Parallel Rollout for Online Solution of DEC-POMDPs

Camille Besse & Brahim Chaib-draa
DAMAS Laboratory
Department of Computer Science and Software Engineering
Laval University, G1K 7P4, Quebec (Qc), Canada
{besse,chaib}@damas.ift.ilaval.ca

Abstract

A major research challenge is presented by scalability of algorithms for solving decentralized POMDPs because of their double exponential worst-case complexity for finite horizon problems. First algorithms have only been able to solve very small instances on very small horizons. One exception is the Memory-Bounded Dynamic Programming algorithm – an approximation technique that has proved efficient in handling same sized problems but on large horizons. In this paper, we propose an online algorithm that also approximates larger instances of finite horizon DEC-POMDPs based on the Rollout algorithm. To evaluate the effectiveness of this approach, we compare the presented approach to a recently proposed algorithm called memory bounded dynamic programming. Experimental results show that despite the very high complexity of DEC-POMDPs, the combination of Rollout techniques and estimation techniques performs well and leads to a significant improvement of existing approximation techniques.

Introduction

Markov Decision Processes (MDPs) and their partially observable pendant (POMDPs) have proved useful for a single agent that plans and acts under uncertainty. Decentralized POMDPs (DEC-POMDPs) propose a natural extension of these frameworks for cooperative, multiple and physically distributed agents (Bernstein, Zilberstein, & Immerman 2000). They actually capture situations in which agents may have different partial knowledge about the state of the environment and the other agents. Many distributed decision problems in real life, such as multi-robot coordination, unmanned vehicles cooperation, information gathering and load balancing can be modeled as DEC-POMDPs. Unfortunately, solving optimally finite-horizon DEC-POMDPs is NEXP- complete (Bernstein, Zilberstein, & Immerman 2000) and even ε-approximation are NEXP-hard (Rabinovich, Goldman, & Rosenschein 2003). That's why exact algorithms have mostly theoretical significance. Seuken & Zilberstein (Seuken & Zilberstein 2005) proposed a survey of existing formal models, complexity results and planning algorithms for the interested reader.

Several methods exist for solving DEC-POMDPs approximately or exactly (Amato, Bernstein, & Zilberstein 2007; Hansen, Bernstein, & Zilberstein 2004; Nair *et al.* 2003; Roth, Simmons, & Veloso 2005; Seuken & Zilberstein 2007b; Szer & Charpillet 2005). Unfortunately, these algorithms cannot deal with problems with more than 10 states. One exception is the Memory-Bounded Dynamic Programming (MBDP) algorithm (Seuken & Zilberstein 2007b). As opposed to the double exponential growth of the optimal algorithm, MBDP's runtime grows polynomially with the horizon using some approximation on beliefs over other agents' policies. As a result, this algorithm can solve small sized problems with horizons that are multiple orders of magnitude larger than what was previously possible (e.g. horizon 1,000 for the multi-agent tiger problem). However, MBDP's efficiency still diminishes exponentially with the size of the observation space, and even with very small problems (e.g. around 5 observations) this can be prohibitive. Thus, Seuken & Zilberstein (Seuken & Zilberstein 2007a) proposed an improvement of their algorithm by bounding also the number of observations explored during the belief points selection to the most likely to occur. Then, their algorithm becomes polynomial in time and in space for a fixed number of agents.

In contrast to the work of Seuken & Zilberstein we propose an online alternative way of approximating DEC-POMDPs based on the rollout algorithm of Bertsekas (Bertsekas, Tsitsiklis, & Wu 1997) and memory bounded dynamic programming so that horizon length does not influence space or time complexity more than polynomially in the horizon and the number of observations. The main insights of this approach is that the Rollout algorithm is known to improve substantially a given lower bound. Thus, once a quick and good lower bound is computed, executing a one-step-lookahead-greedy policy according to these heuristics is straightforward. Results presented here show that, for considered problems, this technique brings a significant improvement as soon as the lower bound lets room for it.

This paper is organized as follows: in a first part, a background on the DEC-POMDP model is done and some recalls about the Rollout algorithm are given. Second, an approximation technique called particle filters is introduced in order to improve Monte Carlo trials' quality. Then, the parallel Rollout approach for online DEC-POMDPs is described before presenting some experimental results.

