An Online POMDP Algorithm for Complex Multiagent Environments

Sébastien Paquet, Ludovic Tobin and Brahim Chaib-draa
DAMAS Laboratory
Department of Computer Science and Software Engineering
Laval University, Canada
{spaquet;tobin;chaib}@damas.ift.ulaval.ca

ABSTRACT

In this paper, we present an online method for POMDPs, called RTBSS (Real-Time Belief Space Search), which is based on a look-ahead search to find the best action to execute at each cycle in an environment. We thus avoid the overwhelming complexity of computing a policy for each possible situation. By doing so, we show that this method is particularly efficient for large real-time environments where offline approaches are not applicable because of their complexity. We first describe the formalism of our online method, followed by some results on standard POMDPs. Then, we present an adaptation of our method for a complex multiagent environment and results showing its efficiency in such environments.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*

General Terms

Algorithms, Experimentation

Keywords

POMDP, Online Search

1. INTRODUCTION

When faced with partially observable environments, a general model for sequential decision problems is to use the Partially Observable Markov Decision Processes (POMDPs). A lot of problems can be modelled with POMDPs, but very few can be solved because of their computational complexity (POMDPs are PSPACE-complete [16]). The main problem with POMDPs is that their complexity makes them applicable only on small environments. However, most problems of interest have a huge state space, which motivates the search for approximation methods [9]. This is especially the case for multiagent systems where we often have a huge state space with autonomous agents interacting with each other.

POMDPs have generated a lot of interest in the AI community and many approximation algorithms have been developed recently [5, 17, 18, 19, 21]. They all share in common the fact that they solve the problem offline. While these algorithms can achieve very good performances, they are not applicable on large multiagent problems.

In this paper, instead of computing a complete policy offline, we present an online approach based on a look-ahead search in the belief state space to find the best action to execute at each cycle in the environment. Our algorithm, called RTBSS (Real-Time Belief Space Search), only explores reachable belief states starting from the agent's current belief state [14]. This online exploration has to be as fast as possible, since our algorithm has to work under some real-time constraints. To achieve that, we opted for a factored POMDP representation and a branch and bound strategy. By pruning some branches of the search tree, the algorithm is able to search deeper, while still respecting the real-time constraint.

By doing an online search, 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. Other approaches have used an online search for POMDPs, but they were not immediately efficient without any offline computations. For example, the BI-POMDP algorithm needs the underlying MDP to be solved offline to choose in which order to expand the search [22]. Similarly, the BEL-RTDP algorithm [6] needs successive trials in the environment in addition to the solution for the underlyning MDP, thus it needs an offline training before becoming efficient.

In the second part of this paper, the basic RTBSS algorithm has been slightly modified to take the specificity of multiagent systems into consideration for the transition model and the maintenance of the belief state. The main idea is to abstract some of the dynamic parts of the environment in order to respect the real-time constraint.

In this article, we first describe the formalism of our online algorithm, followed by some results on standard POMDPs, then we present an adaptation of our method for a complex multiagent environment and some results showing its

AAMAS'05, July 25-29, 2005, Utrecht, Netherlands.

efficiency in such environments. The results show that it is possible to achieve relatively good performances by using a very short amount of time online. The tradeoff between the solution quality and the computing time is very interesting.

2. MOTIVATIONS

The development of our algorithm is in fact constrained by the following constraints:

- the algorithm has to be efficient in large state spaces;
- the environment is partially observable;
- an agent using this algorithm has to be efficient in previously unknown instances of an environment;
- the agent's response time has to respect a real-time constraint.

An example of problems with those constraints is an autonomous robot that evolves in unknown environments and where it has to decide which action to pursue in a timely manner even if it is the first time it sees the environment. Another example is the RoboCupRescue simulation [12] in which agents have to be immediately efficient in previously unseen cities. In this environment, agents have to act fast to stop the degradation of the rescue situation.

In such environments, a POMDP algorithm should guarantee a fast agent response time (for example, under 500ms). Moreover, an agent should be immediately efficient in any configuration of the modeled environment. This last constraint eliminates all offline approaches, because the agent does not have time to learn a complete policy before its execution. It is not realistic to let an offline algorithm take 2 days, for example, to compute a complete policy each time their is a little modification in the environment, because the situation can get worst rapidly. Those constraints motivated us to develop our online POMDP algorithm that can ensure a quick response time in a huge state space.

