Planning in Decentralized POMDPs with Predictive Policy Representations

Abdeslam Boularias and **Brahim Chaib-draa**
Department of Computer Science and Software Engineering
Laval University, Canada, G1K 7P4
{boularias,chaib}@damas.ift.ulaval.ca

Abstract

We discuss the problem of policy representation in stochastic and partially observable systems, and address the case where the policy is a hidden parameter of the planning problem. We propose an adaptation of the Predictive State Representations (PSRs) to this problem by introducing tests (sequences of actions and observations) on policies. The new model, called the Predictive Policy Representations (PPRs), is more compact and uses less parameters than the usual representations, such as decision trees or Finite-State Controllers (FSCs). In this paper, we show how PPRs can be used to improve the performances of a point-based algorithm for DEC-POMDP.

Introduction

Partially Observable Markov Decision Processes (POMDPs) are a very popular model used for planning under state uncertainty (Smallwood & Sondik 1971). The success of this model is due to its simplicity and robustness at the same time. However, POMDPs suffer from a crucial problem known as *the curse of dimensionality*. In fact, the number of states grows exponentially w.r.t the number of variables describing the states. The Predictive State Representations (PSRs) (Littman, Sutton, & Singh 2001; Singh, James, & Rudary 2004) are an alternative representation of POMDPs, where the hidden states are replaced by sequences of actions and observations, leading to potentially more compact models.

While the problem of state representation has been receiving a significant amount of attention, there were less focus on the problem of policy representation. Decision trees are frequently used for finite horizon planning tasks. Most of Dynamic Programming (DP) algorithms for POMDPs use decision trees, whereas the number of policies evaluated at each planning step is exponential w.r.t the horizon and the number of observations. Another drawback of decision trees is that they cannot exploit potential recurrences that can arise in the policy. Finite State Controllers (FSCs) provide a more efficient policy model (Hansen 1998; Meleau *et al.* 1999). Contrary to decision trees, FSCs can be used to represent infinite horizon policies.

In many multiagent systems, the policy of a given agent is unknown for other agents. This problem is very similar to the problem of hidden states in POMDPs. From an external standpoint, an agent is seen as a dynamical system that responds to observations by executing actions according to some policy. Thus, we can exploit all the techniques that were originally developed for hidden states representation in order to represent hidden policies. Moreover, contrary to physical states, policy states (which are a kind of mental states) generally cannot be easily specified. PSRs are well adapted for this problem because they do not use explicit state representation. We propose in this paper an adaptation of PSRs for the policy representation purpose, and we prove that the new model, called the Predictive Policy Representations (PPRs), can represent any FSC model. We also show how we can use this representation to enhance the Point-Based Value Iteration (PBDP) (Szer & Charpillet 2006) algorithm for Decentralized POMDPs (Bernstein, Immerman, & Zilberstein 2002; Hansen, Bernstein, & Zilberstein 2004; Szer & Charpillet 2006).

A motivating example

Figure 1 represents a 3×3 grid world with two agents. If the goal of each agent is to meet the other one, then we can see that each agent must know the policy of the other in order to choose the best policy leading to a meeting point. If the agents are not communicating their observations, than each agent can only calculate a probability distribution over the policies of the other agent. Figure 2 represents a set of possible policies for a given agent. Using this type of representations leads to a low scalability of planning algorithms.

Background

POMDPs

Formally, a POMDP is defined by the following parameters: a finite set of hidden states S; a finite set of actions A; a finite set of observations Ω; a transition function

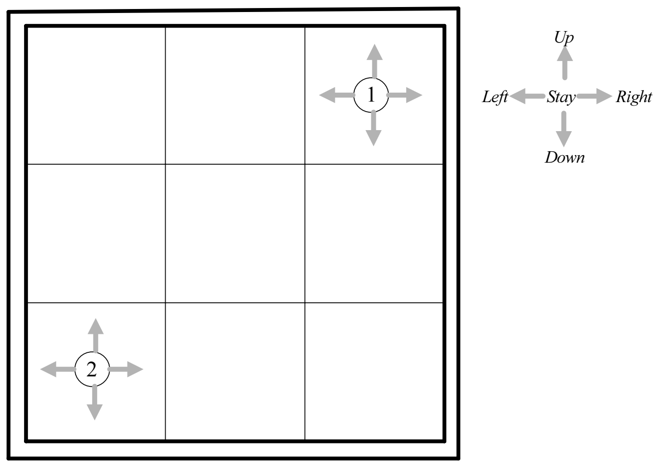

Figure 1: A multi-agent grid world environment.