Background on DEC-POMDP model

In this section we introduce the DEC-POMDP model and present some techniques used in partially observable environments that are related to our approach.

The formalism used here is based on decentralized partially observable Markov Decision Processes (DEC-POMDPs) (Bernstein, Zilberstein, & Immerman 2000). Results presented in this paper, however, can also be applied to equivalent models such as the MTDP (Pynadath & Tambe 2002) framework or POIPSG (Peshkin *et al.* 2000).

Definition 1 *An infinite horizon* DEC-POMDP *is defined by a tuple* $\langle \alpha, \mathcal{S}, \{\mathcal{A}_i\}_{i\in\alpha}, \mathcal{T}, \{\Omega_i\}_{i\in\alpha}, \mathcal{O}, \mathcal{R}\rangle$*, where:*

- α *is a finite set of indexed* agents $i \in \alpha, 1 \leq i \leq n$*;*
- $\mathcal{S}$ *is a finite set of* states $s \in \mathcal{S}$ *with an initial state* s_0*;*
- $\mathcal{A}_i$ *is the finite set of* actions *of agent* i $a_i \in \mathcal{A}_i$ *and* $\boldsymbol{\mathcal{A}} = \times_{i\in\alpha}\mathcal{A}_i$ *is the set of joint actions where* $\boldsymbol{a} = \langle a_1, \ldots, a_n\rangle \in \boldsymbol{\mathcal{A}}$ *denotes a joint action;*
- $\mathcal{T}(s'|s, \boldsymbol{a}) : \mathcal{S} \times \boldsymbol{\mathcal{A}} \times \mathcal{S} \mapsto [0, 1]$ *is the* transition probability *of the system that taking joint action* $\boldsymbol{a}$ *in state* s *results in state* s'*;*
- Ω_i *is the finite set of* observations *of agent* i *and* $\boldsymbol{\Omega} = \times_{i\in\alpha}\Omega_i$ *is the set of joint observations where* $\boldsymbol{o} = \langle o_1, \ldots, o_n\rangle \in \boldsymbol{\Omega}$ *denotes a joint observation;*
- $\mathcal{O}(\boldsymbol{o}|s, \boldsymbol{a}, s') : \mathcal{S} \times \boldsymbol{\mathcal{A}} \times \boldsymbol{\Omega} \times \mathcal{S} \mapsto [0, 1]$ *is the* observation function *representing the probability to get the joint observation* $\boldsymbol{o}$ *given that joint action* $\boldsymbol{a}$ *was taken in state* s *and results in state* s'*;*
- $\mathcal{R}(s, \boldsymbol{a}) : \mathcal{S} \times \boldsymbol{\mathcal{A}} \mapsto \mathbb{R}$ *is the* reward *produced by the system when the joint action* $\boldsymbol{a}$ *is executed in state* s*.*

The process that considers each joint action of the DEC-POMDP *as an atomic action is called the underlying* POMDP.

In fact, the underlying system state s of a DEC-POMDP is not available to the agents during execution, so they must base their belief about the current situation. In a single-agent setting, the *belief state* of belief point $\boldsymbol{b}$, which is a distribution over states, is sufficient for optimal planning. In a distributed setting, each agent must maintain a *multiagent belief state* that is a not only a distribution over states but also over other agents' policies. However, as noted by Seuken & Zilberstein (Seuken & Zilberstein 2007a), only a few of these policies are enough to have a good approximation of other agent's beliefs. Let us now see how rollout principle works.

Rollout Algorithm

Bertsekas *et al.* (Bertsekas, Tsitsiklis, & Wu 1997) used sampling to design a method of policy improvement called *Rollout*. This method uses Monte-Carlo sampling to improve a given policy π in an online fashion. At each time step, the set of next states are computed according to available action in the current state. Then, the given policy is simulated using Monte Carlo sampling in order to estimate the expected value of each next state, and the results of the simulation are used to select the (apparently) best current action which might differ from the one prescribed by π. The action selected is then the action with the highest Q-value at the current state, as estimated by sampling. It is possible to show that the resulting online policy outperforms the initial given policy unless this last is optimal in which case Rollout performs optimally as well. More recently, Chang *et al.* (Chang, Givan, & Chong 2004) apply this method to POMDPs using belief sampling instead of state sampling.

