Exact Dynamic Programming for Decentralized POMDPs with Lossless Policy Compression

Abdeslam Boularias and **Brahim Chaib-draa**
Computer Science & Software Engineering Dept.
Laval University, Quebec G1k 7p4, CANADA
{boularias,chaib}@damas.ift.ulaval.ca

Abstract

High dimensionality of belief space in DEC-POMDPs is one of the major causes that makes the optimal joint policy computation intractable. The belief state for a given agent is a probability distribution over the system states and the policies of other agents. Belief compression is an efficient POMDP approach that speeds up planning algorithms by projecting the belief state space to a low-dimensional one. In this paper, we introduce a new method for solving DEC-POMDP problems, based on the compression of the policy belief space. The reduced policy space contains sequences of actions and observations that are linearly independent. We tested our approach on two benchmark problems, and the preliminary results confirm that Dynamic Programming algorithm scales up better when the policy belief is compressed.

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 & 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 & 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, & 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, & Zilberstein 2002), and even finding ε-optimal solutions is NEXP-hard (Rabinovich, Goldman, & 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 Dynamic Programming for DEC-POMDPs, based on a lossless compression 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, & Singh 2001). 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.

Related Work

A brute force solution for DEC-POMDPs consists in performing an exhaustive search in the space of joint policies (Bernstein, Immerman, & Zilberstein 2002), but this approach is almost useless in practice, even for the smallest domains. In fact, the search should focus only on the policies that are likely to be dominant. There are two main approaches for finding these policies: top-down heuristics and bottom-up dynamic programming. MAA* was the first

algorithm to use a top-down heuristic (Szer, Charpillet, & Zilberstein 2005). It is an adaptation of A* using a heuristic evaluation function to prune dominated nodes, where the nodes correspond to joint policies. A big disadvantage of such top-down search approaches is that the starting point should be known in advance. On the other hand, Dynamic Programming (DP) algorithm, proposed in (Hansen, Bernstein, & Zilberstein 2004), consists in constructing the optimal policies from leaves to root by using iterated policy elimination. DP algorithm can solve problems that are unfeasible with the 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 with Point Based Dynamic Programming (PBDP) algorithm proposed in (Szer & Charpillet 2006). PBDP makes use of top-down heuristic search to determine which belief points will be reached during the execution, and constructs the best policy from leaves to root with DP, by keeping only the policies that are dominant in the reachable points. In the same vein, Memory Bounded Dynamic Programming (MBDP) is an algorithm that has been proposed recently in (Seuken & Zilberstein 2007) and which is close to PBDP, it is based on bounding the maximum number of policies kept in memory after each iteration.

All these algorithms are based on reducing the number of policies to be evaluated or kept in memory. Alternatively, we can preserve the original policies space, and use a more compact representation of the policies. In decision trees, policies are constructed by combining multiple sequences of actions and observations, and the same sequences can be replicated in different policies. The sequential representation takes advantage of this characteristic: all the possible sequences of a given length are represented explicitly, and each policy is represented by a binary vector that indicates which sequences are contained in this policy. This method has been applied efficiently to DEC-POMDPs in (Aras, Dutech, & Charpillet 2007), the proposed algorithm uses a mixed integer linear program to find the optimal policies, where each variable corresponds to a sequence. However, there is no guarantee that the number of sequences will not exceed the number of policies, this can happen when we have a few policies with large horizons.

Decentralized POMDPs

DEC-POMDPs, introduced in (Bernstein, Immerman, & Zilberstein 2002), are a straight generalization of POMDPs to multi-agent systems. 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 for executing action $\vec{a}$ in state s.
- T is the horizon of planning (the total number of steps).
- γ is a discount factor.

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 $o_i^1 o_i^2 \dots o_i^t$ to actions in A_i. Usually, we use decision trees to represent local policies, 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. To ease notations, we consider in the remainder of this paper that we have only two agents, i and j, all the results can be easily extended to the general case. A 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.

Since the states are partially observable, agents should choose their actions according to a belief on the current state. The *belief state*, denoted by the vector b, is a probability distribution over the different states. In the single-agent context, the belief state is a sufficient statistic for making optimal decisions, but in multi-agent systems, the reward given to an individual action depends on all the actions taken by the other agents at the same time; the belief state is then no longer sufficient for making optimal decisions. In order to choose optimally its action, an agent should take into account which actions 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 use of communication. In fact, the joint policy is provided to all the agents, and the first joint action can easily be predicted, since we just have to look at 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 agents. This distribution is what we call a *multi-agent belief state*. The 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. Notice that $b_i(s, q_j)$ is the probability that the system is in state s and the current policy of agent j is q_j.

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 1);
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).

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 & Sondik 1971) presented the value iteration algorithm. Recently, (Hansen, Bernstein, & 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*:

$$V_{q^t}(s) = R(s,\vec{A}(q^t)) + \gamma \sum_{s' \in S} P(s'|s,\vec{A}(q^t)) \sum_{\vec{o} \in \vec{\Omega}} O(\vec{o}|s',\vec{A}(q^t)) V_{\vec{o}(q^t)}(s') \quad (1)$$

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 $\vec{o}$.

