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 potentially more compact than the other 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

Decision making under state uncertainty is one of the greatest challenges in artificial intelligence. State uncertainty is a direct result of stochastic actions and noised, or aliased, observations. Partially Observable Markov Decision Processes (POMDPs) provide a powerful Bayesian model to solve this problem (Smallwood and Sondik 1971). In this model, the state is represented by a probability distribution over all the possible states, that we call a *belief state*. The complexity of POMDPs algorithms, which is proved to be PSPACE-complete (Papadimitriou and Tsitsiklis 1987), depends heavily on the dimension of the belief state. During the last two decades, significant efforts have been devoted to developing fast algorithms for large POMDPs, and nowadays, even problems with a thousand of states can be solved within a few seconds (Virin et al. 2007).

The rise of applications requiring cooperation between different agents, like robotic teams, distributed sensors and communication networks, has made the presence of many decision makers a key challenge for building autonomous agents. For this purpose, a generalization of POMDPs to multi-agent domains, called DEC-POMDPs (Decentralized POMPDs), was introduced in (Bernstein, Immerman, and Zilberstein 2002), and since then, this framework has been receiving a growing amount of attention. This research is basically motivated by the fact that many real world problems need to be formalized as DEC-POMDPs, while planning with DEC-POMDPs is NEXP-complete (Bernstein, Immerman, and Zilberstein 2002), and even finding ε-optimal solutions is NEXP-hard (Rabinovich, Goldman, and Rosenschein 2003).

Finding good solutions to DEC-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 policies of the other agents. Consequently, the dimensionality of the policy space is a crucial factor in the scalability of DEC-POMDPs algorithms. In this paper, we propose a new method for scaling up Point Based Dynamic Programming (PBDP) algorithm for DEC-POMDPs (Szer and Charpillet 2006), based on a compact representation the policy 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. This method is an adaptation to DEC-POMDPs of another approach that was originally proposed to reduce the state space dimensionality in POMDPs, and which is known as the Predictive State Representations (PSRs) (Littman, Sutton, and Singh 2001b). In PSRs, states are replaced by sequences of actions and observations that have linearly independent probabilities. Similarly, a poll of policies can be represented by a smaller set of sequences that have linearly independent probabilities of occurring in these policies.

The remainder of this paper is structured in the following manner. We briefly review POMDPs and PSRs background in Section 2, and introduce the Predictive Policy Representations in Section 3. In Section 4, we recall DEC-POMDPs formalism and Dynamic Programming algorithms for DEC-POMDPs. In Section 5, we describe how to accelerate these algorithms by using predictive representations. Section 6 presents some empirical results of this method. We conclude this paper with a discussion in Section 7.

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 each agent must know the policy of the other in order to choose the best policy leading to a meeting point. Usually, we use decision

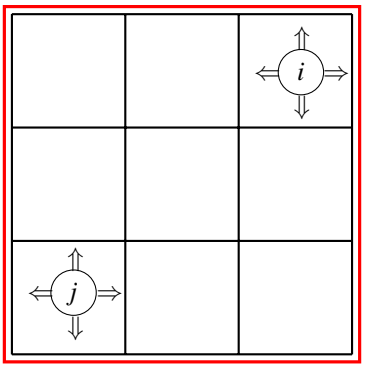

Figure 1: A multi-agent grid world environment.

trees to represent policies, every node of the decision tree is labeled with an action a_i, and every arc is labeled with an observation o_i. Ideally, every agent should know exactly the actions of the other agent, but unfortunately, this cannot be achieved when the agents are unable to communicate. In fact, each agent knows exactly the policy of the other agent at the beginning, and the first action of each agent can be easily predicted, since it corresponds to the first node of every local policy tree. But then, the next actions depend on the local observations perceived by each agent, and without communication, we can only consider a probability distribution over the actions (or the remaining subtrees) of the other agent. This distribution for a given agent corresponds to a distribution over the histories of observations of the other agent. Figure 2 represents a set of possible policies for a given agent. We can see that the number of policies grows double exponentially with respect to the number of observations and the length of histories (the planning horizon). Using decision trees to represent policies leads to a low scalability of the planning algorithms. The main idea of PSRs is to use a probability distribution over the possible sequences of actions and observations instead of decision trees. The number of sequences grows exponentially with respect to the planning horizon, but only polynomially with respect to the number of observations.