We propose here a similar sample-based approach using a portfolio of heuristics. Then, using particle filters in Monte-Carlo trials to estimate the value of each policy for each action, we ensure improving the heuristics using a fixed amount of memory and time. Formally, given a set of heuristic multiagent policies Π over a multiagent belief state, the parallel Rollout policy selects action according to:

$$\boldsymbol{a} = \arg\max_{\boldsymbol{a}\in\boldsymbol{\mathcal{A}}} \max_{\pi\in\Pi} \mathrm{E}\left[\sum_{t=0}^{T} \mathcal{R}(\boldsymbol{b}^t, \pi(\boldsymbol{b}^t))\right]$$

where $\mathcal{R}(\boldsymbol{b}^t, \boldsymbol{a})$ is the expected reward of the belief $\boldsymbol{b}^t$.

The main issue of Rollout algorithm is that it needs at least one lower-bounding heuristic in order to estimate correctly each action value. Unfortunately, there are not many lower bounds for DEC-POMDPs. Indeed, any suboptimal algorithm is a lower bound of the optimal policy. We thus choose to use some literature algorithms known to be suboptimal: the random and the greedy policies are both well known in the domain. Two more we use are the dynamic programming algorithm from Hansen *et al.* (Hansen, Bernstein, & Zilberstein 2004) and the memory bounded algorithm we mentioned earlier in this paper and that provides a very good lower bound of the optimal policy. Let us now see this two approaches.

Optimal Dynamic Programming

The dynamic programming (DP) algorithm iteratively constructs a set of *policy trees* Q^t for each horizon t based on the set of policy trees of previous horizon $t-1$. Each *depth-t* policy tree $\delta^t = \langle q_1^t, \ldots, q_n^t\rangle$ is comprised of n trees q_i^t, one for each agent and each tree q_i^t consists of nodes, labeled with actions, and edges, labeled with observations. To execute a policy tree q_i^t agent A_i begins at the root, takes the corresponding action, then follows the branch labeled with the observation got, repeating the process for t steps.

Unfortunately, computing some value-maximizing δ^t is made complex by doubly exponential growth in the set of all depth-t trees. Even for simple problems, this number quickly grows so large that it results in an infeasible exhaustive enumeration. Thus, the DP algorithm instead selectively prunes away policies as it goes in order to conserve only non dominated policies.

Indeed, the DP algorithm outputs a sequence of depth-t policy trees, $\delta^t = \langle q_1^t, \ldots, q_n^t\rangle$, each of which maximizes the value for the given agent, by looping over two main steps, *backup* and *pruning*:

1. *Backup:* Generation of the set Q_i^t of all depth-t policy-trees for agent A_i given the set Q_i^{t-1} of policy trees of previous iteration, i.e. considering every possible action/observation-transition into any depth-t tree in Q_i^{t-1}.

2. *Pruning:* Considering each tree of the set Q_i^t, and considering all belief states reachable from an initial belief state, this step eliminates all *dominated* trees, in the sense that some other policy is more valuable given any possible multiagent belief.

Memory Bounded Dynamic Programming

The Memory Bounded Dynamic Programming (MBDP) algorithm is a point based dynamic programming algorithm that restrains the number of policies kept in memory at each iteration. Every iteration consists of the following steps:

1. *Backup:* Generation of the set Q_i^t of the $|\mathcal{A}_i|maxTree^{|\Omega_i|}$ depth-t policy-trees for agent A_i given the set Q_i^{t-1} of policy trees of previous iteration, i.e. considering every possible action/ observation-transition into any depth-t tree in Q_i^{t-1}.
2. *Belief Selection:* Selection of a finite set of belief points according to a portfolio of heuristics.
3. *Tree Selection:* Considering each tree of the set Q_i^t, and considering all belief states selected at the previous step, this step selects the $maxTree$ best-valued trees of Q_i^t to keep them for the next iteration.

Here, notice that, (i) the larger is $maxTree$ the better is the approximation of the policy, and (ii) the better are the heuristics to select the "good" beliefs points where evaluating the policies, the better is the approximation. As the choice of the heuristics is an entire area of research, we keep basic heuristics (random, Q_{MDP}) and only range $maxTree$ and the horizon to vary the approximate ratio of the MBDP algorithm. For further details about MBDP, the interested reader is invited to refer to the original work of Seuken & Zilberstein (Seuken & Zilberstein 2007b; 2007a).