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

$$V_{q_i^t}(b_i) = \sum_{s \in S} \sum_{q_j^t \in Q_j^t} b_i(s,q_j^t) V_{\langle q_i^t, q_j^t \rangle}(s) \quad (2)$$

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 prune. 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:

$$\forall b_i \in \Delta(S \times Q_j^t), \exists q_i^{t\prime} \in Q_i^t - \{q_i^t\} : V_{q_i^{t\prime}}(b_i) \geq V_{q_i^t}(b_i) \quad (3)$$

minimize: ε
subject to:

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

$$\forall s \in S, \forall q_j^t \in Q_j^t :$$

$$0 \leq b_i(s,q_j^t) \leq 1$$

$$\forall q_i^{t\prime} \in Q_i^t - \{q_i^t\} :$$

$$\sum_{s \in S} \sum_{q_j^t \in Q_j^t} b_i(s,q_j^t)[V^t_{\langle q_i^t, q_j^t \rangle}(s) - V^t_{\langle q_i^{t\prime}, q_j^t \rangle}(s)] + \varepsilon > 0$$

Table 1: The linear program used to check if a policy q_i^t is dominated or not (Hansen, Bernstein, & Zilberstein 2004).

Policy Space Compression

Motivation

The main problem with DEC-POMDPs, compared to POMDPs, is the dimensionality of the policy space. In fact, the number of policies grows double exponentially with respect to the planning horizon and the number of observations. If we get $|Q_i^{t-1}|$ policies for agent i at step $t-1$, then $|A_i||Q_i^{t-1}|^{|\Omega_i|}$ new policies will be created at step t.

This *curse of dimensionality* has dramatic consequences on both time and space complexity of the DEC-POMDPs algorithms. From Algorithm 1, we can see that the dynamic programming operator spends most of its time determining the weakly dominated policies by checking the inequality (3) for every policy q_i^t. The usual approach for performing this test is to use the linear program of Table 1. The objective function to be minimized is defined by ε, which is the greatest difference between the value of q_i^t and the value of any other policy $q_i^{t\prime}$ over the belief space. If $\varepsilon \geq 0$ then the policy q_i^t is dominated and should be removed, and if $\varepsilon < 0$, then there is some region in $\Delta(S \times Q_j^t)$ where q_i^t is dominant. The variables are ε and the probabilities $b(.,.)$ of the multi-agent belief state, so there are $|S||Q_j^t| + 1$ variables.

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. However, the main problem of Dynamic Programming (Algorithm 1) remains the size of the memory space required to represent the value vectors V^t for each joint policy. The memory space required to represent these vectors at step t is $|Q_i^t||Q_j^t||S|$ floats. Indeed, this algorithm runs out of memory several iterations before running out of time.

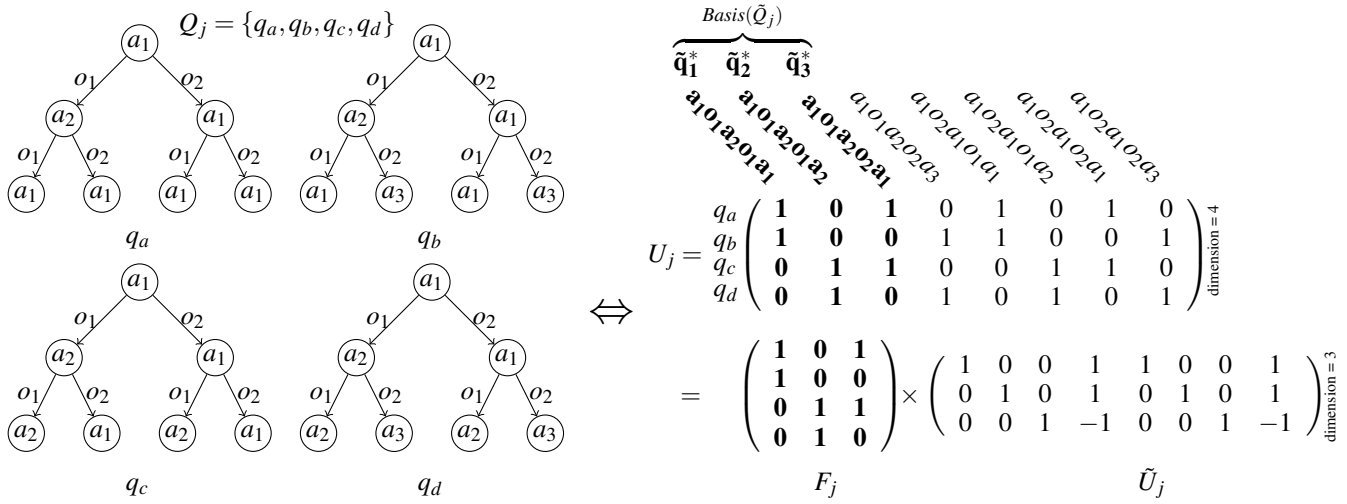

Figure 1: Reducing the policy space dimensionality.

