency between different states, rather than just immediate features. Empirical results showed that using homomorphisms leads to a significant improvement in the quality of the learned policies. However, our approach lacks of a theoretical guarantee beyond the intuition. In fact, control policies are local and reactive in most states, such as avoiding obstacles during a navigation task, but there are always some critic states where the optimal actions cannot be explained by only the local dynamics of the system.

The second proposed method consists in using the provided examples of the expert's policy in order to define a modified Markov Decision Process, where the learner agent is constrained to select the same actions as the expert in all the states that appear in the examples. Using an initial reward function, an optimal policy in the modified MDP is found by using a planning algorithm. This latter policy is then used as the expert's policy and fed back to the learning algorithm.

The final method that we proposed was a Predictive Reward Representation. Past work on IRL considered the reward function as a linear combination of basic features, and the expected frequencies of these features under the expert's policy are empirically estimated from the demonstration. However, the quality of the learned policies is highly sensitive to

the frequency estimation error. This error is mainly due to the sparsity of the features over the state space. To solve this problem, we introduced a new representation where the reward function is a linear combination of the predictions of the basic features after a sequence of actions. These extended features are in general more informative than the basic ones. Preliminary results on a simulated car racing problem show that this approach is able to learn good policies from a small number of demonstrations. The main drawback of this latter approach is the selection of useful extended features. In fact, this problem is equivalent to the problem of selecting core tests (or core histories) in predictive state (or policy) representations.

7.2 Future Research

This dissertation may be extended in many ways to solve more efficiently problems of sequential decision-making under uncertainty. Below, we briefly outline three of what we consider to be the most important directions of future work.

Selection of core tests and core histories: Throughout this thesis, we noticed that the major limitation of using predictive representations lies in the process of selecting the core tests, or histories. This problem, known as the *discovery* problem, has been tackled with different techniques in the past, but all of the proposed solutions were based on defining simple criteria for deciding if a given test is a core one or not [Makino and Takagi, 2008]. Similar heuristics were used in this thesis. For instance, in the problem of reducing a POMDP to a compact PSR, we considered the length of a test as an indicator of its importance. Alternatively, we can consider a large set of tests (larger than the state space), and introduce binary variables, with values in $\{0, 1\}$, in the objective function for selecting only important tests. One can also add an L_1 regularization on the weights of the extended core tests in order to reduce their number.

Constraining the belief states in PSRs and PPRs: From the experimental analysis performed on PSRs and PPRs across this thesis, we noticed that the belief states of these models are often unstable. This is because the belief state is a vector of predictions, and errors in predictions accumulate over time. After a certain number of belief updates using Bayes' rule, the belief state diverges from the domain of valid probabilities, and predicts probabilities higher than 1 or lower than 0. This is not a problem in models defined by probability distributions, such as POMDPs. In POMDPs, if the initial belief state is a valid probability distribution, then it suffices to ensure that the transition and observation matrices are stochastic ones (i.e. they have positive entries and each row sums to 1) in order to ensure that the belief state will be a valid distribution after an arbitrary number of updates. This is not the case in PSRs. A first solution to this problem was proposed by James [2005], where reset points were periodically used in order to correct the predictions of the PSR model by resetting the belief state to a deterministic state. Recently, a family of simple sufficient, though not necessary, constraints for

defining valid parameters for PSR was given by Wolfe [2010b]. This problem should be efficiently addressed before considering the deployment of PSR techniques on real applications.

Robotic applications: After addressing the two issues above, most of the techniques proposed in this thesis can be deployed on real applications. For instance, many problems in robotics can be directly casted as problems of sequential decision-making under uncertainty. In fact, robotics traditionally have been a favorite field for applying artificial intelligence and machine learning techniques in general [Giguère and Dudek, 2009]. Given that this thesis was originally inspired by robotic applications, such as the smart wheelchair project [Pineau and Atrash, 2007], a natural extension of this work would be to deploy the proposed techniques on this type of applications.

Bibliography

Abbeel, P. and Ng, A. Y. (2004). Apprenticeship Learning via Inverse Reinforcement Learning. In *Proceedings of the Twenty-first International Conference on Machine Learning (ICML'04)*, pages 1–8.

Aberdeen, D. and Baxter, J. (2002). Scaling Internal-State Policy-Gradient Methods for POMDPs. In *Proceedings of the 19th International Conference on Machine Learning (ICML'02)*, pages 3–10.

Aberdeen, D., Buffet, O., and Thomas, O. (2007). Policy-Gradients for PSRs and POMDPs. In *Proceeding of the 11th Conference on Artificial Intelligence and Statistics (AISTAT'07)*.

Aboaf, E., Drucker, S., and Atkeson, C. (1989). Task-Level Robot Learning: Juggling a Tennis Ball More Accurately. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'89)*, pages 1290–1295.

Amari, S.-I. (1998). Natural Gradient Works Efficiently in Learning. *Neural Computing*, 10(2):251–276.

Amato, C., Bernstein, D., and Zilberstein, S. (2007a). Optimizing Fixed-Size Stochastic Controllers for POMDPs and Decentralized POMDPs . *Technical Report UM-CS-2007-70, University of Massachusetts*.

