Real-Time Decision Making for Large POMDPs

Sébastien Paquet, Ludovic Tobin, and Brahim Chaib-draa

DAMAS Laboratory
Department of Computer Science and Software Engineering, Laval University
{spaquet;tobin;chaib}@damas.ift.ulaval.ca

Abstract. In this paper, we introduce an approach called RTBSS (Real-Time Belief Space Search) for real-time decision making in large POMDPs. The approach is based on a look-ahead search that is applied online each time the agent has to make a decision. RTBSS is particularly interesting for large real-time environments where offline solutions are not applicable because of their complexity.

1 Introduction

Partially Observable Markov Decision Processes (POMDPs) provide a very general model for sequential decision problems in partially observable environments. The main problem with POMDPs is that their complexity makes them applicable only on small environments.

In this paper, we introduce a novel idea for POMDPs that, to our knowledge, has not received a lot of attention. The idea is to use an online approach based on a look-ahead search to find the best action to execute at each cycle in the environment. By doing so, we avoid the overwhelming complexity of computing a policy for every possible situation the agent could encounter. Since there is no computation offline, the algorithm is immediately applicable to previously unseen environments, if the environments' dynamics are known. Also, since we need a fast online algorithm, we opted for a factored POMDP representation and a branch and bound strategy based on a limited depth first search instead of classical dynamic programming. The tradeoff obtained between the solution quality and the computing time is very interesting.

2 Belief State Value Approximation

The main idea of our online approach is to estimate the value of a belief state by constructing a tree where the nodes are belief states and where the branches are a combination of actions and observations (see Figure 1). To do so, we have defined a new function $\delta : B \times \mathbb{N} \rightarrow \mathbb{R}$ which is based on a depth-first search. The function takes as parameters a belief state b and a remaining depth d and returns an estimation of the value of b by performing a search of depth d. For the first call, d is initialized at D, the maximum depth allowed for the search.

$$\delta(b,d) = \begin{cases} U(b) & \text{, if } d = 0 \\ R(b) + \gamma \max_a \sum_{o \in \Omega} \left(P(o \mid b,a) \times \delta(\tau(b,a,o), d-1)\right) & \text{, if } d > 0 \end{cases} \tag{1}$$

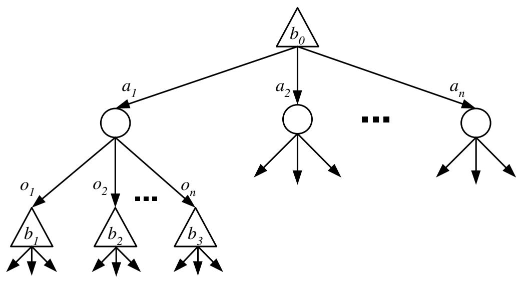

Fig. 1. A search tree.

When $d = 0$, we are at the bottom of the search tree. In this situation, we need a utility function $U(b)$, that gives an estimation of the real value of this belief state. If it is not possible to find a better utility function, we can use $U(b) = R(b)$. When $d > 0$, the value of a belief state at a depth of $D - d$ is the immediate reward for being in this belief state added to the maximum discounted reward of the subtrees underneath this belief state.

Finally, the agent's policy which returns the action the agent should do in a certain belief state is defined as:

$$\pi(b, D) = \underset{a}{\operatorname{argmax}} \sum_{o \in \Omega} P(o \mid b, a)\delta(\tau(b, a, o), D - 1) \tag{2}$$

3 RTBSS Algorithm

We have elaborated an algorithm, called RTBSS (see Algorithm 1), that is used to construct the search tree and to find the best action. Since it is an online algorithm, it must be applied each time the agent has to make a decision.

To speed up the search, our algorithm uses a "Branch and Bound" strategy to cut some sub-trees. The algorithm first explores a path in the tree up to the desired depth D and then computes the value for this path. This value then becomes a lower bound on the maximal expected value. Afterwards, for each node of the tree visited, the algorithm can evaluate with an heuristic function if it is possible to improve the lower bound by pursuing the search (Prune function at line 10). The heuristic function must be defined for each problem and it must always overestimate the true value. Moreover, the purpose of sorting the actions at line 13 is to try the actions that are the most promising first because it generates more pruning early in the search tree.

With RTBSS the agent finds at each turn the action that has the maximal expected value up to a certain horizon of D. As a matter of fact, the performance of the algorithm strongly depends on the maximal depth D of the search.

```
1: Function RTBSS($b$, $d$ , $rAcc$)
   Inputs: $b$: The current belief state.
           $d$: The current depth.
           $rAcc$: Accumulated rewards.
   Statics: $D$: The maximal depth.
            $bestValue$: The best value found up to now.
            $action$: The best action.
2: if $d = 0$ then
3:    $finalValue \leftarrow rAcc + \gamma^D \times U(b)$
4:    if $finalValue > bestValue$ then
5:       $bestValue \leftarrow finalValue$
6:    end if
7:    return $finalValue$
8: end if
9: $rAcc \leftarrow rAcc + \gamma^{D-d} \times R(b)$
10: if PRUNE($rAcc$, $d$) then
11:    return $-\infty$
12: end if
13: $actionList \leftarrow$ SORT($b$, $A$)
14: $max \leftarrow -\infty$
15: for all $a \in actionList$ do
16:    $expReward \leftarrow 0$
17:    for all $o \in \Omega$ do
18:       $b' \leftarrow \tau(b, a, o)$
19:       $expReward \leftarrow expReward + \gamma^{D-d} \times P(o|a, b) \times$ RTBSS($b', d - 1, rAcc$)
20:    end for
21:    if ($d = D \wedge expReward > max$) then
22:       $max \leftarrow expReward$
23:       $action \leftarrow a$
24:    end if
25: end for
26: return $max$
```

Algorithm 1: The RTBSS algorithm.