To alleviate these problems, we need to use a more compact technique to represent the belief of agent i on the policies of agent j, instead of the naive probability distribution over all the policies Q_j^t. We can exploit the structure of the policies, and find some *features* $\tilde{Q}_j^t = \{\tilde{q}_1^t, \ldots, \tilde{q}_{|\tilde{Q}_j^t|}^t\}$ which constitute a *sufficient information*: each point in $\Delta Q^t{}_j$ will correspond to a point in $\Delta\tilde{Q}_j^t$, and vice versa. We should also guarantee that $|\tilde{Q}_j^t| \leq |Q_j^t|$ and that most of the time $|\tilde{Q}_j^t| \ll |Q_j^t|$. A policy is a collection of sequences of actions and observations. A sequence $\tilde{q}_j^t$ of horizon t for agent j is an ordered list of t individual actions and $t-1$ individual observations $(a_j^1, o_j^1, \ldots, o_j^{t-1}, a_j^t)$. A joint sequence $\tilde{q}^t$ of horizon t is a couple $\langle \tilde{q}_i^t, \tilde{q}_j^t \rangle$ where $\tilde{q}_i^t$ is an individual sequence for agent i and $\tilde{q}_j^t$ is an individual sequence for agent j, a joint sequence can also be seen as one ordered list of joint actions and observations. The set Q_j^t can be completely replaced, without any loss of information, by a matrix U_j^t called *the outcome matrix* (Singh, James, & Rudary 2004). Each row of U_j^t will correspond to a policy $q_j^t \in Q_j^t$ and each column will correspond to a sequence $\tilde{q}_j^t$. $U_j^t(q_j^t, \tilde{q}_j^t)$ is defined as the probability that agent j will execute the actions of $\tilde{q}_j^t$ if the observations of $\tilde{q}_j^t$ occur, such that the actual policy of agent j is q_j^t. Since the policies are deterministic, we have $U_j^t(q_j^t, \tilde{q}_j^t) = 1$ if the sequence $\tilde{q}_j^t$ appears in the policy q_j^t, and $U_j^t(q_j^t, \tilde{q}_j^t) = 0$ else. Figure 1 shows a set Q_j^t containing 4 individual policies for agent j: q_a, q_b, q_c and q_d. There are 8 different sequences in these policies, the matrix U_j has then 4 rows and 8 columns.

If $b_i(s,.)$ is a multi-agent belief state, i.e. a probability distribution over the policies of Q_j^t for some state $s \in S$, then the product $b_i(s,.)U_j^t$ returns a vector containing the probability of every sequence of $\tilde{Q}_j^t$. To reduce the dimensionality of $b_i(s,.)$ from $|Q_j^t|$ to N, we should find a *transformation function* f defined by:

$$\begin{aligned} f : \Delta Q_j^t &\rightarrow [0,1]^N \\ b_i(s,.) &\mapsto \tilde{b}_i(s,.) \end{aligned}$$

$\tilde{b}_i(s,.)$ is called *the reduced multi-agent belief state*, it corresponds to the belief about the sequences, whereas $b_i(s,.)$ is the belief about the policies. In order that f be an accurate transformation function, we should be able to make predictions about any sequence $\tilde{q}_j$ by using only the vector $\tilde{b}_i(s,.)$.

Linear reduction of policy space dimensionality

The outcome matrix U_j^t can be factorized as follows:

$$U_j^t = F_j^t \tilde{U}_j^t$$

In this case, our transformation function is simply the matrix F_j^t. The reduced belief state $\tilde{b}_i$ can be generated from b_i by:

$$\tilde{b}_i(s,.) = b_i(s,.)F_j^t$$

and the probabilities of $\tilde{Q}_j^t$ sequences can be found by:

$$\begin{aligned} b_i(s,.)U_j^t &= b_i(s,.)(F_j^t \tilde{U}_j^t) \\ &= \tilde{b}_i(s,.)\tilde{U}_j^t \end{aligned}$$

The matrix F_j^t is a basis for the matrix U_j^t, it is formed by linearly independent columns of U_j^t. We use $Basis(\tilde{Q}_j^t)$ to indicate the set of sequences corresponding to these linearly independent columns. In Figure 1, $Basis(\tilde{Q}_j^t)$ contains the sequences $\tilde{q}_1^*$, $\tilde{q}_2^*$ and $\tilde{q}_3^*$. Note that $|Basis(\tilde{Q}_j^t)| \leq |Q_j^t|$, because the linear rank of a matrix cannot be more than the number of rows in this matrix, and F_j^t has exactly $|Q_j^t|$ rows.

Since F_j^t is a basis for the outcome matrix U_j^t, we can write any column of U_j^t as linear combination of F_j^t columns:

$$Pr(\tilde{q}_j^t | q_j^t) = U_j^t(q_j^t, \tilde{q}_j^t) = F_j^t(q_j^t, .) w_{\tilde{q}_j^t}$$

where $w_{\tilde{q}^t_j}$ is a weight vector[1] associated to the sequence $\tilde{q}^t_j$ (we have one weight vector per sequence). Similarly, $w_{\tilde{q}^t_j}$ is the column corresponding to $\tilde{q}^t_j$ in the matrix $\tilde{U}^t_j$.

If $\tilde{b}_i(s,.) = b_i(s,.)F_j$ is a reduced belief state, then:

$$\begin{aligned} Pr(s,\tilde{q}^t_j|b_i) &= \sum_{q^t_j \in Q^t_j} b_i(s,q^t_j) Pr(\tilde{q}^t_j|q^t_j) \\ &= \sum_{q^t_j \in Q^t_j} b_i(s,q^t_j)(F(q^t_j,.)w_{\tilde{q}^t_j}) \\ &= \tilde{b}_i(s,.)w_{\tilde{q}^t_j} \end{aligned}$$

This means that given a reduced belief state, the probability of any sequence is a linear combination of the probabilities contained in this reduced belief state.

Finding the basis sequences

The main problem now is how to find the basis sequences of $\tilde{Q}^t_j$ and their matrix F^t_j without factorizing the entire matrix U^t_j at each step. The theorem below states that if $Basis(\tilde{Q}^{t-1}_j)$ is the set of basis sequences for horizon $t-1$, then the sequences of $Basis(\tilde{Q}^t_j)$ for horizon t will be among the *one step* extensions of the $Basis(\tilde{Q}^{t-1}_j)$ sequences. This means that we need to extend only the sequences of $Basis(\tilde{Q}^{t-1}_j)$ at each step, and construct the matrix F^t_j where the columns correspond to these extended sequences. F^t_j may contain some linearly dependent columns, along with the basis columns. Thus, we use a Gauss-Jordan elimination on F^t_j, or any other decomposition technique, to extract $Basis(\tilde{Q}^t_j)$ in a polynomial time.