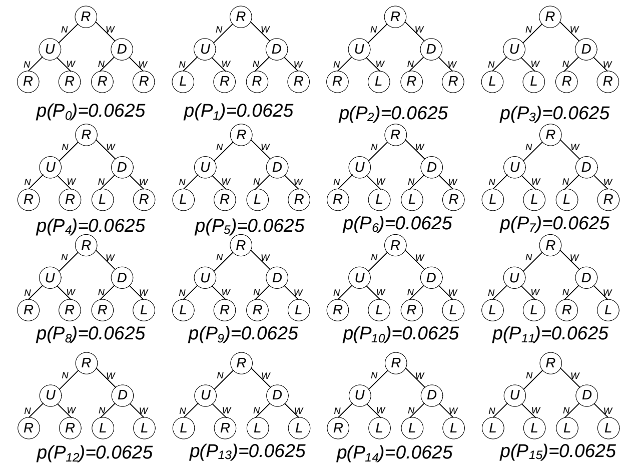

Figure 2: A probability distribution over a set of policies.

$P : S \times A \times S \to [0,1]$, such that $P(s'|s,a)$ is the probability that the agent will end up in state s' after taking action a in state s; an observation function $O : A \times S \times \Omega \to [0,1]$, such that $O(o|a,s')$ gives the probability that the agent receives observation o after taking action a and getting to state s'; an initial belief state b_0, which is a probability distribution over the set of hidden states S; and a reward function $R : S \times A \to \mathbb{R}$, such that $R(s,a)$ is the immediate reward received when the agent executes a in the state s. Additionally, there can be a discount factor, $\gamma \in [0,1]$, used to weigh less rewards received farther into the future.

The sufficient statistic in a POMDP is the belief state b, which is a vector of length $|S|$ specifying a probability distribution over hidden states. The elements of this vector, $b(s_i)$, specify the conditional probability of the agent being in state s_i, given the initial belief b_0 and the history (sequence of actions and observations) experienced so far.

Predictive State Representations (PSRs)

PSRs (Littman, Sutton, & Singh 2001) are an alternative model for representing partially observable environments without reference to hidden variables. The fundamental idea of PSRs is to replace the probabilities on states by probabilities on future scenarios, called *state tests*. A test $t = a^1o^1 \dots a^ko^k$ is an ordered sequence of actions and observations. The probability of t starting at step i is defined by:

$$Pr(t|h_i) = Pr(o_{i+1} = o^1, \dots o_{i+k} = o^k | h_i, a_{i+1} = a^1, \dots, a_{i+k} = a^k)$$

where $h_i = a^0o^0 \dots, a^io^i$ is the whole history until step i.

The *system-dynamics matrix D* is an infinite matrix where $\forall t,h : D(t,h) = Pr(t|h)$. This matrix can be seen as a general model, able to describe any discrete dynamical system, since it returns the probability of any event after any history. In practice, we can only generate a finite sub-matrix of D that should contain the same information as the full matrix. For a large category of dynamical systems, we can prove that the probability of any state test t depends only on the probabilities of few state tests, called *core state tests*, which constitute a sufficient statistic for the system.

We indicate the core state tests by $q^1, q^2, \dots q^N$. Q indicates the set of these tests, and $Pr(Q|h_j) = (Pr(q^1|h_j), Pr(q^2|h_j), \dots, Pr(q^N|h_j))^T$ is the probability vector for the core state tests at time step j, which is equivalent to the belief state in POMDPs. We have then:

$$Pr(t^i|h_j) = f_{t^i}(Pr(Q|h_j)) \qquad (1)$$

where f_{t^i} is a function associated to the test t^i, this function is independent of the history, and allows us to calculate the probability of t^i by using only the probabilities of Q. The Bayes update function for PSRs is given by:

$$Pr(q^i|h_jao) = \frac{Pr(ao(q^i)|h_j)}{Pr(ao|h_j)} = \frac{f_{ao(q^i)}(Q|h_j)}{f_{ao}(Q|h_j)} \qquad (2)$$

Predictive Policy Representations (PPRs)