Amato, C., Bernstein, D., and Zilberstein, S. (2007b). Optimizing Memory-Bounded Controllers for Decentralized POMDPs. In *Proceedings of the Twenty Third Conference on Uncertainty in Artificial Intelligence (UAI'07)*.

Aras, R., Dutech, A., and Charpillet, F. (2007). Mixed Integer Linear Programming for Exact Finite-Horizon Planning in Decentralized POMDPs. In *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS'07)*, pages 18–25.

Atkeson, C. and Schaal, S. (1997). Robot Learning From Demonstration. In *Proceedings of the Fourteenth International Conference on Machine Learning (ICML'97)*, pages 12–20.

Bagnell, J. A., Kakade, S., Ng, A. Y., and Schneider, J. G. (2003). Policy Search by Dynamic Programming. In *Advances in Neural Information Processing Systems 15 (NIPS'03)*.

Baird, L. (1993). Advantage Updating. *Technical report WL-TR-93-1146, Wright Lab.*

Baird, L. and Moore, A. (1999). Gradient Descent for General Reinforcement Learning. In *Advances in Neural Information Processing Systems 11 (NIPS'99)*, pages 968–974.

Bakker, B. (2001). Reinforcement Learning with Long Short-Term Memory. In *Advances in Neural Information Processing Systems 13 (NIPS'01)*, pages 1475–1482.

Baxter, J. and Bartlett, P. (2000). Reinforcement Learning in POMDP's via Direct Gradient Ascent. In *Proceeding of the 15th International Conference on Machine Learning (ICML'00)*, pages 41–48.

Becker, R., Zilberstein, S., Lesser, V., and Goldman, C. (2004). Solving Transition Independent Decentralized Markov Decision Processes. *Journal of Artificial Intelligence Research*, (22):423–455.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press.

Bernstein, D., Hansen, E., and Zilberstein, S. (2007). Bounded Policy Iteration for Decentralized POMDPs. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 1287–1292.

Bernstein, D., Immerman, N., and Zilberstein, S. (2002). The Complexity of Decentralized Control of Markov Decision Processes. *Mathematics of Operations Research*, 27(4):819–840.

Besse, C. and Chaib-draa, B. (2009). Quasi-Deterministic Partially Observable Markov Decision Processes. In *Proceedings of 16th International Conference On Neural Information Processing (ICONIP'09)*, pages 237–246.

Billard, A. (2002). Imitation: A Review. In *The Handbook of Brain Theory and Neural Network. 2nd Edition. Michael A. Arbib (editor), pages 566-569. MIT Press.*

Bonet, B. (2002). An epsilon-Optimal Grid-Based Algorithm for Partially Observable Markov Decision Processes. In *Proceedings of the Nineteenth International Conference on Machine Learning (ICML'02)*, pages 51–58.

Bonet, B. and Geffner, H. (2003). Faster Heuristic Search Algorithms for Planning with Uncertainty and Full Feedback. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 1233–1238.

Boots, B., Siddiqi, S. M., and Gordon, G. J. (2010). Closing the Learning-Planning Loop with Predictive State Representations (Extended Abstract). In *Proceedings of the Ninth International Joint Conference on Autonomous Agents and Multiagent Aystems (AAMAS'10).*

Boularias, A. (2008). A Predictive Model for Imitation Learning in Partially Observable Environments. In *Proceedings of the International Conference on Machine Learning and Applications (ICMLA'08)*, pages 83–90.

Boularias, A. and Chaib-draa, B. (2007). Les Représentations Prédictives des États et des Politiques. In *Actes des Quatrièmes Journées Francophones Modèles Formels de l'Interaction (MFI'07).*

Boularias, A. and Chaib-draa, B. (2008a). Exact Dynamic Programming for Decentralized POMDPs with Lossless Policy Compression. In *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS'08)*, pages 20–27.

Boularias, A. and Chaib-draa, B. (2008b). Planning in Decentralized POMDPs with Predictive Policy Representations. In *Proceedings of ICAPS'08 Multiagent Planning Workshop (MASPLAN'08)*.

Boularias, A. and Chaib-draa, B. (2009). Predictive Representations for Policy Gradient in POMDPs. In *Proceedings of the Twenty-sixth International Conference on Machine Learning (ICML'09)*, pages 65–72.

Boularias, A. and Chaib-draa, B. (2010). Apprenticeship Learning via Soft Local Homomorphisms. In *Proceedings of 2010 IEEE International Conference on Robotics and Automation (ICRA'10)*.

Boularias, A., Izadi, M., and Chaib-draa, B. (2008). Prediction-directed Compression of POMDPs. In *Proceedings of the International Conference on Machine Learning and Applications (ICMLA'08)*, pages 99–105.

Boutilier, C., Dean, T., and Hanks, S. (1999). Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. *Journal of Artificial Intelligence Research (JAIR)*, 11:1–94.

Boutilier, C. and Poole, D. (1996). Computing Optimal Policies for Partially Observable Decision Processes Using Compact Representations. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI'96)*, pages 1168–1175.

Bowling, M., McCracken, P., James, M., Neufeld, J., and Wilkinson, D. (2006). Learning Predictive State Representations Using Non-blind Policies. In *Proceedings of the Twenty-Third International Conference on Machine Learning (ICML'06)*.