Theorem 1. *Let Q^{t-1}_j be a set of horizon $t-1$ policies, $\tilde{Q}^{t-1}_j$ is the set of horizon $t-1$ sequences corresponding to Q^{t-1}_j, Q^t_j is the set of horizon t policies created from Q^{t-1}_j by an exhaustive backup, and $\tilde{Q}^t_j$ is the set of horizon t sequences corresponding to Q^t_j, then:*

$$Basis(\tilde{Q}^t_j) \subseteq \{ao\tilde{q}^{t-1}_j : a \in A_j, o \in \Omega_j, \tilde{q}^{t-1}_j \in Basis(\tilde{Q}^{t-1}_j)\}$$

Proof. This theorem claims that the sequences of $Basis(\tilde{Q}^t_j)$ are contained in the one step extensions of the $Basis(\tilde{Q}^{t-1}_j)$ sequences. In other words, $\forall \tilde{q}^{t-1}_j \in \tilde{Q}^{t-1}_j, \forall a \in A_j, \forall o \in \Omega_j$, we should be able to write the column $U^t_j(.,ao\tilde{q}^t_j)$ as a linear combination of the columns of F^t_j, which is the sub-matrix of U^t_j corresponding to the sequences $ao\tilde{q}^{t-1}_j : a \in A_j, o \in \Omega_j, \tilde{q}^{t-1}_j \in Basis(\tilde{Q}^{t-1}_j)$. U^t_j is the outcome matrix of Q^t_j, U^{t-1}_j is the outcome matrix of Q^{t-1}_j, F^{t-1}_j is the sub-matrix of U^{t-1}_j corresponding to the sequences of $Basis(\tilde{Q}^{t-1}_j)$.

Let q^t_j be a policy of Q^t_j, $\tilde{q}^{t-1}_j$ a sequence of $\tilde{Q}^{t-1}_j$, a an action of A_j and o an observations of Ω_j, then we can be in one of the following two situations:

- Case 1: the policy tree q^t_j starts with an action different from a, then the sequence $ao\tilde{q}^{t-1}_j$ does not appear in the policy q^t_j, so: $U^t_j(q^t_j, ao\tilde{q}^{t-1}_j) = 0$.

- Case 2: the policy tree q^t_j starts with the action a, then the sequence $ao\tilde{q}^{t-1}_j$ appears in the policy q^t_j iff $\tilde{q}^{t-1}_j$ appears in $o(q^t_j)$, the sub-tree of q^t_j below the first action a and the observation o. We have then $U^t_j(q^t_j, ao\tilde{q}^{t-1}_j) = U^{t-1}_j(o(q^t_j), \tilde{q}^{t-1}_j)$ and $\forall \tilde{q}^*_j \in Basis(\tilde{Q}^{t-1}_j) : F^t_j(q^t_j, ao\tilde{q}^*_j) = F^{t-1}_j(o(q^t_j), \tilde{q}^*_j)$. So [2]:

$$\begin{aligned} U^t_j(q^t_j, ao\tilde{q}^{t-1}_j) &= U^{t-1}_j(o(q^t_j), \tilde{q}^{t-1}_j) \\ &= \sum_{\tilde{q}^*_j \in Basis(\tilde{Q}^{t-1}_j)} F^{t-1}_j(o(q^t_j), \tilde{q}^*_j) w_{\tilde{q}^{t-1}_j}(\tilde{q}^*_j) \\ &= \sum_{\tilde{q}^*_j \in Basis(\tilde{Q}^{t-1}_j)} F^t_j(q^t_j, ao\tilde{q}^*_j) w_{ao\tilde{q}^{t-1}_j}(ao\tilde{q}^*_j) \\ &= F^t_j(q^t_j,.) w_{ao\tilde{q}^{t-1}_j} \end{aligned}$$

where we define $w_{ao\tilde{q}^{t-1}_j}$ as follows:

$$\begin{cases} w_{ao\tilde{q}^{t-1}_j}(a'o'\tilde{q}^*_j) = w_{\tilde{q}^{t-1}_j}(\tilde{q}^*_j) \text{ if } a = a' \text{ and } o = o', \\ w_{ao\tilde{q}^{t-1}_j}(a'o'\tilde{q}^*_j) = 0 \text{ else.} \end{cases}$$

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

$$U^t_j(q^t_j, ao\tilde{q}^{t-1}_j) = F^t_j(q^t_j,.) w_{ao\tilde{q}^{t-1}_j}$$

Notice that the vector $w_{ao\tilde{q}^{t-1}_j}$ is defined independently on the policy q^t_j. So, the sequences $\{ao\tilde{q}^*_j : a \in A_j, o \in \Omega_j, \tilde{q}^*_j \in Basis(\tilde{Q}^{t-1}_j)\}$ are indeed a basis for the matrix U^t_j. □

Reduced value vectors

Since we want to use this representation for planning, we have to redefine the value function (equations 1 and 2), given that the belief state is now about sequences instead of policies. First, we define the expected value $V_{\tilde{q}^t}(s)$ of a joint sequence $\tilde{q}^t = \vec{a}_1\vec{o}_1\vec{a}_2\vec{o}_2\ldots\vec{a}_t$ in state s as follows:

$$\begin{aligned} V_{\tilde{q}^t}(s) &= Pr(\tilde{q}^t|s) R_{\tilde{q}^t}(s) \\ &= Pr(\vec{o}_1\vec{o}_2\ldots\vec{o}_{t-1}|s_1 = s, \vec{a}_1\vec{a}_2\ldots\vec{a}_{t-1}) R_{\tilde{q}^t}(s) \end{aligned}$$

[1] To simplify the notations, we consider that the belief vectors are rows and the weight vectors are columns.