In the Predictive Policy Representation (PPR) model that we introduce here, we define a new type of tests, called *policy tests*. A policy test can be seen as the dual of a state test, where the actions and observations roles are switched. The probability that a policy test $t = o^0, a^1, \dots o^{k-1}, a^k$ succeeds is given by:

$$Pr(t|h_i) = Pr(a_{i+1} = a^1, \dots, a_{i+k} = a^k | h_i, o_i = o^0, \dots, o_{i+k-1} = o^{k-1})$$

The history h_i ends with an action and not an observation as in PSRs, because the policy tests start with an observation. In fact, step i is the moment after executing a_i and before perceiving o_i. We also consider that all the histories start with a fictive observation o^* which has probability 1 (the default observation). A policy test can be seen as a type of question regarding what the agent will do when it perceives a specified sequence of observations. PPR is different from the environment state based representations, where the actions are related directly to the physical state (or belief state), and from the Finite-State Controllers (Hansen 1998), where internal states are defined, and the actions are chosen according to these states.

Much like the system dynamics matrix, the *policy matrix P* is defined by the infinite set of all possible policy tests and histories. This matrix is the largest model that can represent any discrete behavior, even if this behavior is not stationary. An entry $P(h_j,t^i)$ is given by $Pr(t^i|h_j)$, the probability that the actions indicated in the policy test t^i will be executed by the agent, such that the current history of the system is h_j and the future observations will be the observations of t^i.

We define also the set of *core policy tests* $Q = \{q^1,q^2,\dots q^N\}$, these tests are sufficient to determine the probability of any other policy test t^i:

$$Pr(t^i|h_j) = f_{t^i}(Pr(Q|h_j)) \tag{3}$$

where f_{t_i} is the function associated to t_i, and $Pr(Q|h_j) = (Pr(q^1|h_j),Pr(q^2|h_j),\dots,Pr(q^N|h_j))^T$.

Similarly, the update function of the core policy tests probabilities is given by:

$$Pr(q^i|h_j oa) = \frac{Pr(oaq^i|h_j)}{Pr(oa|h_j)} = \frac{f_{oaq^i}(Q|h_j)}{f_{oa}(Q|h_j)} \tag{4}$$

A Predictive Policy Representation is defined by the following parameters:

- Q, the core policy tests.
- $Pr(Q|\emptyset)$, the initial probabilities of the core policy tests.
- $\forall a \in A, \forall o \in O : f_{oa}$, the function associated to the policy test oa.
- $\forall a \in A, \forall o \in O, \forall q^i \in Q : f_{oaq^i}$, the function associated to the policy test oaq^i, composed by the test oa followed by the test q^i.

If the function f_{t^i} is linear, then it can be replaced by a vector m_{t^i}, and the update function becomes:

$$Pr(q^i|h_j oa) = \frac{Pr(oaq^i|h_j)}{Pr(oa|h_j)} = \frac{m^T_{oaq^i}Pr(Q|h_j)}{m^T_{oa}Pr(Q|h_j)} \tag{5}$$

The core policy tests of a linear PPR corresponds to a basis of the policy matrix P.

The PPR model functions as follows: we start by initializing the probabilities $Pr(Q|\emptyset)$ of the core policy tests. After observing o_{j+1} (or the default observation o^* if we did not yet executed any action, i.e. $h_j = \emptyset$), the probabilities of the core policy tests are used to calculate a distribution of probabilities over the next actions by:

$$Pr(a_{j+1} = a^i|h_j o_{j+1}) = m^T_{o_{j+1}a^i}Pr(Q|h_j) \tag{6}$$

We select an action a_{j+1} according to this distribution, we execute this action, and then we use this new information (o_{j+1},a_{j+1}) to update our core tests probabilities by using Equation 5. We add the event (o_{j+1},a_{j+1}) at the end of the history h_j, which becomes h_{j+1}, and we repeat the same process for the next steps.

The following theorem makes a comparison between the PPR and the stochastic Finite-State Controllers. FSCs are a popular model used to represent the infinite-horizon policies. FSCs are to POMDPs exactly what PPRs are to PSRs. Many policy representations can be seen as a special case of FSCs, such as the decision trees or the 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 FSC. However, at present we have no general conditions under which PPRs are more compact than FSCs, but 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 FSC.