Brafman, R. I. and Tennenholtz, M. (2003). R-max - A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning. *Journal of Machine Learning Research*, 3:213–231.

Bush, K. and Pineau, J. (2009). Manifold Embeddings for Model-based Reinforcement Learning in Partially Observable Domains. In *Advances in Neural Information Processing Systems (NIPS'09)*, pages 189–197.

Byrne, R. (1995). *The Thinking Ape*. Oxford University Press.

Calinon, S. and Billard, A. (2005). Recognition and Reproduction of Gestures using a Probabilistic Framework Combining PCA, ICA and HMM. In *Proceedings of the 22nd International Conference on Machine Learning (ICML'05)*, pages 105–112.

Casella, G. and Robert, C. P. (1996). Rao-blackwellisation of Sampling Schemes. *Biometrika*, 15:229–235.

Cassandra, A. (1998). *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, Brown University, Providence, RI, USA.

Cassandra, A., Littman, M. L., and Zhang, N. L. (1997). Incremental Pruning: A Simple, Fast, Exact Method for Partially Observable Markov Decision Processes. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI'97)*, pages 54–61.

Chrisman, L. (1992). Reinforcement Learning with Perceptual Aliasing: The Perceptual Distinctions Approach. In *Proceeding of the National Conference on Artificial Intelligence (AAAI'92)*, pages 183–188.

Dai, P. and Goldsmith, J. (2007). Topological Value Iteration Algorithm for Markov Decision processes. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 1860–1865.

Dautenhahn, K. (1995). Getting to Know Each Other—Artificial Social Intelligence for Autonomous Robots. *Robotics and Autonomous Systems*, (16):333–356.

Dean, T. and Lin, S. (1995). Decomposition Techniques for Planning in Stochastic Domains. Technical Report CS-95-10, Providence, RI, USA.

Decety, J., Chaminade, T., Grezes, J., and Meltzoff, A. (2002). A Pet Exploration of the Neural Mechanisms Involved in Reciprocal Imitation. *Neuroimage*, 15:265–272.

Dibangoye, J. (2010). *Contributions à la résolution des processus décisionnels de Markov centralisés et décentralisés: Algorithmes et Théorie*. PhD thesis, Université Laval and Université de Caen.

Dibangoye, J. S., Mouaddib, A., and Chai-draa, B. (2009a). Point-based Incremental Pruning Heuristic for Solving Finite-Horizon DEC-POMDPs. In *Proceedings of the Eighth International Joint Conference on Autonomous Agents and Multiagent Aystems (AAMAS'09)*, pages 569–576.

Dibangoye, J. S., Shani, G., Chaib-draa, B., and Mouaddib, A. (2009b). Topological Order Planner for POMDPs. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI'09)*, pages 1684–1689.

Doshi, F., Pineau, J., and Roy, N. (2008). Reinforcement Learning with Limited Reinforcement: Using Bayes Risk for Active Learning in POMDPs. In *Proceeding of the Twenty-Fifth International Conference on Machine Learning (ICML'08)*, pages 256–263.

Emery-Montemerlo, R., Gordon, G., Schneider, J., and Thrun, S. (2004). Approximate Solutions for Partially Observable Stochastic Games with Common Payoffs. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'04)*, pages 136–143.

Feng, Z. and Zilberstein, S. (2005). Efficient Maximization in Solving POMDPs. In *Proceedings of the 20th national conference on Artificial intelligence (AAAI'05)*, pages 975–980.

Fine, S., Singer, Y., and Tishby, N. (1998). The Hierarchical Hidden Markov Model: Analysis and Applications. *Machine Learning*, 32(1):41–62.

Foka, A. and Trahanias, P. (2007). Real-time Hierarchical POMDPs for Autonomous Robot Navigation. *Robotics and Autonomous Systems*, 55(7):561–571.

Ghavamzadeh, M. and Engel, Y. (2007). Bayesian Actor-Critic Algorithms. In *Proceedings of the Twenty-fourth International Conference on Machine Learning (ICML'07)*, pages 297–304.

Giguère, P. and Dudek, G. (2009). Clustering Sensor Data for Autonomous Terrain Identification Using Time-Dependency. *Autonomous Robots*, 26(2-3):171–186.

Glynn, P. (1987). Likelihood Ratio Gradient Estimation: An Overview. In *Proceedings of the 1987 Winter Simulation Conference*, pages 366–375.

Glynn, P. (1990). Likelihood Ratio Gradient Estimation for Stochastic Systems. *Communications of the ACM*, 33(10):75–84.

Gomez, F. J. and Schmidhuber, J. (2005). Co-Evolving Recurrent Neurons Learn Deep Memory POMDPs. In *Genetic and Evolutionary Computation Conference (GECCO'05)*, pages 491–498.

Hachiya, H., Peters, J., and Sugiyama, M. (2009). Efficient Sample Reuse in EM-Based Policy Search. In *Proceedings of the 20th European Conference on Machine Learning (ECML'09)*, pages 469–484.