[2] We use $\tilde{q}^*_j$ to indicate a basis sequence, the horizon of this sequence can be inferred from the context.

$Pr(\vec{o}_1\vec{o}_2\ldots\vec{o}_{t-1}|s,\vec{a}_1\vec{a}_2\ldots\vec{a}_{t-1})$ is the probability that the observations of $\tilde{q}^t$ will occur if we start executing the actions of $\tilde{q}^t$ at s, and $R_{\tilde{q}^t}(s)$ is the reward expected from the actions of $\tilde{q}^t$ such that the observations of $\tilde{q}^t$ will occur. $R_{\tilde{q}^t}(s)$ is given by:

$$R_{\tilde{q}^t}(s) = \sum_{k=1}^{t}\gamma^{k-1}\sum_{s'\in S}Pr(s_k=s'|s_1=s,\vec{a}_1\vec{o}_1\ldots\vec{a}_t)R(s',\vec{a}_k)$$

$$= \frac{\sum_{k=1}^{t}\gamma^{k-1}\sum_{s'\in S}Pr(s_k=s',\vec{o}_1\ldots\vec{o}_{t-1}|s_1=s,\vec{a}_1\ldots\vec{a}_{t-1})R(s',\vec{a}_k)}{Pr(\vec{o}_1\vec{o}_2\ldots\vec{o}_{t-1}|s_1=s,\vec{a}_1\vec{a}_2\ldots\vec{a}_{t-1})}$$

$$= \frac{\sum_{k=1}^{t}\gamma^{k-1}\sum_{s'\in S}\alpha_k(s,s',\tilde{q}^t)\beta^k(s',\tilde{q}^t)R(s',\vec{a}_k)}{Pr(\vec{o}_1\vec{o}_2\ldots\vec{o}_{t-1}|s_1=s,\vec{a}_1\vec{a}_2\ldots\vec{a}_{t-1})}$$

where:

$$\begin{cases}\alpha_k(s,s',\tilde{q}^t) = Pr(\vec{o}_1\ldots\vec{o}_{k-1},s_k=s'|s_1=s,\vec{a}_1\ldots\vec{a}_{k-1})\\ \beta_k(s',\tilde{q}^t) = Pr(\vec{o}_k\ldots\vec{o}_{t-1}|s_k=s',\vec{a}_k\vec{a}_2\ldots\vec{a}_{t-1})\end{cases}$$

We applied Bayes' rule, and then we decomposed $Pr(s_k=s',\vec{o}_1\ldots\vec{o}_{t-1}|s_1=s,\vec{a}_1\ldots\vec{a}_{t-1})$ into $\alpha_k(s,s',\tilde{q}^t)$ and $\beta_k(s',\tilde{q}^t)$, taking advantage of Markov property.

So, the expected value of a joint sequence $\tilde{q}^t$ is given by:

$$V_{\tilde{q}^t}(s) = \sum_{k=1}^{t}\gamma^{k-1}\sum_{s'\in S}\alpha_k(s,s',\tilde{q}^t)\beta_k(s',\tilde{q}^t)R(s',\vec{a}_k)$$

This value is calculated recursively as follows:

$$V_{\vec{a}\vec{o}\tilde{q}^{t-1}}(s) = \sum_{k=1}^{t}\gamma^{k-1}\sum_{s'\in S}\alpha_k(s,s',\vec{a}\vec{o}\tilde{q}^{t-1})\beta_k(s',\vec{a}\vec{o}\tilde{q}^{t-1})R(s',\vec{a})$$

$$= \beta_1(s,\vec{a}\vec{o}\tilde{q}^{t-1})R(s,\vec{a}) + \sum_{k=2}^{t}\gamma^{k-1}\sum_{s'\in S}\alpha_k(s,s',\vec{a}\vec{o}\tilde{q}^{t-1})\beta_k(s',\vec{a}\vec{o}\tilde{q}^{t-1})R(s',\vec{a})$$

$$= \beta_1(s,\vec{a}\vec{o}\tilde{q}^{t-1})R(s,\vec{a}) + \gamma\sum_{k=1}^{t-1}\gamma^{k-1}\sum_{s'\in S}\sum_{s''\in S}[P(s''|s,\vec{a})P(\vec{o}|s'',\vec{a})\alpha_k(s'',s',\tilde{q}^{t-1})\beta_k(s',\tilde{q}^{t-1})R(s',\vec{a})]$$

$$= \beta_1(s,\vec{a}\vec{o}\tilde{q}^{t-1})R(s,\vec{a}) + \gamma\sum_{s'\in S}P(s'|s,\vec{a})P(\vec{o}|s',\vec{a})V_{\tilde{q}^{t-1}}(s')$$

We used the following properties: $\alpha_1(s,s,\vec{a}\vec{o}\tilde{q}^{t-1}) = 1$, $\alpha_1(s,s',\vec{a}\vec{o}\tilde{q}^{t-1}) = 0$ for $s'\neq s$, $\beta_k(s',\vec{a}\vec{o}\tilde{q}^{t-1}) = \beta_{k-1}(s',\tilde{q}^{t-1})$ for $k\geq 2$, and $\alpha_k(s,s',\vec{a}\vec{o}\tilde{q}^{t-1}) = \sum_{s''\in S}P(s''|s,\vec{a})P(\vec{o}|s'',\vec{a})\alpha_{k-1}(s'',s',\tilde{q}^{t-1})$.