A stochastic Finite-State Controller is a 3-tuple $\langle Q, \psi, \eta \rangle$, where Q is a finite set of controller states s^i (internal states), ψ is a mapping from Q to a probability distribution on A, *s.t.* $\psi(s_i,a_i)$ is the probability to choose the action a_j in the state s_i. η is a transition function, *s.t.* $\eta(s_i,a_i,o_i,s_{i+1})$ is the probability to transit from the state s_i to the state s_{i+1} when we execute the action a_i and receive the observation o_i.

Theorem 1. *Every stochastic Finite-State Controller can be represented by an equivalent linear PPR using at most the same number of parameters.*

Proof. This proof uses an idea introduced in (Singh, James, & Rudary 2004). Let P be the policy matrix corresponding to a given FSC. Since $P(h,t) = Pr(t|h) = \sum_{s \in Q} Pr(s|h)Pr(t|s)$, then we can decompose P into $P = FB$, where F is a $\infty \times |Q|$ matrix, defined by $F(h,s) = Pr(s|h)$, and B is a $|Q| \times \infty$ matrix, defined by $B(s,t) = Pr(t|s)$. F and B can be constructed by using the functions ψ and η. Since $rank(F) \leq |Q|$ and $rank(B) \leq |Q|$, then $rank(P) = rank(FB) \leq |Q|$. The number of core policy tests needed in the linear PPR model is at most equal to $|Q|$, since the core tests corresponds to linearly independent vectors in the matrix P. □

Figure 3 shows a simple example of deterministic Finite-State Controllers, which are a subclass of the stochastic controllers. Every state is labeled with the action to be executed in that state. This controller contains four different states, but can be represented with only two core policy tests: $t_1 = o_1 a_1$ and $t_2 = o_2 a_2$. The answers to these two tests are

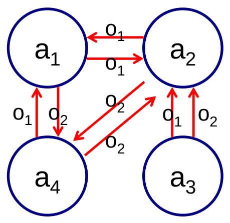

Figure 3: A deterministic four-states controller that can be represented with only two core policy tests $t_1 = o_1a_1$ and $t_2 = o_2a_2$.

sufficient to determine the state of the controller. For example, if we have $Pr(o_1a_1|h) = 1$ and $Pr(o_2a_2|h) = 0$, then we conclude that $Pr(o_2a_4|h) = 1, Pr(o_2a_4o_2a_2|h) = 1$, and so on (implicitly we are in the state of the action a_2), we have $Pr(o_2a_4|h) = Pr(t_1|h)(1 - Pr(t_2|h)) + (1 - Pr(t_1|h))(1 - Pr(t_2|h)) = 1 - Pr(t_2|h)$, and we can drive similar nonlinear prediction rules for every sequence.

In the remaining of this paper, we will see how we can use PPRs to represent policies in Decentralized POMDPs. However, adapting PPRs to dynamic programming algorithms requires a quite complicated construction of the *value vectors* (Boularias & Chaib-draa 2008), though the final algorithm is simple. Thus, we will consider a greatly simplified version of linear PPRs, where each policy test $t = a_jo_jq^{t-1}$ is a pair of action a_j and observation o_j, followed by a decision tree q^{t-1}, instead of a sequence of actions and observations. We will also treat all the tests as core tests, consequently, every test $t = a_jo_jq^{t-1}$ will be explicitly represented, and we will completely ignore the linear dependencies between tests.

Using PPRs in DEC-POMDP

Decentralized POMDPs

DEC-POMDPs, introduced in (Bernstein, Immerman, & Zilberstein 2002), are a straight generalization of POMDPs to multi-agent systems. Planning in DEC-POMDPs is generally centralized, but the execution is always decentralized: each agent chooses its actions according to its own local history and policy, independently of the other agents.

Formally, a DEC-POMDP with n agents is a tuple $\langle I, S, \{A_i\}, P, \{\Omega\}, O, R, T, \gamma\rangle$, where:

- I is a finite set of agents, indexed $1 \ldots n$.
- S is a finite set of states.
- A_i is a finite set of individual actions for agent i. $\vec{A} = \otimes_{i \in I} A_i$ is the set of joint actions, and $\vec{a} = \langle a_1, \ldots, a_n\rangle$ denotes a joint action.
- P is a transition function, $P(s'|s, \vec{a})$ is the probability that the system changes from state s to state s', when the agents execute the joint action $\vec{a}$.
- Ω_i is a finite set of individual observations for agent i. $\vec{\Omega} = \otimes_{i \in I} \Omega_i$ is the set of joint observations, and $\vec{o} = \langle o_1, \ldots, o_n\rangle$ denotes a joint observation.
- O is an observation function, $O(\vec{o}|s', \vec{a})$ is the probability that the agents observe $\vec{o}$ when the current state is s' 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) given to the action $\vec{a}$ in state s.
- T is the horizon of planning (the total number of steps).
- γ is a discount factor, generally used when T is infinite.