Background

Partially Observable Markov Decision Processes

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 $P : S \times A \times S \rightarrow [0,1]$, 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 $O : A \times S \times \Omega \rightarrow [0,1]$, 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

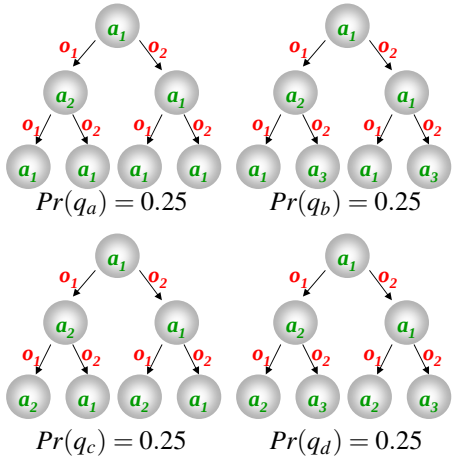

Figure 2: A probability distribution over a set of policies.

$R : S \times A \rightarrow \mathbb{R}$, such that $R(s,a)$ is the immediate reward received when the agent executes action a in 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 size $|S|$ specifying a probability distribution over the 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 (Littman, Sutton, and Singh 2001a) 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 *tests*. A test $t = a^1 o^1 \ldots a^k o^k$ is an ordered sequence of actions and observations. The probability of t starting at step i is defined by:

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

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

For a large category of dynamical systems, we can prove that the probability of any test t depends only on the probabilities of a few tests, called *core tests*, which constitute a sufficient statistic for the system. We indicate the core tests by $q^1, q^2, \ldots q^N$. Q indicates the set of these tests, and $Pr(Q|h_j) = (Pr(q^1|h_j), Pr(q^2|h_j), \ldots, Pr(q^N|h_j))^T$ is the probability vector for the core tests at time step j, which is equivalent to the belief state in POMDPs. For any test t^i, we have:

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

where f_{t^i} is a function associated to the test t^i, this function is independent of the history h_j, and allows us to calculate

the probability of t^i by using only the probabilities of Q at history h_j. After executing an action a and perceiving an observation o, the following Bayes function is used to update the probabilities of the core tests:

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

Predictive Policy Representations (PPRs)

In PPRs, the roles of the actions and observations are switched. The probability that a test $t = o^0, a^1, \ldots o^{k-1}, a^k$ succeeds is given by:

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

The history h_i ends with an action and not an observation as in PSRs, because tests in PPRs 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 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 environment state based representations, where the actions are related directly to physical states (or belief states), and from Finite-State Controllers (Hansen 1998), where internal states are defined, and the actions are chosen according to these states.

After executing an action a and perceiving an observation o, the following Bayes function is used to update the probabilities of the core tests:

$$Pr(q^i|h_j oa) = \frac{Pr(oaq^i|h_j)}{Pr(oa|h_j)} = \frac{f_{oaq^i}(Pr(Q|h_j))}{f_{oa}(Pr(Q|h_j))} \quad (3)$$

We can see from this equation that to calculate the new probability of core test q after executing a and perceiving o, we need only to know for the probabilities $Pr(oaq^i|h_j)$ and $Pr(oa|h_j)$ of the tests oa and oaq. These probabilities can be calculated from the probabilities of core tests probabilities $Pr(Q|h_j)$.

A Predictive Policy Representation is defined by the following parameters:

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

$$Pr(q^i|h_j oa) = \frac{Pr(oaq^i|h_j)}{Pr(oa|h_j)} = \frac{m^T_{oaq^i} Pr(Q|h_j)}{m^T_{oa} Pr(Q|h_j)} \quad (4)$$

The vector $Pr(Q|h_j)$ of core tests probabilities is used to calculate the probability of selecting action a after perceiving observation o. Indeed, we have: $Pr(oa|h_j) = m^T_{oa} Pr(Q|h_j)$

The *policy matrix* P is defined by the infinite set of all possible tests and histories. An entry $P(h_j, t^i)$ is given by $Pr(t^i|h_j)$, the probability that the actions indicated in the 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 indicated in t^i. The core tests of a linear PPR corresponds to a basis of the policy matrix P.

The following theorem makes a comparison between PPR and Stochastic Finite-State Controllers. FSCs are a popular model used to represent 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 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 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 $\langle S, \psi, \eta \rangle$, where S is a finite set of controller states s^i (internal states, not to be confused with the environment states), ψ is a mapping from S 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 transformed to an equivalent linear PPR using at most the same number of parameters.*

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

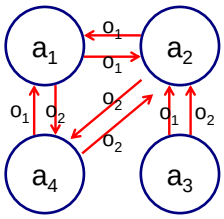

Figure 3: A deterministic four-states controller that can be represented with only two core tests $t_1 = o_1a_1$ and $t_2 = o_2a_2$.