The value function of an individual policy q_i^t in a reduced belief state $\tilde{b}_i$ is given by:

$$V_{q_i^t}(\tilde{b}_i) = \sum_{s\in S}Pr(s|\tilde{b}_i)\sum_{\substack{\tilde{q}_j^t\in\tilde{Q}_j^t\\ \tilde{q}_i^t\in\tilde{Q}_i^t}}Pr(\tilde{q}_j^t|\tilde{b}_i)Pr(\tilde{q}_i^t|q_i^t)V_{\langle\tilde{q}_i^t,\tilde{q}_j^t\rangle}(s)$$

$$= \sum_{s\in S}\sum_{\substack{\tilde{q}_j^*\in Basis(\tilde{Q}_j^t)\\ \tilde{q}_i^*\in Basis(\tilde{Q}_i^t)\\ F_i^t(q_i^t,\tilde{q}_i^*)=1}}\tilde{b}_i(s,\tilde{q}_j^*)\tilde{V}_{\langle\tilde{q}_i^*,\tilde{q}_j^*\rangle}(s)$$

where:

$$\tilde{V}_{\langle\tilde{q}_i^*,\tilde{q}_j^*\rangle}(s) \stackrel{def}{=} \sum_{\substack{\tilde{q}_j^t\in\tilde{Q}_j^t\\ \tilde{q}_i^t\in\tilde{Q}_i^t}}w_{\tilde{q}_j^t}(\tilde{q}_j^*)w_{\tilde{q}_i^t}(\tilde{q}_i^*)V_{\langle\tilde{q}_i^t,\tilde{q}_j^t\rangle}(s)$$

We substituted the probability of each sequence $\tilde{q}_j^t$ with a linear combination of the probabilities $\tilde{b}_i(s,\tilde{q}_j^*)$ of the basis sequences $\tilde{q}_j^*$, and the probability of each sequence $\tilde{q}_i^t$ with a linear combination of the probabilities $F_i^t(q_i^t,\tilde{q}_i^*)\in\{0,1\}$ of the basis sequences $\tilde{q}_i^*$. The reduced vector $\tilde{V}_{\langle\tilde{q}_i^*,\tilde{q}_j^*\rangle}$ defines the contribution of $\langle\tilde{q}_i^*,\tilde{q}_j^*\rangle$ to the value of a joint policy, by including a proportion of the values of all the sequences which depend on $\langle\tilde{q}_i^*,\tilde{q}_j^*\rangle$. If we want to calculate the value of a policy q_i^t for a given multi-agent belief state $\tilde{b}_i$, all we need are the vectors $\tilde{V}_{\langle\tilde{q}_i^*,\tilde{q}_j^*\rangle}$ and the matrix F_i^t.

We will see now how to calculate the value vectors $\tilde{V}^t$ given the value vectors $\tilde{V}^{t-1}$. At horizon 1, each policy is a single action. We have then $Basis(Q_j^1) = A_j$, and $\tilde{V}_{\langle a_i,a_j\rangle}(s) = V_{\langle a_i,a_j\rangle}(s) = R(s,\langle a_i,a_j\rangle)$. At horizon $t>1$, we know from Theorem 1 that the basis sequences are of the form $a_io_i\tilde{q}_i^*$ and $a_jo_j\tilde{q}_j^*$, where $\tilde{q}_i^*\in Basis(\tilde{Q}_i^{t-1})$ and $\tilde{q}_j^*\in Basis(\tilde{Q}_j^{t-1})$.

$$\begin{aligned}\tilde{V}_{\langle a_io_i\tilde{q}_i^*,a_jo_j\tilde{q}_j^*\rangle}(s) &= \sum_{\substack{\tilde{q}_j^{t-1}\in\tilde{Q}_j^{t-1}\\ \tilde{q}_i^{t-1}\in\tilde{Q}_i^{t-1}}}w_{a_jo_j\tilde{q}_j^{t-1}}(\tilde{q}_j^*)w_{a_io_i\tilde{q}_i^{t-1}}(\tilde{q}_i^*)\\ &\qquad V_{\langle a_io_i\tilde{q}_i^{t-1},a_jo_j\tilde{q}_j^{t-1}\rangle}(s)\\ &= R(s,\langle a_i,a_j\rangle)C_{\langle a_io_i\tilde{q}_i^*,a_jo_j\tilde{q}_j^*\rangle}(s)\\ &+ \gamma\sum_{s'\in S}Pr(\langle o_i,o_j\rangle,s'|s,\langle a_i,a_j\rangle)V_{\langle\tilde{q}_i^*,\tilde{q}_j^*\rangle}(s')\end{aligned} \tag{4}$$

where:

$$\begin{aligned}C_{\langle a_io_i\tilde{q}_i^*,a_jo_j\tilde{q}_j^*\rangle}(s) &\stackrel{def}{=} \sum_{\substack{\tilde{q}_j^{t-1}\in\tilde{Q}_j^{t-1}\\ \tilde{q}_i^{t-1}\in\tilde{Q}_i^{t-1}}}w_{a_jo_j\tilde{q}_j^{t-1}}(\tilde{q}_j^*)w_{a_io_i\tilde{q}_i^{t-1}}(\tilde{q}_i^*)\\ &\qquad Pr(\langle a_io_i\tilde{q}_i^{t-1},a_jo_j\tilde{q}_j^{t-1}\rangle)|s)\\ &= \sum_{s'\in S}P(s'|s,\langle a_i,a_j\rangle)O(\langle o_i,o_j\rangle|s',\langle a_i,a_j\rangle)C_{\langle\tilde{q}_i^*,\tilde{q}_j^*\rangle}(s')\end{aligned} \tag{5}$$