4 Experiments and Results

In this section we present the results we have obtained on two problems: *Tag* [1] and *RockSample* [2]. If we compare RTBSS with different existing approaches (see Table 1), we see that our algorithm can be executed much faster than all the other approaches. RTBSS does not require any time offline and takes only a few tenths of a second at each turn. On small problems the performance is not as good as the best algorithms but the difference is not too important. However, on the biggest problem, RTBSS is much better than HSVI.

Another advantage of RTBSS is its adaptability to environment changes, which enable agents using RTBSS to be deployed immediately and obtain good results even if the environment configuration has never been seen before. Offline

Problem	Reward	Time (s)
Tag (870s,5a,30o)		
Q_{MDP}	-16.75	11.8
RTBSS	-10.56	0.23[1]
PBVI [1]	-9.18	180880
BBSLS [3]	∽ -8.3	∽100000
BPI [4]	-6.65	250
HSVI [2]	-6.37	10113
Perséus [5]	-6.17	1670
RockSample[4,4] (257s,9a,2o)		
RTBSS	16.2	0.1[1]
PBVI [2][2]	17.1	∼ 2000
HSVI [2]	18.0	577
RockSample[5,5] (801s,10a,2o)		
RTBSS	18.7	0.1[1]
HSVI [2]	19.0	10208
RockSample[5,7] (3201s,12a,2o)		
RTBSS	22.6	0.1[1]
HSVI [2]	23.1	10263
RockSample[7,8] (12545s,13a,2o)		
RTBSS	20.1	0.2[1]
HSVI [2]	15.1	10266

Table 1. Comparison of our approach.

algorithms require recomputing a new policy for each new configuration while our algorithm could be applied right away.

Figure 2 compares our RTBSS algorithm with a version without pruning. On the first graphic, we see that the heuristic can greatly improve the performance of the search. The complexity is still exponential but it grows slower than the brute force version. The second graphic presents the performance of our algorithm in function of the depth of the search used. The rewards obtained are the same whether we use the pruning or not; the slight variation comes from randomness in the tests. We see that our algorithm does not require an heuristic to work properly. However, if we are able to find a good heuristic for a problem, it greatly improves the algorithm's speed.

The two graphics on Figure 2 are also used to choose the maximal depth D allowed. The depth is chosen experimentally depending on the problem and the amount of time available to make a decision, considering that at a certain depth, it might not be worth exploring much deeper. For example, on Figure 2, we can see that the agent does not get much better after depth 8 and the time needed is really small until depth 10, thus a depth of 10 would be a good maximal depth for this problem.

[1] It corresponds to the average time taken by the algorithm at each time it is called in a simulation.

[2] PBVI was presented in [1], but the result on *RockSample* was published in [2].

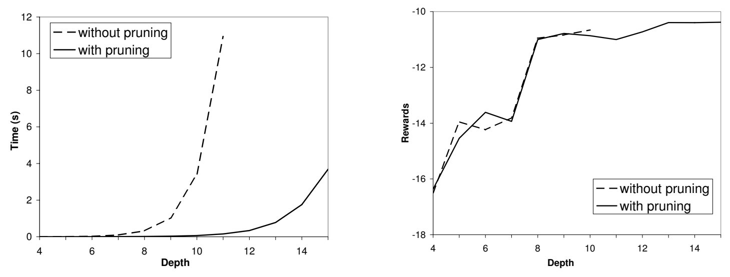

Fig. 2. Average deliberation time and reward on *Tag*.

5 Related Work and Conclusion

For POMDPs, very few researchers have explored the possibilities of online algorithms. [6] used a real-time dynamic programming approach to learn a belief state estimation by successive trials in the environment. The main differences are that they do not search in the belief state tree and they need offline time to calculate their starting heuristic based on the Q_{MDP} approach.

To summarize, this paper introduces RTBSS, an online POMDP algorithm useful for large, dynamic and uncertain environments. The main advantage of such a method is that it can be applied to problems with huge state spaces where other algorithms would take way too much time to find a solution. Our results show that RTBSS becomes better as the environment becomes bigger, compared to state of the art POMDP approximation algorithms. Also, because of its adaptability, RTBSS is more suited for environments in which the initial configurations can change and when the agent has to be deployed rapidly, compared to existing offline approaches.

References

1. Pineau, J., Gordon, G., Thrun, S.: Point-based value iteration: An anytime algorithm for pomdps. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-03), Acapulco, Mexico (2003) 1025–1032
2. Smith, T., Simmons, R.: Heuristic search value iteration for pomdps. In: Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence(UAI-04), Banff, Canada (2004)
3. Braziunas, D., Boutilier, C.: Stochastic local search for pomdp controllers. In: The Nineteenth National Conference on Artificial Intelligence (AAAI-04). (2004)
4. Poupart, P.: Exploiting Structure to Efficiently Solve Large Scale Partially Observable Markov Decision Processes. PhD thesis, University of Toronto (2005) (to appear).
5. Spaan, M.T.J., Vlassis, N.: A point-based pomdp algorithm for robot planning. In: In Proceedings of the IEEE International Conference on Robotics and Automation, New Orleans, Louisiana (2004) 2399–2404
6. Geffner, H., Bonet, B.: Solving large pomdps using real time dynamic programming (1998)