3. POMDP

In this section we describe POMDPs and then we introduce factored POMDPs. For a more detailed presentation of POMDPs, we refer the reader to [1, 10]. Formally, a POMDP is a tuple described as $\langle S, A, T, R, \Omega, O\rangle$ where:

- S is the set of all the environment states;
- A is the set of all possible actions;
- $T(s, a, s')$ is the probability of ending in state s' if the agent performs action a in state s;
- $R(s)$ is the reward associated with being in state s.
- Ω is the set of all possible observations;
- $O(s', a, o)$ is the probability of observing o if action a is performed and the resulting state is s'.

Since the environment is partially observable, an agent cannot perfectly distinguish in which state it is. To manage this uncertainty, an agent can maintain a belief state b which is defined as a probability distribution over S. $b(s)$ means the probability of being in state s according to belief state b. We will use B to denote the set of all possible belief states which is infinite and uncountable.

The agent also needs to choose an action to do in function of its current belief state. This action is determined by the policy $\pi : B \rightarrow A$, which is a function that maps a belief state to the action the agent should execute in this belief state. The optimal policy can be computed offline using well known algorithms such as the enumeration algorithm [20] or Witness [13]. Some interesting approximation algorithms like PBVI [17] and HSVI [19] allows finding an approximate solution much faster than optimal algorithms.

Formally, the value function of a belief state for an horizon of t is given by:

$$V_t(b) = R(b) + \gamma \max_a \sum_{o\in\Omega} P(o \mid b, a)\, V_{t-1}(\tau(b, a, o)) \quad (1)$$

$$R(b) = \sum_{s\in S} b(s)R(s) \quad (2)$$

$R(b)$ is the expected reward for the belief state b. The second part of equation 1 is the discounted expected future rewards. $P(o \mid b, a)$ is the probability of observing o if action a is performed in belief state b.

$$P(o \mid b, a) = \sum_{s'\in S} O(s', a, o) \sum_{s\in S} T(s, a, s')b(s) \quad (3)$$

Also, $\tau(b, a, o)$ is the belief state update function. It returns the resulting belief state if action a is done in belief state b and observation o is perceived. If $b' = \tau(b, a, o)$, then:

$$b'(s') = \eta O(s', a, o) \sum_{s\in S} T(s, a, s')b(s) \quad (4)$$

where η is a normalizing constant. Finally, the policy can be obtained according to:

$$\pi_t(b) = \operatorname*{argmax}_a \left[R(b) + \gamma \sum_{o\in\Omega} P(o \mid b, a)\, V_{t-1}(\tau(b, a, o)) \right] \quad (5)$$

3.1 Factored POMDP

The traditional POMDP model is not necessarily suited for large environments because it requires enumerating explicitly all the states. However, most environments can be described as a set of different features which allows representing the states much more compactly. The states can then be defined with a set of random variables. Let $X = \{X_1, \ldots, X_M\}$ be the set of M random variables that fully describe a state. Then $D_i = \operatorname{dom} X_i$ is the set of all possible values for the random variable X_i. We suppose that each variable has a finite number of possible values. Therefore, a state is defined by assigning a value to each variable: $s = \{X_1 = x_1, \ldots, X_M = x_M\}$ where $x_i \in D_i$. We also use a more compact representation $s = \{x_i\}_{i=1}^{M}$.

Moreover, we need to slightly modify the belief state definition according to the factored representation of states. We define b as a full joint probability distribution on all random variables: $b = \mathbf{P}(X_1, \ldots, X_M)$. In our approach, we consider that all the variables are independent; this is why we can reformulate the belief state as: $b = \mathbf{P}(X_1) \cdots \mathbf{P}(X_M)$. It follows that the probability of being in a state can easily be computed by doing the product of the variables' probabilities.