Hansen, E., Bernstein, D., and Zilberstein, S. (2004). Dynamic Programming for Partially Observable Stochastic Games. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI'04)*, pages 709–715.

Hansen, E. and Zilberstein, S. (1998). Heuristic Search in Cyclic AND/OR Graphs. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'98)*, pages 412–418.

Hansen, E. A. (1998). Solving POMDPs by Searching in Policy Space. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI'98)*, pages 211–219.

Hauskrecht, M. (1997). Incremental Methods for Computing Bounds in Partially Observable Markov Decision Processes. In *Proceedings of the Twelfth Conference on Artificial Intelligence (AAAI'97)*, pages 734–739.

Hauskrecht, M. (2000). Value-Function Approximations for Partially Observable Markov Decision Processes. *Journal of Artificial Intelligence Research*, (13):33–94.

Hoey, J. and Little, J. (2007). Value-Directed Human Behavior Analysis from Video Using Partially Observable Markov Decision Processes. *IEEE transactions on pattern analysis and machine intelligence*, 29(7):1118–1132.

Holmes, M. P. and Isbell, C. (2006). Looping Suffix Tree-based Inference of Partially Observable Hidden State. In *Proceedings of the 23rd International Conference on Machine learning (ICML'06)*, pages 409–416.

Howard, R. (1960). *Dynamic Programming and and Markov Processes*. The M.I.T. Press.

Hundt, C., Panangaden, P., Pineau, P., and Precup, D. (2006). Representing Systems with Hidden State. In *Proceedings of The 21st National Conference on Artificial Intelligence (AAAI'06)*, pages 368–374.

Huynh, V. A. (2008). Combining Local and Global Optimization for Planning and Control in Information Space. Master's thesis, Massachusetts Institute of Technology.

Izadi, M. (2007). *On Knowledge Representation and Decision Making under Uncertainty*. PhD thesis, McGill University, Montreal, Canada.

Izadi, M. and Precup, D. (2003). A Planning Algorithm for Predictive State Representation. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 1520–1521.

Jaeger, H. (2000). Observable Operator Models for Discrete Stochastic Time Series. *Neural Computation*, 12(6):1371–1398.

James, M. (2005). *Using Predictions for Planning and Modeling in Stochastic Environments*. PhD thesis, The University of Michigan.

James, M., Singh, S., and Littman, M. (2004). Planning with Predictive State Representations. In *Proceedings of 2004 International Conference on Machine Learning and Applications (ICMLA'04)*, pages 304–311.

James, M. R. and Singh, S. (2004). Learning and Discovery of Predictive State Representations in Dynamical Systems with Reset. In *Proceedings of the Twenty-first International Conference on Machine learning (ICML'04)*.

Kakade, S. (2002). A Natural Policy Gradient. In *Advances in Neural Information Processing Systems 13 (NIPS'01)*, pages 1531–1538.

Kearns, M. and Singh, S. (2002). Near-Optimal Reinforcement Learning in Polynomial Time. *Machine Learning*, 49(2-3):209–232.

Keshavan, R., Montanari, A., and Oh, S. (2009). Matrix Completion from Noisy Entries. In *Neural Information Processing Systems (NIPS'09)*.

Khan, O. Z., Poupart, P., and Black, J. P. (2009). Minimal Sufficient Explanations for Factored Markov Decision Processes. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*.

Koenig, S. and Simmons, R. (1996). Unsupervised Learning of Probabilistic Models for Robot Navigation. In *Proceedings of the International Conference on Robotics and Automation (ICRA'96)*, pages 2301–2308.

Koller, D., Megiddo, N., and von Stengel, B. (1994). Fast Algorithms for Finding Randomized Strategies in Game Trees. In *Proceedings of 26th Annual ACM Symposium on Theory of Computing*, pages 750–759.

Kurniawati, H., Hsu, D., and Lee, W. S. (2008). SARSOP: Efficient Point-based POMDP Planning by Approximating Optimally Reachable Belief Spaces. In *Proceedings of the International Conference on Robotics: Science and Systems (RSS'08)*.

L'Ecuyer, P. (1991). An Overview of Derivative Estimation. In *Proceedings of the 1991 Winter Simulation Conference*, pages 207–217.

Lee, D. and Seung, S. (2000). Algorithms for Non-negative Matrix Factorization. In *Advances in Neural Information Processing Systems (NIPS'00)*, pages 556–562.

Li, X., Cheung, W., and Liu, J. (2009). Improving POMDP Tractability via Belief Compression and Clustering. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 40(1):125–136.

Li, X., Cheung, W., Liu, J., and Wu, Z. (2007). A Novel Orthogonal NMF-based Belief Compression for POMDPs. In *Proceedings of the 24th international conference on Machine learning (ICML'07)*, pages 537–544.

Littman, M. (1996). *Algorithms for Sequential Decision Making*. PhD thesis, Brown University.

Littman, M., Sutton, R., and Singh, S. (2001). Predictive Representations of State. In *Advances in Neural Information Processing Systems 14 (NIPS'01)*, pages 1555–1561.

Lopes, M., Melo, F., and Montesano, L. (2009). Active Learning for Reward Estimation in Inverse Reinforcement Learning. In *European Conference on Machine Learning (ECML'09)*, pages 31–46.