In order to calculate the value vector $\tilde{V}_{\langle a_io_i\tilde{q}_i^*,a_jo_j\tilde{q}_j^*\rangle}$, we only need to know the vectors $C_{\langle\tilde{q}_i^*,\tilde{q}_j^*\rangle}$ and $\tilde{V}_{\langle\tilde{q}_i^*,\tilde{q}_j^*\rangle}$, provided by the last iteration of the dynamic programming algorithm.

Algorithm

Algorithm 2 describes the main steps of the dynamic programming algorithm where the policies are evaluated in a reduced dimensional space. We keep the same structures Q_i^t and Q_j^t used in Algorithm 1. The value vectors V^t are replaced by lower dimensional vectors $\tilde{V}^t$. We also need to

Input: Q_i^{t-1},Q_j^{t-1},$Basis(\tilde{Q}_i^{t-1})$,$Basis(\tilde{Q}_j^{t-1})$, $\tilde{V}_i^{t-1}$,$\tilde{V}_j^{t-1}$,C_i^{t-1},C_j^{t-1};
Q_i^t, $Q_j^t \leftarrow$ fullBackup(Q_i^{t-1}), fullBackup(Q_j^{t-1});
$Basis(\tilde{Q}_i^t) \leftarrow A_i \times O_i \times Basis(\tilde{Q}_i^{t-1})$;
$Basis(\tilde{Q}_j^t) \leftarrow A_j \times O_j \times Basis(\tilde{Q}_j^{t-1})$;
Calculate the vectors C^t by using C^{t-1} (Equation 5);
Calculate the vectors $\tilde{V}^t$ by using $\tilde{V}^{t-1}$ (Equation 4);
repeat
  remove the policies of Q_j^t that are dominated (Table 2);
  remove the policies of Q_j^t that are dominated (Table 2);
until *no more policies in* Q_i^t *or* Q_j^t *can be removed* ;
$removeDependence(Q_i^t, Basis(\tilde{Q}_i^t), Basis(\tilde{Q}_j^t), C^t, \tilde{V}^t)$;
$removeDependence(Q_j^t, Basis(\tilde{Q}_j^t), Basis(\tilde{Q}_j^t), C^t, \tilde{V}^t)$;
Output: Q_i^t, Q_j^t, $Basis(\tilde{Q}_i^t)$, $Basis(\tilde{Q}_j^t)$, $\tilde{V}_i^t$, $\tilde{V}_j^t$, C^t;

Algorithm 2: Dynamic Programming for DEC-POMDPs with Lossless Policy Belief Compression.

Input: $Q_i^t, Basis(\tilde{Q}_i^t), Basis(\tilde{Q}_j^t), C^t, \tilde{V}^t$;
Use a decomposition method to find the linearly dependent sequences in $Basis(\tilde{Q}_i^t)$, and remove them from $Basis(\tilde{Q}_i^t)$;
foreach *each removed sequence* $\tilde{q}_i$ *from* $Basis(\tilde{Q}_i^t)$ **do**
  foreach *basis sequence* $\tilde{q}_i^*$ *from* $Basis(\tilde{Q}_i^t)$ **do**
    foreach *basis sequence* $\tilde{q}_j^*$ *from* $Basis(\tilde{Q}_j^t)$ **do**
      $C_{\langle \tilde{q}_i^*, \tilde{q}_j^* \rangle} \leftarrow C_{\langle \tilde{q}_i^*, \tilde{q}_j^* \rangle} + w_{\tilde{q}_i}(\tilde{q}_i^*) C_{\langle \tilde{q}_i, \tilde{q}_j^* \rangle}$;
      $\tilde{V}_{\langle \tilde{q}_i^*, \tilde{q}_j^* \rangle} \leftarrow \tilde{V}_{\langle \tilde{q}_i^*, \tilde{q}_j^* \rangle} + w_{\tilde{q}_i}(\tilde{q}_i^*) \tilde{V}_{\langle \tilde{q}_i, \tilde{q}_j^* \rangle}$;
    end
  end
end
Output: Updated $Basis(\tilde{Q}_i^t), C^t, \tilde{V}^t$;

Algorithm 3: Removing the dependent sequences from $Basis(\tilde{Q}_i^t)$ and updating the vectors C^t, $\tilde{V}^t$.

maintain the lists $Basis(\tilde{Q}_i^t)$, $Basis(\tilde{Q}_j^t)$, and the probability vectors C^t. The matrix F_i^t (resp. F_j^t) is implicitly represented by specifying for each basis sequence the list of policies where this sequence occurs.