Planning algorithms for DEC-POMDPs aim to find the best joint policy of horizon T, which is a collection of several local policies, one for each agent. A local policy of horizon t for agent i, denoted by q_i^t, is a mapping from local histories of observations $o_i^1 o_i^2 \ldots o_i^t$ to actions in A_i. Usually, we use decision trees to represent the local policies, every node of the decision tree is labeled with an individual action a_i, and every arc is labeled with an individual observation o_i. For clarity, we consider in the remainder of this paper that we have only two agents, i and j, and all the results can be easily extended to the general case. The joint policy of horizon t for agents i and j is denoted by $q^t = \langle q_i^t, q_j^t\rangle$. We also use Q_i^t, Q_j^t to indicate the sets of local policies for agents i and j respectively, and Q^t for the set of joint policies. The *multiagent belief state* b_i for agent i contains a probability distribution over the states S, and another probability distribution over the current policies Q_j of agent j. $b_i(s, q_j)$ is the probability that the system is in the state s and the current policy of agent j is q_j.

Dynamic Programming for DEC-POMDPs

Dynamic Programming is by far the technique most used for solving multistage decision problems, where the optimal policies of horizon t are recursively constructed from the optimal sub-policies of horizon $t - 1$. This method has been widely used for finding optimal finite horizon policies for POMDPs since Smallwood *et al.* (Smallwood & Sondik 1971) presented the value iteration algorithm. Recently, Hansen *et al.* (Hansen, Bernstein, & Zilberstein 2004) proposed an interesting extension of the value iteration algorithm to the decentralized POMDPs, called Dynamic Programming Operator for DEC-POMDPs. We briefly review here the principal steps of this algorithm.

The expected discounted reward of a joint policy q^t, started from state s, is given recursively by *Bellman value function*:

$$V_{q^t}(s) = R(s, \vec{A}(q^t)) + \gamma \sum_{s' \in S} P(s'|s, \vec{A}(q^t)) \sum_{\vec{o} \in \vec{\Omega}} O(\vec{o}|s', \vec{A}(q^t)) V_{\vec{o}(q^t)}(s') \quad (7)$$

```
Input: $Q_i^{t-1}$, $Q_j^{t-1}$ and $V^{t-1}$;
$Q_i^t$, $Q_j^t \leftarrow$ fullBackup($Q_i^{t-1}$), fullBackup($Q_j^{t-1}$);
Calculate the value vectors $V^t$ by using $V^{t-1}$(Equation 10);
repeat
    remove the policies of $Q_j^t$ that are dominated (Table 1);
    remove the policies of $Q_j^t$ that are dominated (Table 1);
until no more policies in $Q_i^t$ or $Q_j^t$ can be removed ;
Output: $Q_i^t$,$Q_j^t$ and $V^t$;
```
Algorithm 1: Dynamic Programming for DEC-POMDPs (Hansen, Bernstein, & Zilberstein 2004).

where $\vec{A}(q^t)$ is the first joint action of the policy q^t (the root node), $\vec{o}$ is a joint observation, and $\vec{o}(q^t)$ is the sub-policy of q^t remaining after the action $\vec{A}(q^t)$ and the observation $\vec{o}$.

The value of an individual policy q_i^t, according to a belief state b_i, is given by the following function:

$$V_{q_i^t}(b_i) = \sum_{s\in S} \sum_{q_j^t \in Q_j^t} b_i(s, q_j^t) V_{\langle q_i^t, q_j^t \rangle}(s) \quad (8)$$

where $b_i(s, q_j^t)$ is the probability that the system is in the state s and the current policy of the other agent j is q_j^t. $\langle q_i^t, q_j^t \rangle$ denotes the joint policy made up of q_i^t and q_j^t.