Monte Carlo Trials

Monte Carlo trials are known to be really time consuming. However, recent advances in this domain considerably reduce the time to simulate. Novel techniques, called *particle filters*, allow simulating several Monte Carlo trials in one shot while using fixed amount of memory, and providing bounds on the estimation of the probability distribution.

Thrun (Thrun 1999) successfully applied them in POMDPs. He estimated the belief state at each time step using particle filters and then selected the best action using a dynamic programming algorithm and a sample-based version of nearest neighbor to generalize. Particle filters are a sample-based variant of Bayes filters, which recursively estimate posterior densities, or beliefs $\boldsymbol{b}$ in our case, over the state s of a dynamic system (Fox *et al.* 2001).:

$$\boldsymbol{b}^{t+1}(s') = \eta\mathcal{O}(\boldsymbol{o}|s,\boldsymbol{a},s')\int \mathcal{T}(s'|s,\boldsymbol{a})\boldsymbol{b}^t(s)\,\mathrm{d}s$$

As defined previously, $\boldsymbol{o}$ is the observation made and $\boldsymbol{a}$ the control chosen in state s establishing the dynamics of the system. Particle filters represent beliefs by sets S of N weighted samples $\langle s^{(i)}, w^{(i)}\rangle$. each $s^{(i)}$ is a sample representing a state, and the $w^{(i)}$ are non-negative numerical factors called *importance weights*, which sum up to one. The basic form of the particle filter realizes the recursive Bayes filter according to a sampling procedure, often referred to as sequential importance sampling with resampling (SISR):

1. *Resampling:* Draw with replacement a random sample (or state) s from the set S (representing the current belief $\boldsymbol{b}(s)$) according to the (discrete) distribution defined through the importance weight $w^{(i)}$;
2. *Sampling:* Use current s and the action $\boldsymbol{a}$ to sample s' according to the transition function $\mathcal{T}(s'|s,\boldsymbol{a})$, which describes the dynamics of the agent;
3. *Importance Sampling:* Weight the sample s' by the observation likelihood $w' = \mathcal{O}(\boldsymbol{o}|s,\boldsymbol{a},s')$. This likelihood is extracted from a model of the agent sensors (e.g. camera, sonar, laser range-finder) and a map of the environment.

In some cases, however, particle filters may have a very low rate of update when the chosen number of samples is too large for example. Many improvements of particle filters can then be found in the literature. One of the most interesting improvements is the adaptive and real-time particle filter of Kwok *et al.* (Kwok, Fox, & Meila 2003). Indeed, these authors show first how to choose dynamically the number of samples needed to ensure a bound on the error made on the belief approximation regarding to the Kullback-Leibler (KL) distance. Then, they also show how to optimize the use of computational resources when the update rate of the particle filter is lower than the rate of incoming observations. The former improvement can help to derive error bounds on the estimation of the belief while the latter can helps in computing online a policy for the agent. However, even if we aim to compute an online policy, we leave the latter improvement for future work.

Adaptive Particle Filters

As described by Fox in (Fox 2001), the key idea of adaptive particle filters is to bound the error introduced by the sample based representation of the particle filter. To derive this bound, he assumes that the true posterior is given by a discrete, piecewise constant distribution. For such a representation he shows a way to determine the number of samples so that, with probability $1-\delta$, the distance between the maximum likelihood estimate based on the samples and the true posterior does not exceed a specified threshold ε. He denotes the resulting approaches the KLD-sampling algorithm since the distance is measured with the KL-distance.

Suppose the estimated distribution has k bins (i.e. the belief is over k states). For fixed error bounds ε and δ, the formula (1) computes the required number of samples N as a function of k (Fox 2001):

$$N = \frac{k-1}{2\varepsilon}\left[1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}}z_{1-\delta}\right]^3 \qquad (1)$$

where $z_{1-\delta}$ is the upper $1-\delta$ quantile of a normal distribution $\mathcal{N}(0,1)$. As we can see, the required number of samples is proportional to the inverse of the error bound ε, and to the first order linear in the number k of bins with support. KLD-sampling estimates k by the number of states that are represented by at least one particle.