trollers. 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: $t_1 = o_1a_1$ and $t_2 = o_2a_2$. The answers to these two tests are 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 derive 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 and Chaib-draa 2008), though the final algorithm is simple. Thus, we will consider a greatly simplified version of linear PPRs, where each test $a_jo_jq^{t-1}$ is a pair of action a_j and observation o_j, followed by a full decision tree q^{t-1}, instead of one sequence of actions and observations. We will also treat all the tests as core tests, consequently, every test $a_jo_jq^{t-1}$ 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, and 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 $1 \dots n$.
- 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 $\vec{a} = \langle a_1, \dots, a_n\rangle$ 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 $\vec{o} = \langle o_1, \dots, o_n\rangle$ 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 $o_i^1o_i^2 \dots o_i^t$ to actions in A_i. For clarity in the remainder of this paper, we consider 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 *multi-agent belief state* b_i for agent i contains a probability distribution over the system 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 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 and Sondik 1971) presented the value iteration algorithm. Recently, (Hansen, Bernstein, and Zilberstein 2004) proposed an interesting extension of the value iteration algorithm to decentralized POMDPs, called Dynamic Programming Operator for DEC-POMDPs. We review here briefly 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*:

$$V_{q^t}(s) = R(s,\vec{A}(q^t)) + \gamma \sum_{s'\in S} P(s'|s,\vec{A}(q^t)) \sum_{\vec{o}\in\vec{\Omega}} O(\vec{o}|s',\vec{A}(q^t)) V_{\vec{o}(q^t)}(s') \quad (5)$$

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 below the root node and the observation $\vec{o}$.

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 5);
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, and Zilberstein 2004).

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

$$V_{q_i^t}(b_i) = \sum_{s \in S} \sum_{q_j^t \in Q_j^t} b_i(s, q_j^t) V_{\langle q_i^t, q_j^t \rangle}(s) \quad (6)$$

where $\langle q_i^t, q_j^t \rangle$ denotes the joint policy made up of q_i^t and q_j^t, $V_{\langle q_i^t, q_j^t \rangle}(s)$ is given by equation 1.

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, the sets Q_i^t, Q_j^t are generated by extending the policies of Q_i^{t-1}, ${Q_j}^{t-1}$, and V^t are calculated by using V^{t-1} in equation 1, then the *weakly dominated* policies of each agent are iteratively pruned. 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 to be weakly dominated if and only if:

$$\forall b_i \in \Delta(S \times Q_j^t), \exists q_i^{t\prime} \in Q_i^t - \{q_i^t\} \colon V_{q_i^{t\prime}}(b_i) \geq V_{q_i^t}(b_i) \quad (7)$$

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 (7) 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 by (Szer and 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 inequality (7) 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 and Zilberstein 2007) is a 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 between these two algorithms is in the number of policies considered and not the method used to represent these policies. For our experiments, we implemented PBDP algorithm (Szer and Charpillet 2006) with randomly generated belief points in inequality (7), 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 significant 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, 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.

Point Based Dynamic Programming with PPRs

We propose to use the Predictive Policy Representations in order to reduce the dimensionality of belief points in PBDP algorithm. To do so, we have to redefine the belief state and the value vectors with PPRs. A belief state $b_i(s, a_j o_j q_j^{t-1})$ for agent i is the probability that the system is in 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 q_j^{t-1} (q_j^{t-1} is a decision tree). 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) $a_j o_j q_j^{t-1}$, a_j is the first action (root) of the policy q_j^t, q_j^{t-1} is the subtree of q_j^t under action a_j and observation o_j. So, for each policy q_j^t, we have $|\Omega_j||Q_j^{t-1}|$ corresponding tests. Generally, every test is used in several policies, and the number of tests is much smaller than the number of policies. This is particularly true right after the exhaustive backup step of Algorithm 1, since $|Q_j^t| = |A_j||Q_j^{t-1}|^{|\Omega_j|}$ new policies are generated, while we need only $|A_j||\Omega_j||Q_j^{t-1}|$ tests to represent all these policies. Consequently, the size of a belief state defined over tests can be exponentially smaller than the size of a 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 $A_j\times\Omega_j\times Q_j^{t-1}$ (the tests used to represent Q_j^t policies) will be an accurate belief (i.e. defines a distribution over states and policies), we should verify:

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

$$\forall a_j\in A_j,\forall o_j,o_j'\in\Omega_j:$$

$$\sum_{s\in S}\sum_{q_j^{t-1}\in Q_j^{t-1}} b(s,a_jo_jq_j^{t-1})=\sum_{s\in S}\sum_{q_j^{t-1}\in Q_j^{t-1}} b(s,a_jo_j'q_j^{t-1})$$

In fact, the sum $\sum_{s\in S}\sum_{q_j^{t-1}\in Q_j^{t-1}} b(s,a_jo_jq_j^{t-1})$ 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 which are always mutually exclusive, the tests starting with the same action and followed by different observations are not mutually exclusive.