The Dynamic Programming Operator (Algorithm 1) finds the optimal policies of horizon t, given the optimal policies of horizon $(t-1)$. V^t is the set of value vectors V_{q^t} corresponding to the joint policies of horizon t. First, we generate the sets Q_i^t, Q_j^t and by extending the policies Q_i^{t-1}, ${Q_j}^{t-1}$, and calculate V^t by using V^{t-1} in Equation 1, then we iteratively prune the *weakly dominated* policies of each agent. The pruning process stops when no more policies can be removed from Q_i^t or Q_j^t. A policy q_i^t is said weakly dominated if and only if:

$$\forall b_i \in \Delta(S \times Q_j^t), \exists q_i^{t\prime} \in Q_i^t - \{q_i\}\colon V_{q_i^{t\prime}}(b_i) \geq V_{q_i^t}(b_i) \quad (9)$$

In other words, whatever the belief b_i of agent i is, we can always find another policy $q_i^{t\prime}$ that has a least the same expected value in b_i as the policy q_i^t.

From Algorithm 1, we can see that the DP operator spends most of its time on determining the weakly dominated policies by checking the inequality (9) for every policy q_i^t. Usually, a linear program is used for this purpose. The time complexity of a linear program solver depends on the number of variables and constraints defined in the problem, so, it depends directly on the number of policies and the way we represent the beliefs over these policies.

Point Based Dynamic Programming

This later problem has been efficiently addressed with Point Based Dynamic Programming (PBDP) algorithm proposed in (Szer & Charpillet 2006). It makes use of top-down heuristic search to determine which belief points will be reached during the execution time, and constructs the best policy from leaves to root with DP, by keeping only the policies that are dominant in the reachable belief points. The most important difference between PBDP and exact DP is the fact that Equation (9) is checked only for a finite set of reachable belief points b_i. Therefore, the runtime of this algorithm is significantly small compared to the original DP algorithm. An approximate 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 & Zilberstein 2007) is a very fast algorithm that is close to PBDP, it is based on bounding the maximum number of policies kept in memory after each iteration. PPRs can be used with PBDP as well as with MBDP since the main difference these two algorithms is in the number of policies considered and not the method used to represent these policies. For our experiments, we implemented the PBDP algorithm (Szer & Charpillet 2006) with a random heuristic for selecting the belief points in Equation (9), without considering if these points are reachable or not. In fact, for the small problems used in DEC-POMDPs literature, we found that there is no improvement when we consider only the reachable belief points. Instead of spending lot of time to find reachable belief points, we consider a larger set of random belief points, and the optimal policies will be dominant in most of these points.

Point Based Dynamic Programming with PPRs

We propose to use the Predictive Policy Representations (PPRs) in PBDP algorithm. To do so, we have to redefine the belief state and the value vectors with PPRs. A belief state $b_i(s, a_j o_j q_j^{t-1})$ is the probability that the system is in the state s, and agent j will execute the action a_j, and if the observation of j will be o_j, then the next policy of j will be the decision tree q_j^{t-1}. The only difference between this definition and the usual definition of multi-agent belief states is that each tree q_j^t is factored at the first level and separated into several components (tests) $a_j o_j q_j^{t-1}$, while the subtrees q_j^{t-1} are represented explicitly. Generally, we have much less tests $a_j o_j q_j^{t-1}$ than policies q_j^t. This is particularly true just after the exhaustive backup step in Algorithm 1, since we will generate $|Q_j^t| = |A_j||Q_j^{t-1}|^{|\Omega_j|}$ new policies, while we will need only $|A_j||\Omega_j||Q_j^{t-1}|$ tests $a_j o_j q_j^{t-1}$ to represent all these policies. Consequently, the size of a belief state defined over tests can be exponentially smaller than the size of belief state defined over policies.