Lusena, C., Goldsmith, J., and Mundhenk, M. (2001). Nonapproximability Results for Partially Observable Markov Decision Processes. *Journal of Artificial Intelligence Research (JAIR)*, 14:83–103.

Madani, O., Hanks, S., and Condon, A. (1999). On the Undecidability of Probabilistic Planning and Infinite-horizon Partially Observable Markov Decision Problems. In *Proceedings of the 11th Innovative Applications of Artificial Intelligence Conference (ICTAI'99)*, pages 541–548.

Makino, T. and Takagi, T. (2008). On-line Discovery of Temporal-Difference Networks. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML'08)*, pages 632–639.

Markov, A. (1913). An Example of Statistical Investigation in The Text of "Eugene Onegin" Illustrating Coupling "Tests" in Chains. In *Proceedings of the Academy of Science of St. Petersburg*, pages 7:153–162.

Martinez-Cantin, R., de Freitas, N., Brochu, E., Castellanos, J. A., and Doucet, A. (2009). A Bayesian Exploration-Exploitation Approach for Optimal Online Sensing and Planning with a Visually Guided Mobile Robot. *Autonomous Robots*, 27(2):93–103.

McCallum, A. (1993). Overcoming Incomplete Perception with Util Distinction Memory. In *International Conference on Machine Learning (ICML'93)*, pages 190–196.

McCallum, A. (1995). Instance-Based Utile Distinctions for Reinforcement Learning with Hidden State. In *Proceedings of the International Conference on Machine Learning (ICML'95)*, pages 387–395.

Mccallum, A. (1996). *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, Rochester, New York ,USA.

McCracken, P. and Bowling, M. (2006). Online Discovery and Learning of Predictive State Representations. In *Advances in Neural Information Processing Systems 18 (NIPS'06)*, pages 875–882.

Meltzoff, A. (1995). Understanding the Intentions of Others : Re-enactment of Intended Acts by 18-month-old Children. *Developmental Psychology*, 31:838–850.

Meltzoff, A. and Moore, M. (1977). Imitation of Facial and Manual Gestures by Human Neonates. *Science*, 198:74–78.

Meuleau, N., Peshkin, L., Kim, K., and Kaelbling, L. (1999). Learning Finite-State Controllers for Partially Observable Environments. In *Proceeding of the 15th Conference Uncertainty in Artificial Intelligence (UAI'99)*, pages 427–436.

Moore, A. and Atkeson, C. (1993). Prioritized Sweeping: Reinforcement Learning with Less Data and Less Time. *Machince Learning*, 13:103–130.

Nechyba, M. and Xu, Y. (1995). Human Skill Transfer: Neural Networks as Learners and Teachers. In *International Conference on Intelligent Robots and Systems, Human Robot Interaction and Cooperative Robots (IROS'95)*, pages 314–319.

Neu, G. and Szepesvári, C. (2007). Apprenticeship Learning using Inverse Reinforcement Learning and Gradient Methods. In *Conference on Uncertainty in Artificial Intelligence (UAI'07)*, pages 295–302.

Ng, A. and Russell, S. (2000). Algorithms for Inverse Reinforcement Learning. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML'00)*, pages 663–670.

Nourbakhsh, I., Powers, R., and Birchfield, S. (1995). DERVISH- An Office-Navigating Robot. *AI Magazine*, 16:53–60.

Oliehoek, F. A., Spaan, M. T. J., and Vlassis, N. A. (2008). Optimal and Approximate Q-value Functions for Decentralized POMDPs. *Journal of Artificial Intelligence Research (JAIR)*, 32:289–353.

Papadimitriou, C. and Tsitsiklis, J. (1987). The Complexity of Markov Decision Process. *Mathematics of Operations Research*, 12(3):441–450.

Paquet, S. (2006). *Distributed Decision-Making and Task Coordination in Dynamic, Uncertain and Real-Time Multiagent Environments*. PhD thesis, Université Laval.

Paquet, S., Tobin, L., and Chaib-draa, B. (2005). An Online POMDP Algorithm for Complex Multiagent Environments. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 970–977.

Peshkin, L. (2001). *Reinforcement Learning by Policy Search*. PhD thesis, Massachusetts Institute of Technology.

Peters, J. and Schaal, S. (2006). Policy Gradient Methods for Robotics. In *Proceedings of the IEEE International Conference on Intelligent Robotics Systems*, pages 2219–2225.

Peters, J. and Schaal, S. (2008). Reinforcement Learning of Motor Skills with Policy Gradients. *Neural Networks*, 21(4):682–697.

Peters, J., Vijayakumar, S., and Schaal, S. (2005). Natural Actor-Critic. In *Proceedings of the 16th European Conference on Machine Learning (ECML'05)*, pages 280–291.

Pfeffer, A. (2001). Sufficiency, Separability and Temporal Probabilistic Models. In *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence (UAI'01)*, pages 421–428.

Piaget, J. (1951). *Play, Dreams, and Imitation in Childhood*. W.W. Norton.

Pineau, J. (2004). *Tractable Planning Under Uncertainty: Exploiting Structure*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA.