To integrate KLD-sampling in the standard particle filter algorithm, we will use a coarse, fixed grid to approximate the belief distribution over states. Then, during the prediction step of the particle filter (step 2 in SISR procedure), the algorithm determines whether a newly generated sample falls into an empty cell of the grid or not (the grid is reset after each filter update). If the grid cell is empty, the number of bins k is incremented and the cell is marked as non-empty. After each sample, the number of required samples is updated using equation (1) with the updated number of bins. Adding samples is stopped as soon as no empty bins are filled, since k does not increase and consequently N converges.

Thus KLD-sampling initially uses a large number of samples when almost nothing is known about current state, but decreases the number as the knowledge about the current state increases. Conversely, the number of samples used to estimate the distribution over other agents' policies may increase as time passes until it reaches a certain threshold to be set carefully. The interested reader can refer to (Fox 2001) for details on adaptive particle filters. Now let us see how to use particle filters in the context of DEC-POMDPs.

Parallel Rollout Approach

As depicted by Algorithm 1, the basis of our approach is, at each decision step, to compute the exact joint belief after executing each action and receiving each observation according to the current belief. Then, a set of approximate joint policies $\Pi = \langle \pi_1(\boldsymbol{b}^0), \ldots, \pi_n(\boldsymbol{b}^0) \rangle$ is computed over the set of next reachable beliefs over states $\boldsymbol{b}'$ (Algorithm 2). Then, using particle filters to estimate the belief over the state, we can estimate each further action value according to each policy by Monte-Carlo trials in a given amount of time. Selecting the action with the maximal value in $\boldsymbol{b}'$ may then lead to an overall improvement of each heuristic used regarding to the quality of the estimation.

Algorithm 1 Rollout for DEC-POMDPs

Require: A set of heuristics $\Pi = \langle \pi_1, \ldots, \pi_n \rangle$, A initial belief $\boldsymbol{b}^0$, an horizon T, a fix amount of time T_f.

Ensure: An action to perform in the current belief state $\boldsymbol{b}^0$.

function DEC-ROLLOUT(Π,$\boldsymbol{b}^0$,T,T_f)
  $t_0 =$ CURRENTTIME()
  while CURRENTTIME() - $t_0 \leq T_f$ **do**
    for all $\boldsymbol{a} \in \boldsymbol{\mathcal{A}}$, $\boldsymbol{o} \in \boldsymbol{\Omega}$ **do**
      $\boldsymbol{b}^1 \leftarrow$ UPDATE($\boldsymbol{b}^0, \boldsymbol{a}, \boldsymbol{o}$)
      $\Upsilon \leftarrow$ MCETRIAL($\Pi, \boldsymbol{b}^1, T$)
    end for
  end while
  $\boldsymbol{a} \leftarrow \arg\max_{\boldsymbol{a} \in \boldsymbol{\mathcal{A}}} \sum_{\boldsymbol{o} \in \boldsymbol{\Omega}} \Pr(\boldsymbol{o}|\boldsymbol{b}^0, \boldsymbol{a}) \max_{\pi \in \Pi} \Upsilon[\pi]$
  return $\boldsymbol{a}$
end function

Notice that the quality of the approximation produced by the parallel Rollout approach depends on the number of particles used. Indeed, we have seen in Section on adaptive particle filters that, if resampling is made correctly, according to (1), error in estimating the belief state is beneath ε with probability $1 - \delta$. We conjecture then that a bound on the error is theoretically obtainable. However, the proof of this statement remains for the moment an issue of future work.

Algorithm 2 Monte-Carlo Estimation of each $\pi \in \Pi$

Require: A set of policies $\Pi = \langle \pi_1, \ldots, \pi_n \rangle$, A belief $\boldsymbol{b}^1$, an horizon T.