In order that a randomly generated belief over S and Q_j^t will be an accurate belief, we need only to guarantee that:

$$\sum_{s\in S} \sum_{q_j^t \in Q_j^t} b(s, q_j^t) = 1$$

But to guarantee that a randomly generated belief over S and the tests of Q_j^t policies will be an accurate belief (i.e. defines a distribution over states and policies), we should verify:

$$\forall o_j \in \Omega_j : \sum_{s \in S} \sum_{a_j \in A_j} \sum_{q_j^{t-1} \in Q_j^{t-1}} b(s, a_j o_j q_j^{t-1}) = 1$$

$$\forall a_j \in A_j, \forall o_j, o'_j \in \Omega_j :$$
$$\sum_{s \in S} \sum_{q_j^{t-1} \in Q_j^{t-1}} b(s, a_j o_j q_j^{t-1}) = \sum_{s \in S} \sum_{q_j^{t-1} \in Q_j^{t-1}} b(s, a_j o'_j q_j^{t-1})$$

Indeed, the sum $\sum_{s \in S} \sum_{q_j^{t-1} \in Q_j^{t-1}} b(s, a_j o_j q_j^{t-1})$ is the probability that agent j will execute the action a_j, and it must be the same for any next observation o_j. Contrary to decision trees, tests starting with the same action followed by different observations are not mutually exclusive events.

The value of a joint test $\langle a_i o_i q_i^{t-1}, a_j o_j q_j^{t-1} \rangle$ when we start from state s is:

$$V_{\langle a_i o_i q_i^{t-1}, a_j o_j q_j^{t-1} \rangle}(s) =$$
$$\Big(\sum_{s' \in S} Pr(s'|s, \langle a_i, a_j \rangle) Pr(\langle o_i, o_j \rangle | s', \langle a_i, a_j \rangle)\Big) R(s, \langle a_i, a_j \rangle)$$
$$+\gamma \sum_{s' \in S} Pr(s'|s, \langle a_i, a_j \rangle) Pr(\langle o_i, o_j \rangle | s', \langle a_i, a_j \rangle) V_{\langle q_i^{t-1}, q_j^{t-1} \rangle}(s')$$

The value of an individual policy q_i^t, according to a belief state b_i, is given by the following function:

$$V_{q_i^t}(b_i) =$$
$$\sum_{s \in S} \sum_{a_j \in A_j} \sum_{\substack{o_i \in \Omega_i \\ o_j \in \Omega_j}} \sum_{q_j^{t-1} \in Q_j^{t-1}} b_i(s, a_j o_j q_j^{t-1}) V_{\langle A(q_i^t) o_i \, \text{subtree}_{o_i}(q_i^t), a_j o_j q_j^{t-1} \rangle}(s)$$

where $A(q_i^t)$ is the first action of the tree q_i^t, and $\text{subtree}_{o_i}(q_i^t)$ is the subtree of q_i^t under the observation o_i.

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

Empirical Results

We implemented PBDP algorithm with both full decision trees and our modified version of PPRs, and we compared the performances of theses two approaches on three standard problems taken from DEC-POMDPs literature (Seuken & Zilberstein 2007): MA-Tiger, MABC, and The Meeting problem (with a 2×2 grid). The code of the implementation is written in C++, and the tests were performed on a 1.73 GHz Pentium M processor, with a RAM of 512 Mo. In our implementation, we relaxed the constraints on the belief points on tests, so some belief points may be 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.

Meeting	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8
Decision Trees	0	0	0.81	1.90	n.a.	n.a.	n.a.	n.a.
PPRs	0	0.81	1.79	2.78	3.78	4.78	5.78	6.78

MA-Tiger	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8
Decision Trees	-2	-4	5.19	4.80	n.a.	n.a.	n.a.	n.a.
PPRs	-2	-4	5.19	4.39	4.21	2.27	0.41	-1.5

MABC	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8
Decision Trees	1	2	2.90	3.89	4.79	5.69	6.59	7.49
PPRs	1	2	2.99	3.80	4.79	5.60	6.50	7.49

Table 1: The values of the optimal policies returned by PBDP using decision trees and PPRs to represent policies.

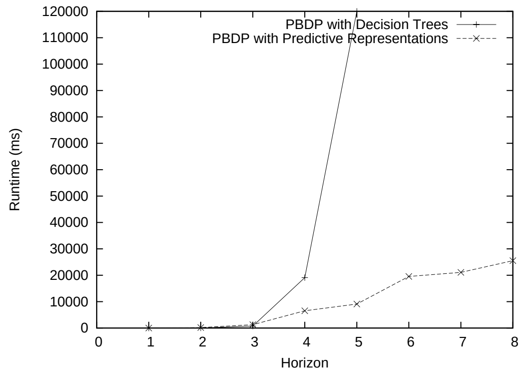

Figure 4: The effective runtime of the PBDP algorithm as a function of the horizon, with the MA-Tiger problem.