Pineau, J. and Atrash, A. (2007). SmartWheeler: A Robotic Wheelchair Test-bed for Investigating New Models of Human-Robot Interaction. In *AAAI Spring Symposium on Multidisciplinary Collaboration for Socially Assistive Robotics*.

Pineau, J. and Gordon, G. (2005). POMDP Planning for Robust Robot Control. *International Symposium on Robotics Research (ISRR)*, 28:69–82.

Pineau, J., Gordon, J., and Thrun, J. (2003). Point-based Value Iteration: An Anytime Algorithm for POMDPs. In *International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 1025 – 1032.

Pomerleau, D. (1989). ALVINN: An Autonomous Land Vehicle in a Neural Network. In *Neural Information Processing Systems (NIPS'89)*, pages 769–776.

Poupart, P. (2005). *Exploiting Structure to Efficiently Solve Large Scale Partially Observable Markov Decision Processes*. PhD thesis, University of Toronto, Toronto, Canada.

Poupart, P. and Boutilier, C. (2002). Value-Directed Compression of POMDPs. In *Advances in Neural Information Processing Systems 14 (NIPS)*, pages 1547–1554.

Powell, M. J. D. (1993). On The Number of Iterations of Karmarkar's Algorithm for Linear Programming. *Mathematical Programming: Series A and B*, 62:153–197.

Powell, W. (2007). *Approximate Dynamic Programming: Solving the Curses of Dimensionality.* John Wiley and Sons, Inc.

Prentice, S. and Roy, N. (2009). The Belief Roadmap: Efficient Planning in Belief Space by Factoring the Covariance. *International Journal of Robotics Research*, 28.

Pynadath, D. V. and Tambe, M. (2002). The Communicative Multiagent Team Decision Problem: Analyzing Teamwork Theories and Models. *Journal of Artificial Intelligence Research*, (16):389–423.

Rabiner, L. and Juang., B. (1986). An Introduction to Hidden Markov Models. *IEEE Acoustic Speech Signal Processing Magazine*, 3:4–16.

Rabinovich, Z., Goldman, C., and Rosenschein, J. (2003). The Complexity of Multiagent Systems: the Price of Silence. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'03)*, pages 1102–1103.

Ramachandran, D. and Amir, E. (2007). Bayesian Inverse Reinforcement Learning. In *Proceedings of The twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 2586–2591.

Ratliff, N. (2009). *Learning to Search: Structured Prediction Techniques for Imitation Learning.* PhD thesis, Carnegie Mellon University.

Ratliff, N., Bagnell, J., and Zinkevich, M. (2006). Maximum Margin Planning. In *Proceedings of the Twenty-third International Conference on Machine Learning (ICML'06)*, pages 729–736.

Ratliff, N., Silver, D., and Bagnell, A. (2009). Learning to Search: Functional Gradient Techniques for Imitation Learning. *Autonomous Robots*, 27(1):25–53.

Ravindran, B. (2004). *An Algebraic Approach to Abstraction in Reinforcement Learning.* PhD thesis, University of Massachusetts, Amherst MA.

Rivest, R. and Schapire, R. (1994). Diversity-based Inference of Finite Automata. *Journal of ACM*, 41(3):555–589.

Rosencrantz, M., Gordon, G., and Thrun, S. (2004). Learning Low Dimensional Predictive Representations. In *Proceedings of the twenty-first International Conference on Machine learning (ICML'04)*, pages 695–702.

Ross, S. and Chaib-draa, B. (2007). AEMS: An Anytime Online Search Algorithm for Approximate Policy Refinement in Large POMDPs. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 2592–2598.

Ross, S., Chaib-draa, B., and Pineau, J. (2008a). Bayes-Adaptive POMDPs. In *Advances in Neural Information Processing Systems 20 (NIPS'07)*, pages 1225–1232, Cambridge, MA. MIT Press.

Ross, S. and Pineau, J. (2008). Model-Based Bayesian Reinforcement Learning in Large Structured Domains. In *Uncertainty in Artificial Intelligence (UAI'08)*, pages 476–483.

Ross, S., Pineau, J., Paquet, S., and Chaib-draa, B. (2008b). Online Planning Algorithms for POMDPs. *Journal of Artificial Intelligence Research (JAIR)*, 32:663–704.

Roy, N., Gordon, J., and Thrun, S. (2005). Finding Approximate POMDP Solutions Through Belief Compression. *Journal of Artificial Intelligence Research*, 23:1–40.

Roy, N. and He, R. (2009). Efficient POMDP Forward Search by Predicting the Posterior Belief Distribution. *Technical report MIT-CSAIL-TR-2009-044, Massachusetts Institute of Technology*.

Rudary, M. (2008). *On Predictive Linear Gaussian Models*. PhD thesis, The University of Michigan.

Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 3rd edition edition.

Saad, Y. (1996). *Iterative Methods for Sparse Linear Systems*. PWS publishing company, Boston, USA.

Schaal, S. (1999). Is Imitation Learning the Route to Humanoid Robots? *Trends in Cognitive Sciences*, 3(6):233–242.

Schaal, S., Ijspeert, A., and Billard, A. (2003). Computational Approaches to Motor Learning by Imitation. (1431):537–547.