Ensure: A set of expected value for each policy $\Upsilon = \langle V_{\pi_1}(\boldsymbol{b}^1), \ldots, V_{\pi_n}(\boldsymbol{b}^1) \rangle$.

function MCETRIAL(Π,$\boldsymbol{b}^1$,T)
  for all $\pi \in \Pi$ **do**
    for $t = 1$ **to** T **do**
      $(\boldsymbol{b}^{t+1}, \boldsymbol{a}) \leftarrow$ STEP($\boldsymbol{b}^t, \pi(\boldsymbol{b}^t)$)
      $V_\pi(\boldsymbol{b}^{t+1}) \leftarrow \mathcal{R}(\boldsymbol{b}^{t+1}, \boldsymbol{a}) + V_\pi(\boldsymbol{b}^t)$
    end for
  end for
  return Υ
end function

function STEP(belief $\boldsymbol{b}^t$, action $\boldsymbol{a}$)
  RESAMPLEPF($\boldsymbol{b}^t$) //Step 1.
  SAMPLEPF($\boldsymbol{a}$) //Step 2.
  $\boldsymbol{o} \leftarrow$ GETOBS()//From the model
  IMPORTANCESAMPLEPF($\boldsymbol{o}$) //Step 3.
  return $(\boldsymbol{b}^{t+1}, \boldsymbol{a})$
end function

Experiments

As stated by Seuken & Zilberstein (Seuken & Zilberstein 2007b), most researchers on DEC-POMDPs report performance results for the multiagent tiger problem (Nair *et al.* 2003) or the multiagent broadcast channel problem (Hansen, Bernstein, & Zilberstein 2004). The former problem has 2 agents, 2 states, 3 actions and 2 observations while the later has 2 agents, 4 states, 2 actions and 2 observations. Amato *et al.* (Amato, Bernstein, & Zilberstein 2007) also report results on a slightly larger problem called Recycling Robots with 2 agents, 4 states, 3 actions and 2 observations but this last problem has independent observations and independent transitions and is thus simpler than the two others. As a consequence, we experiment our algorithm on the first two problems.

Multiagent Tiger Problem In the Multiagent Tiger problem (MAT) introduced by Nair *et al.* (Nair *et al.* 2003), there are two doors. Behind one door is a tiger and behind the other is a large treasure. Each agent may open one of the doors or listen. If either agent opens the door with the tiger behind it, a large penalty is given. If the door with the treasure behind it is opened and the tiger door is not, a reward is given. If both agents choose the same action (i.e., both opening the same door) a larger positive reward or a smaller penalty is given to reward this cooperation. If an agent lis-

tens, a small penalty is given and an observation is seen that is a noisy indication of which door the tiger is behind. While listening does not change the location of the tiger, opening a door causes the tiger to be placed behind one of the door with equal probability.

Multiagent Broadcast Channel Problem In the Multiagent Broadcast Channel problem (MABC) defined for DEC-POMDPs by Hansen *et al.* (Hansen, Bernstein, & Zilberstein 2004), nodes need to broadcast messages to each other over a channel, but only one node may broadcast at a time, otherwise a collision occurs. The nodes share the common goal of maximizing the throughput of the channel.

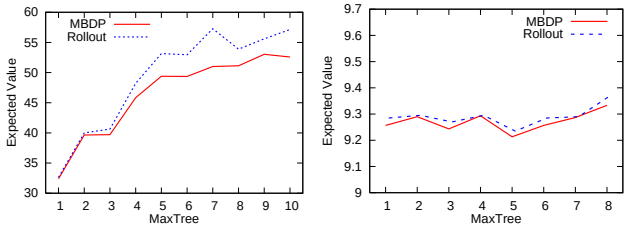
The process proceeds in discrete time steps. At the start of each time step, each node decides whether or not to send a message. The nodes receive a reward of 1 when a message is successfully broadcast and a reward of 0 otherwise. At the end of the time step, each node receives a noisy observation of whether or not a message got through. The message buffer for each agent has space for only one message. If a node is unable to broadcast a message, the message remains in the buffer for the next time step. If a node i is able to send its message, the probability that its buffer will fill up on the next step is pi. Our problem has two nodes, with $p_1 = 0.9$ and $p_2 = 0.1$.

Results

We implemented the Rollout approach and performed intensive experimental tests. The algorithm has two key parameters that affect performance regarding to the given problem. One parameter is the desired level of approximation of the memory bounded lower bound, $maxTree$, which defines the memory requirements. The second parameter is the number of steps T we look ahead in order to select the next action. Whereas optimal dynamic programming horizon T is bounded by 3 for time and memory consumption reasons, the MBDP algorithm allows to choose any horizon to look ahead. However, as we will see later, this parameter is domain dependant and really influences on expected value.