The value of a joint test $\langle a_io_iq_i^{t-1},a_jo_jq_j^{t-1}\rangle$ started from state s is:

$$V_{\langle a_io_iq_i^{t-1},a_jo_jq_j^{t-1}\rangle}(s)=$$

$$\sum_{s'\in S} Pr(s'|s,\langle a_i,a_j\rangle)Pr(\langle o_i,o_j\rangle|s',\langle a_i,a_j\rangle))R(s,\langle a_i,a_j\rangle)$$

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

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

$$V_{q_i^t}(b_i)=$$

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

where $A(q_i^t)$ is the first action of the tree q_i^t, and $\mathrm{subtree}_{o_i}(q_i^t)$ 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 performance of theses two approaches on three standard problems taken from DEC-POMDPs literature (Seuken and 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 experiments 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 tests belief points, 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.

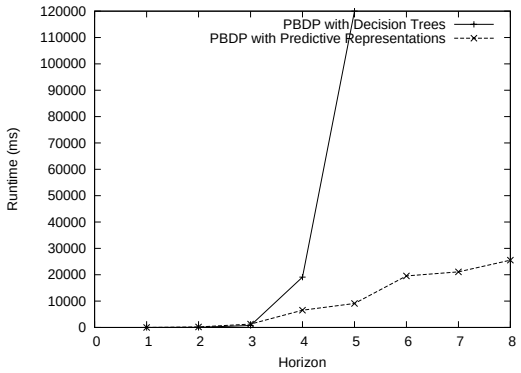

Figure 4: The effective runtime of the PBDP algorithm as a function of the horizon, with the MA-Tiger problem.

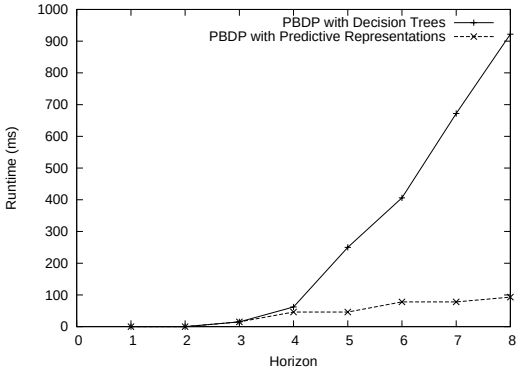

Figure 5: The effective runtime of the PBDP algorithm as a function of the horizon, with the MABC problem.

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 inequality (7) is performed in a shorter time. We noticed also that most of this computational gain is made right after the full backup, where the belief points in decision trees approach 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

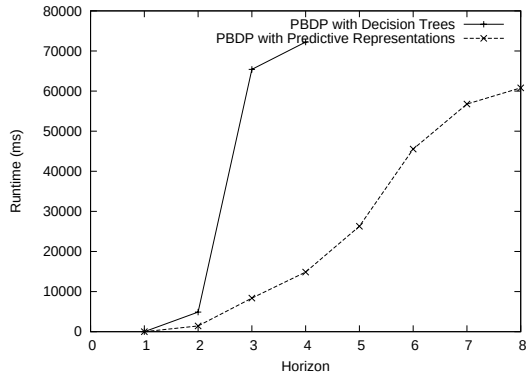

Figure 6: The effective runtime of the PBDP algorithm as a function of the horizon, with the Meeting problem.

Meeting	t=2	t=3	t=4	t=5	t=6	t=7	t=8
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	t=2	t=3	t=4	t=5	t=6	t=7	t=8
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	t=2	t=3	t=4	t=5	t=6	t=7	t=8
Decision Trees	2	2.90	3.89	4.79	5.69	6.59	7.49
PPRs	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.

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 policies of other agents. In this paper, we proposed a new model to represent the agent's belief state based on predicting other agents future actions. The advantage of this model, called Predictive Policy Representations (PPRs), is that agents uses only a minimal and sufficient amount of data to represent their beliefs. We compared the computational performance of a point based algorithm for DEC-POMDP using decisions trees to the performance of 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.; 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)*, 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. 2001a. Predictive representations of state. In *Advances in Neural Information Processing Systems 14 (NIPS'02)*, 1555–1561.

Littman, M.; Sutton, R.; and Singh, S. 2001b. Predictive Representations of State. In *Advances in Neural Information Processing Systems 14 (NIPS'01)*, 1555–1561.

Papadimitriou, C., and Tsitsiklis, J. 1987. The Complexity of Markov Decision Process. *Mathematics of Operations Research* 12(3):441–450.

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)*, 1102–1103.

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.

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)*, 1290–1295.