Sehnke, F., Osendorfer, C., Ruckstiei, T., Graves, A., Peters, J., and Schmidhuber, J. (2010). Parameter-exploring Policy Gradients. *Neural Networks*, 23(4):551–559.

Seuken, S. and Zilberstein, S. (2007a). Improved Memory-Bounded Dynamic Programming for Decentralized POMDPs. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence (UAI'07)*.

Seuken, S. and Zilberstein, S. (2007b). Memory-Bounded Dynamic Programming for DEC-POMDPs. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 1696–1702.

Shalizi, C. and Shalizi, K. (2004). Blind Construction of Optimal Nonlinear Recursive Predictors for Discrete Sequences. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence (UAI'04)*, pages 504–511.

Shani, G., Brafman, R., and Shimony, S. (2008a). Prioritizing Point-based POMDP Solvers. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 38(6):1592–605.

Shani, G., Poupart, P., Brafman, R. I., and Shimony, S. E. (2008b). Efficient ADD Operations for Point-Based Algorithms. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*, pages 330–337.

Shelton, C. R. (2001). *Importance Sampling for Reinforcement Learning with Multiple Objectives*. PhD thesis, Massachusetts Institute of Technology.

Singh, S., Jaakkola, T., and Jordan., M. (1994). Learning without State-estimation in Partially Observable Markovian Decision Processes. In *Proceedings of the 11th International Conference on Machine Learning (ICML'94)*, pages 84–292.

Singh, S., James, M., and Rudary, M. (2004). Predictive State Representations: A New Theory for Modeling Dynamical Systems. In *Uncertainty in Artificial Intelligence: Proceedings of the Twentieth Conference (UAI'04)*.

Singh, S., Littman, M. L., Jong, N. K., Pardoe, D., and Stone, P. (2003). Learning Predictive State Representations. In *Proceedings of the 20th International Conference on Machine Learning (ICML'03)*, pages 712–719.

Smallwood, R. D. and Sondik, E. J. (1971). The Optimal Control of Partially Observable Markov Decision Processes Over A Finite Horizon. *Operations Research*, 21(5):1557–1566.

Smith, T. and Simmons, R. (2004). Heuristic Search Value Iteration for POMDPs. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence (UAI'04)*, pages 520–527.

Smith, T. and Simmons, R. (2005). Point-Based POMDP Algorithms: Improved Analysis and Implementation. In *Proceedings of the 21st conference on Uncertainty in artificial intelligence (UAI'05)*, pages 542–547.

Sondik, E. (1971). *The Optimal Control of Partially Observable Markov Decision Processes*. PhD thesis, Stanford University.

Sondik, E. (1978). The Optimal Control of Partially Observable Markov Processes over the Infinite Horizon: Discounted Costs. *Operations Research*, 26(2):282–304.

Sorg, J. and Singh, S. (2009). Transfer via Soft Homomorphisms. In *Proceedings of The Eighth International Conference on Autonomous Agents and Multiagent Systems (AAMAS'09)*, pages 741–748.

Spaan, M. T. J. and Vlassis, N. A. (2005). Perseus: Randomized Point-based Value Iteration for POMDPs. *Journal of Artificial Intelligence Research (JAIR)*, 24:195–220.

Sutton, R. and Tanner, B. (2004). Temporal-Difference Networks. In *Advances in Neural Information Processing Systems (NIPS'04)*, pages 1377–1384.

Sutton, R. and Tanner, B. (2005). TD(lambda) Networks: Temporal-Difference Networks With Eligibility Traces. In *Proceedings of the Twenty-Second International Conference on Machine Learning (ICML'05)*, pages 888–895.

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

Sutton, R. S., Mcallester, D., Singh, S., and Mansour, Y. (2000). Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Advances in Neural Information Processing Systems 12 (NIPS'00)*, volume 12, pages 1057–1063.

Syed, U., Bowling, M., and Schapire, R. E. (2008). Apprenticeship Learning using Linear Programming. In *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, pages 1032–1039.

Syed, U. and Schapire, R. (2008). A Game-Theoretic Approach to Apprenticeship Learning. In *Advances in Neural Information Processing Systems 20 (NIPS'08)*, pages 1449–1456.

Szepesvári, C. (2009). Reinforcement Learning Algorithms for MDPs – A Survey. *Technical report TR09-13, Department of Computing Science, University of Alberta*.

Szer, D. and Charpillet, F. (2006). Point-Based Dynamic Programming for DEC-POMDPs. In *Proceedings of the 21th National Conference on Artificial Intelligence (AAAI'06)*, pages 304–311.

Szer, D., Charpillet, F., and Zilberstein, S. (2005). MAA*: A Heuristic Search Algorithm for Solving Decentralized POMDPs. In *Proceedings of 21st Conference on Uncertainty in Artificial Intelligence (UAI'05)*.

Taylor, M. E. and Stone, P. (2009). Transfer Learning for Reinforcement Learning Domains: A Survey. *Journal of Machine Learning Research*, 10(1):1633–1685.

Tesauro, G. (1994). TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-level Play. *Neural Computing*, 6(2):215–219.