Figures 4, 5 and 6 show the runtime of the PBDP for different horizons. As expected, we can see that the runtime of PBDP is significantly reduced when we use a predictive representation of the policies. This is explained by the fact that the belief points defined over tests are smaller than the belief points defined over decision trees. Consequently, the value of a given policy can be calculated with fewer operations, and the dominance test of Equation (9) is performed in shorter time. We noticed also that most of this computational gain is made just after the full backup, where the belief points in decision trees approaches contain a number of policies which is exponential w.r.t. the number of observations.

Table 1 shows the values of of the policies returned by PBDP are almost the same for the two policy representations. The values with the PPR approach are slightly suboptimal because the belief points where under constrained, and since we used a limited set of belief points, some optimal policies were dominated in all these points.

Conclusion and Future Work

In many multiagent systems, the uncertainty of an agent is not only about the environment states, but also about the

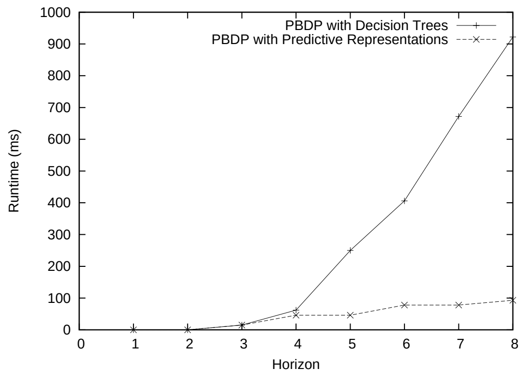

Figure 5: The effective runtime of the PBDP algorithm as a function of the horizon, with the MABC problem.

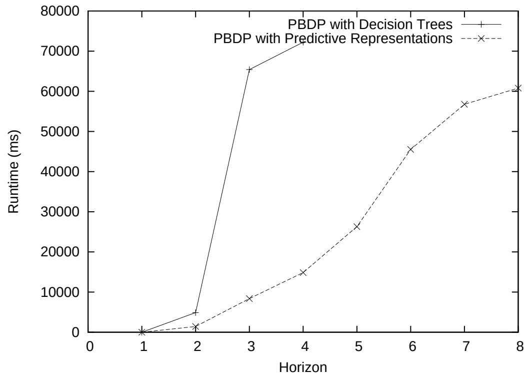

Figure 6: The effective runtime of the PBDP algorithm as a function of the horizon, with the Meeting problem.

policies of other agents. In this paper, we proposed a new model to represent the agent's belief state based on the predicting other agents future actions. The advantage of this model, called Predictive Policy Representations (PPRs), is that the agent uses only the minimal and sufficient amount of data to represent its beliefs. We compared the computational performance of a point based algorithm for DEC-POMDP using decisions trees, with the same algorithm using a simplified version of PPRs, and the preliminary results are promising. Based on these results, we target to develop a new Dynamic Programming algorithm, using the original definition of the PPR model and exploiting the potential dependencies between different tests to reduce even more the dimensionality of the belief points.

References

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.

Boularias, A., and Chaib-draa, B. 2008. Exact dynamic programming for decentralized pomdps with lossless policy compression. In *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS'08)*. To appear.

Hansen, E. A.; Bernstein, D. S.; and Zilberstein, S. 2004. Dynamic programming for partially observable stochastic games. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI'04)*, 709–715.

Hansen, E. A. 1998. Solving pomdps by searching in policy space. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI'98)*, 211–219.

Littman, M.; Sutton, R.; and Singh, S. 2001. Predictive representations of state. In *Advances in Neural Information Processing Systems 14 (NIPS'02)*, 1555–1561.

Meleau, N.; Kim, K. E.; Kaelbling, L. P.; and Cassandra, A. R. 1999. Solving pomdps by searching in the space of finite policies. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI'99)*, 427–436.

Seuken, S., and Zilberstein, S. 2007. Improved Memory-Bounded Dynamic Programming for Decentralized POMDPs. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence (UAI'07)*.

Singh, S.; James, M. R.; and Rudary, M. R. 2004. Predictive state representations: A new theory for modeling dynamical systems. In *Uncertainty in Artificial Intelligence: Proceedings of the 20th conference (UAI'04)*, 512–519.

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.

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)*, 304–311.