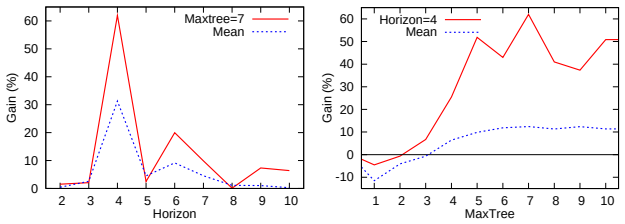
To evaluate our Rollout algorithm, we compared it to MBDP (Seuken & Zilberstein 2007b) that we also used as lower bound in order to measure the improvement brought by the Rollout approach. To perform a consistent comparison, we used an online derivative of the memory bounded dynamic programming algorithm. This online derivative version of MBDP consists in choosing and executing the first action of the policy returned by the algorithm at each decision step while looking a fixed number of steps forward. As soon as a new decision is needed, the algorithm is relaunched from scratch. We also used an online derivative of standard dynamic programming for DEC-POMDPs with a horizon of 2, as a second lower bound.

Results average 500 runs of the online MBDP and the Parallel Rollout algorithm. Figure 1 shows expected reward for the two problems according to the approximation level used and mainly gives evidence that the Rollout approach has a modest contribution to the improvement of the base policy for the MABC problem. This mainly due to the fact that the MBDP algorithm is near-optimal for the MABC problem. However, results for the tiger problem seems more promising (see Figure 1(a)) and particularly for $maxTree = 7$. This result is confirmed by Figure 2(a) which shows that using a $maxTree = 7$ for the TIGER problems leads to a significantly better gain than other approximation levels, even greater[1]. The second interesting result in Figure 2(a) is that there is no need to plan for horizon larger than 4 to have good practical results. This is also confirmed by the results in Figure 2(b) that clearly shows that online planning for the TIGER problem needs only to look 4 steps further[2] in order to guaranty a significantly better policy (up to 60%).



(a) TIGER (2ag, 2s, 3a, 2o) (b) MABC (2ag, 4s, 2a, 2o)

Figure 1: Expected Value of MBDP vs. Rollout depending on the approximation of the lower bound



(a) TIGER Gain vs Horizon (b) TIGER Gain vs MaxTree

Figure 2: Gain of Rollout over MBDP

Conclusion

To our knowledge, the algorithm presented in this paper is the second online algorithm for DEC-POMDPs. The first was presented by Emery-Montemerlo *et al.* (Emery-Montemerlo *et al.* 2004) but was outperformed by an online version of MBDP as stated by Seuken in (Seuken & Zilberstein 2007b), and that's why we do not present results about this algorithm here. However, our experimental study of our algorithm against the online derivative of MBDP shows that a significant improvement is made on current literature problems with appropriate parameters.

Performance of the Rollout approach opens interesting perspectives for research in the field of approximation for DEC-POMDPs. Indeed, because of its guarantee to obtain better results than the best results that can bring each of the

[1]Mean is over a $maxTree$ from 1 to 10.

[2]Mean is over a horizon from 2 to 10.

heuristics employed, the parallel Rollout approach seems, in these earlier stages of work, to be an interesting avenue to improve any suboptimal policy for DEC-POMDPs. However, as soon as heuristics become good enough to bring out near-optimal values, the Rollout's contribution becomes more modest since the cost of evaluating many heuristics is not balanced by the induced improvement.

As future work, there are three main extensions that may be considered. First, as depicted by Algorithm 1, agents still reason about a joint belief state. This implies a belief synchronisation at each step that is not necessarily desired. We must then consider ways to decentralized this belief in further experiments as done by Roth *et al*. (Roth, Simmons, & Veloso 2005) for example. The second one, as discussed earlier, consists of using an online update of particle filters so that we can have more precise belief over states without a costly update during Monte-Carlo Estimation. Third, an increase of the depth of search in the tree of next actions to more than 1 (Line 9 of Algorithm 1) by using an anytime error minimization search as Ross & Chaib-draa (Ross & Chaib-draa 2007) did for POMDPs could be an improvement of the algorithm. Finally, some work can also be done on heuristics that select reachable beliefs and their estimation in approximation algorithms.

Acknowledgments

The authors would like to thank Julien Laumônier, Abdeslam Boularias and Jilles Dibangoye for the helpful discussions we have had together concerning the topic of this paper.