At step $t = 1$, we have $Q_i^1 = Basis(\tilde{Q}_i^1) = A_i, Q_j^1 = Basis(\tilde{Q}_j^1) = A_j$ and $\forall a_i \in A_i, \forall a_j \in A_j : \tilde{V}_{\langle a_i, a_j \rangle} = R(., \langle a_i, a_j \rangle), C_{\langle a_i, a_j \rangle} = 1$. At step $t > 1$, Q_i^t and Q_j^t are the sets of all possible policies of horizon t where the sub-policies of horizon $t-1$ are in Q_i^{t-1} and Q_j^{t-1} respectively. $Basis(\tilde{Q}_i^t)$ and $Basis(\tilde{Q}_j^t)$ are formed by one step extensions of $Basis(\tilde{Q}_i^{t-1})$ and $Basis(\tilde{Q}_i^{t-1})$ respectively.

We now calculate the probability vectors C^t by using the vectors C^{t-1} in Equation (5), and the value vectors $\tilde{V}^t$ using the vectors C^t and $\tilde{V}^{t-1}$ in Equation (4). The vectors $\tilde{V}^t$ are used to determine which policies of i and j are dominated and should be removed. We use the linear program of Table 2 to solve this problem for each policy q_i^t. The variables of this linear program are: ε, and the probabilities $\tilde{b}(.,.)$ of the reduced multi-agent belief state, we have then $|S||Basis(\tilde{Q}_j^t)|+1$ variables. Notice that contrary to the original belief state $b_i(s,.)$ which is a probability distribution over the policies of agent j, the reduced belief space $\tilde{b}_i(s,.)$ is not necessarily a probability distribution, because the sequences are not mutually exclusive. In Figure 1 for example, if $b(s,q_a) = 1$ then $\tilde{b}(s,\tilde{q}_1^*) = 1$ and $b(s,\tilde{q}_3^*) = 1$. The relation between the different basis sequences is more complex and more constraints should be added to make sure that any reduced belief considered in the linear program will really correspond to some belief in the original space. In fact, if we take any belief $b_i(s,.)$, we can always find a reduced belief $\tilde{b}_i(s,.) = b_i(s,.)F_j^t$ where all the policies keep the same values, but given a reduced belief $\tilde{b}_i(s,.)$, we are not sure of finding a belief $b_i(s,.)$ in the original space such that $\tilde{b}_i(s,.) = b_i(s,.)F_j^t$. This is true if and only if the transformation function, represented by the matrix F_j is a bijection. However, if a policy q_i^t is not dominated, then there is a belief $b_i(.,.)$ where $V_{q_i^t}(b_i) > V_{q_i^{t\prime}}(b_i), \forall q_i^{t\prime} \in Q_i^t - \{q_i^t\}$, and in the corresponding reduced belief $\tilde{b}_i(.,.) = b_i(.,.)F_j$, we will also have $V_{q_i^t}(\tilde{b}_i) > V_{q_i^{t\prime}}(\tilde{b}_i), \forall q_i^{t\prime} \in Q_i^t - \{q_i^t\}$. Therefore, the linear program of Table 2. keeps at least all the dominant policies, but can keep some dominated policies (Table 3.).

minimize: ε
subject to:
$\forall s \in S, \forall \tilde{q}_j^* \in Basis(\tilde{Q}_j^t)$:
$0 \leq \tilde{b}_i(s,\tilde{q}_j^*) \leq 1$
$\forall q_i^{t\prime} \in Q_i^t - \{q_i^t\}$:
$V_{q_i^t}(\tilde{b}_i) - V_{q_i^{t\prime}}(\tilde{b}_i) + \varepsilon > 0$

Table 2: The linear program used to determine if a policy q_i^t is dominated or not, with a reduced belief space.

After pruning the dominated policies, some basis sequences become linearly dependent. Algorithm 3 is then used to remove the newly dependent sequences, and to update the parameters C and $\tilde{V}$ of the remaining basis sequences. These two update operations are derived from the definitions of C and $\tilde{V}$. Notice that when we eliminate policies (i.e. eliminate rows from the matrix U), the linearly dependent sequences remain dependent, and keep the same weight vectors.

Experiments

We implemented both of Algorithm 1 (DP) and Algorithm 2 (DP with Policy Compression) using ILOG Cplex 10 solver on an AMD Athlon machine with a 1.80 GH processor and 1.5 GB ram. We used Gauss-Jordan elimination method to

		Dynamic Programming		Dynamic Programming with Lossless Policy Compression			
Problem	T	runtime	policies	runtime	policies	basis sequences	compression 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 3: The runtime (in seconds) and the number of policies and sequences of DP algorithms, with and without compression.

find the basis sequences. 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 sequence. We compared the performances of these two algorithms on two benchmark problems MA-Tiger and MABC (Hansen, Bernstein, & Zilberstein 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 basis sequences. Notice also that the compression ratio (policies number/basis sequences 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 sequences, but the computational efficiency will be possibly affected. We can see that as in all the compression techniques, there is a tradeoff between the space performance and the time performance.

Conclusion

The dimensionality of the policy space is a crucial factor in the complexity of Dynamic Programming algorithms for DEC-POMDPs. In this paper, we introduced a new approach for dealing with this problem, based on projecting the policy beliefs from the high dimensional space of trees to the low dimensional space of sequences, and using matrix factorization methods to reduce even more the number of sequences. Consequently, the memory space used in this algorithm is significantly smaller, while the runtime is lower compared to the original DP algorithm. This method can be used in approximate DP algorithms, mainly for domains with large observations space. We target also to investigate quick and lossy factorization techniques, and more specifically, binary-matrices factorization algorithms.

References

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)*, 18–25.

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.

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.

Littman, M.; Sutton, R.; and Singh, S. 2001. 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.; 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)*.

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.

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)*.

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.