Theocharous, G., Mahadevan, S., and Kaelbling, L. P. (2005). Spatial and Temporal Abstractions in POMDPs Applied to Robot Navigation. *Technical report MIT-CSAIL-TR-2005-058, Massachusetts Institute of Technology*.

Theocharous, G., Rohanimanesh, K., and Mahadevan, S. (2001). Learning hierarchical partially observable markov decision processes for robot navigation. In *Proceedings of the IEEE Conference on Robotics and Automation (ICRA'01)*, pages 511–516.

Timmer, S. (2009). *Reinforcement Learning with History Lists*. PhD thesis, University of Osnabruck.

Toussaint, M., Charlin, L., and Poupart, P. (2008). Hierarchical POMDP Controller Optimization by Likelihood Maximization. In *Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence (UAI'08)*, pages 562–570.

Varakantham, P., Marecki, J., Yabu, Y., Tambe, M., and Yokoo, M. (2007). Letting Loose A SPIDER on A Network of POMDPs: Generating Quality Guaranteed Policies. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Aystems (AAMAS'07)*, pages 822–829.

Verma, D. and Rao, R. P. N. (2006). Goal-Based Imitation as Probabilistic Inference over Graphical Models. In *Advances in Neural Information Processing Systems 18 (NIPS'06)*.

Virin, Y., Shani, G., Shimony, S., and Brafman, R. (2007). Scaling Up: Solving POMDPs through Value Based Clustering. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI'07)*, pages 1290–1295.

Vlassis, N. and Toussaint, M. (2009). Model-free Reinforcement Learning as Mixture Learning. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML'09)*, pages 1081–1088.

Watkins, C. (1989). *Learning from Delayed Rewards*. PhD thesis, University of Cambidge, England.

Whiten, A. (2002). Imitation of Sequential and Hierarchical Structure in Action: Experimental Studies with Children and Chimpanzees. *Imitation in animals and artifacts*, pages 191–209.

Wiering, M. and Schmidhuber, J. (1997). HQ-Learning. *Adaptive Behavior*, 6(2):219–246.

Wierstra, D., Förster, A., Peters, J., and Schmidhuber, J. (2009). Recurrent Policy Gradient. *Logic Journal of IGPL*, 9:1–15.

Wierstra, D. and Schmidhuber, J. (2007). Policy Gradient Critics. In *Proceedings of the 18th European Conference on Machine Learning (ECML'07)*, pages 466–477.

Wiewiora, E. (2005). Learning Predictive Representations from a History. In *Proceedings of the 22nd International Conference on Machine Learning (ICML'05)*.

Wiewiora, E. (2007). *Modeling Probability Distributions with Predictive State Representations*. PhD thesis, The University of California at San Diego.

Williams, J. (2006). *Partially Observable Markov Decision Processes for Spoken Dialogue Management*. PhD thesis, Department of Engineering, University of Cambridge.

Williams, R. (1992). Simple Statistical Gradient-following Algorithms for Connectionist Reinforcement Learning. volume 8, pages 229–256.

Wingate, D. (2008). *Exponential Family Predictive Representations of State*. PhD thesis, The University of Michigan.

Wingate, D. and Seppi, K. (2005). Prioritization Methods for Accelerating MDP Solvers. *Journal of Machince Learning Research*, 6:851–881.

Wolf, T. (2009). Aircraft Collision Avoidance Using Monte Carlo Real-Time Belief Space Search. Master's thesis, Massachusetts Institute of Technology.

Wolfe, A. P. (2010a). *Paying Attention To What Matters: Observation Abstraction In Partially Observable Environments*. PhD thesis, University of Massachusetts.

Wolfe, B. (2009). *Modeling Dynamical Systems with Structured Predictive State Representations*. PhD thesis, The University of Michigan.

Wolfe, B. (2010b). Valid Parameters for Predictive State Representations. In *Eleventh International Symposium on Artificial Intelligence and Mathematics (ISAIM'10)*.

Yang, J., Xu, Y., and Chen, C. (1997). Human Action Learning via Hidden Markov Model. *IEEE Trans. on System, Man, and Cybernetics*, 27(1):34–44.

Yu, H. (2005). A Function Approximation Approach to Estimation of Policy Gradient for POMDP with Structured Policies. In *Proceeding of the 21st Conference Uncertainty in Artificial Intelligence (UAI'05)*, pages 642–657.

Yu, H. and Bertsekas, D. P. (2008). On Near Optimality of the Set of Finite-State Controllers for Average Cost POMDP. *Technical report LIDS 2689, Massachusetts Institute of Technology*.

Zhang, N. and Zhang, W. (2001). Speeding Up the Convergence Of Value Iteration in Partially Observable Markov Decision Processes. *Journal of Artificial Intelligence Research*, 14(1):29–51.

Zhou, R. and Hansen, E. (2001). An Improved Grid-based Approximation Algorithm for POMDPs. In *Proceedings of Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, pages 707–716.

Ziebart, B., Maas, A., Bagnell, J., and Dey, A. (2008). Maximum Entropy Inverse Reinforcement Learning. In *Proceedings of The Twenty-third AAAI Conference on Artificial Intelligence (AAAI'08)*, pages 1433–1438.