References

Amato, C.; Bernstein, D. S.; and Zilberstein, S. 2007. Optimizing Memory-Bounded Controllers for Decentralized POMDPs. In *Proc. of Uncertainty in Artificial Intelligence*.

Bernstein, D. S.; Zilberstein, S.; and Immerman, N. 2000. The Complexity of Decentralized Control of Markov Decision Processes. In *Proc. of Uncertainty in Artificial Intelligence*, 32–37.

Bertsekas, D. P.; Tsitsiklis, J. N.; and Wu, C. 1997. Rollout Algorithms for Combinatorial Optimization. *Journal of Heuristics* 3(3):245–262.

Chang, H. S.; Givan, R.; and Chong, E. K. P. 2004. Parallel Rollout for Online Solution of POMDPs. *Discrete Event Dynamic Systems* 14(3):309–341.

Emery-Montemerlo, R.; Gordon, G. J.; Schneider, J. G.; and Thrun, S. 2004. Approximate Solutions for Partially Observable Stochastic Games with Common Payoffs. In *Proc. of the International Joint Conference on Autonomous Agents and Multi-Agent Systems*.

Fox, D.; Thrun, S.; Burgard, W.; and Dellaert, F. 2001. *Particle Filters for Mobile Robot Localization*. Springer. chapter 19.

Fox, D. 2001. KLD-Sampling: Adaptive Particle Filters. In *Proc. of Advances in Neural Information Processing Systems*, 713–720.

Hansen, E. A.; Bernstein, D. S.; and Zilberstein, S. 2004. Dynamic Programming for Partially Observable Stochastic Games. In *Proc. of Association for the Advancement of Artificial Intelligence*, 709–715.

Kwok, C.; Fox, D.; and Meila, M. 2003. Adaptive Real-Time Particle Filters for Robot Localization. In *Proc. of the IEEE International Conference on Robotics & Automation*.

Nair, R.; Tambe, M.; Yokoo, M.; Pynadath, D. V.; and Marsella, S. 2003. Taming Decentralized POMDPs: Towards Efficient Policy Computation for Multiagent Settings. In *Proc. of the International Joint Conference on Artificial Intelligence*, 705–711.

Peshkin, L.; Kim, K.-E.; Meuleau, N.; and Kaelbling, L. P. 2000. Learning to Cooperate via Policy Search. In *Proc. of Uncertainty in Artificial Intelligence*, 489–496.

Pynadath, D. V., and Tambe, M. 2002. The Communicative Multiagent Team Decision Problem: Analyzing Teamwork Theories and Models. *J. Artif. Intell. Res. (JAIR)* 16:389–423.

Rabinovich, Z.; Goldman, C. V.; and Rosenschein, J. S. 2003. The Complexity of Multiagent Systems: the Price of Silence. In *Proc. of the International Joint Conference on Autonomous Agents & Multiagent Systems*, 1102–1103.

Ross, S., and Chaib-draa, B. 2007. Aems: An anytime online search algorithm for approximate policy refinement in large pomdps. In *Proc. of the International Joint Conference on Artificial Intelligence*, 2592–2598.

Roth, M.; Simmons, R.; and Veloso, M. 2005. Reasoning About Joint Beliefs for Execution-Time Communication Decisions. In *Proc. of the International Joint Conference on Autonomous Agents and Multi-Agent Systems*.

Seuken, S., and Zilberstein, S. 2005. Formal Models and Algorithms for Decentralized Control of Multiple Agents. Technical report, Computer Science Department, University of Massachusetts, Amherst.

Seuken, S., and Zilberstein, S. 2007a. Improved Memory-Bounded Dynamic Programming for Dec-POMDPs. In *Proc. of Uncertainty in Artificial Intelligence*.

Seuken, S., and Zilberstein, S. 2007b. Memory-Bounded Dynamic Programming for Dec-POMDPs. In *Proc. of the International Joint Conference on Artificial Intelligence*, 2009–2015.

Szer, D., and Charpillet, F. 2005. An Optimal Best-First Search Algorithm for Solving Infinite Horizon DEC-POMDPs. In *Proc. of the European Conference on Machine Learning*, 389–399.

Thrun, S. 1999. Monte Carlo POMDPs. In *Proc. of Advances in Neural Information Processing Systems*, 1064–1070.