Of course, this is a very strong hypothesis. However, even if some variables are dependent, it is still possible to factorize

$$b = \left(\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.25 \\ 0.05 \\ 0.50 \\ 0 \\ 0.10 \\ 0.10 \end{bmatrix}, \begin{bmatrix} 0.10 \\ 0.30 \\ 0.15 \\ 0.45 \end{bmatrix} \right)$$

Figure 1: An example of a belief state.

the belief state with minimal degradation of the quality of the solution. In this case, we would have to regroup the variables in dependant subsets. To do so, there are some methods that could be used to find the best factorization [4, 18], but we do not go further in this article.

To illustrate how the factorization works, let's suppose an environment that can be described by 3 variables X_1, X_2 et X_3. Each of these variables can respectively take 5, 6 and 4 different values; the environment then has 120 states. A possible belief state b is shown in Figure 1. Each vector represents the probability for each value of each variable.

By maintaining the probabilities on variables instead of states, it is much easier to update the belief state and it can be used for approximation methods. In the next section, we describe the benefits of a factored representation to avoid having sums defined on all states for the equations 1 to 3. This allows greatly improving the computation speed.

3.2 Using the Factored Representation

Firstly, to take advantage of the factored representation, we define a function $\omega : B \rightarrow \mathbb{P}\, S$:

$$\omega(b) = \{\{x_i\}_{i=1}^{M} \mid (\forall\, x_i)\ P_b(X_i = x_i) > 0\} \tag{6}$$

This function returns all the states the agent could be in, according to a belief state. We know that a state is impossible if one of the variables has a probability of zero according to b. If the variables are ordered approximately according to their certainty, this subset of states can be constructed quite rapidly because each time we encounter a variable with a zero probability, we can immediately exclude all the corresponding states. The following equation can then be computed much more rapidly than equation 2:

$$R(b) = \sum_{s \in \omega(b)} R(s) b(s) \tag{7}$$

The only difference in this equation is that the summation is defined on a subset of states ($\omega(b)$) instead of the whole state space. The less uncertainty the agent has, the smaller is the subset of possible states and the faster is the computation of equation 7 compared to equation 2.

For example, suppose the agent wants to know the reward of being in the belief state illustrated in Figure 1. With the traditional approach, we would have to iterate on all 120 states. However, when we observe the belief state, we immediately see that many states are impossible. With our approach, we only have to iterate on 20 states.

Now that we consider only possible states, we would also like to have a function that returns the states that are reachable from a certain belief state. For this, we define a new function $\alpha : A \times B \times \Omega \rightarrow \mathbb{P}\, S$ that takes as parameters the current belief state b, the action performed a and the observation perceived o, and returns all reachable states.

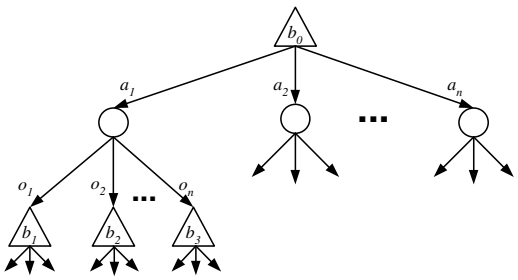

Figure 2: A search tree.

$$\begin{aligned} \alpha(a, b, o) \quad &= \quad \{s' \mid (\forall\, s \in \omega(b))\ T(s, a, s') \neq 0 \\ &\quad\ \wedge\ O(s', a, o) \neq 0\} \end{aligned} \tag{8}$$

The probability of making an observation can also be expressed using α and ω:

$$P(o \,|\, a, b) = \sum_{s' \in \alpha(a,b,o)} O(s', a, o) \sum_{s \in \omega(b)} T(s, a, s') b(s) \tag{9}$$

The belief state update function $\tau(b, a, o)$ can also be computed more efficiently with a factored representation, we refer the reader to [4, 18] for more details.

4. ONLINE DECISION MAKING

As we mentioned above, the different algorithms that use dynamic programming to solve the problem offline can only be applied on small problems. Even state of the art approximation algorithms can at best be applied on medium sized problems. Instead of computing a policy offline, we adopted an online approach where the agent only explores belief states that can be reached from the current belief state. This allows avoiding searching for a complete policy, thus avoiding a lot of computations. The advantage of such a method is that it can be applied on very large problems. It also allows having a model for decision making in large stochastic environments. In this section, we explain in detail how the algorithm for online decision-making works.

4.1 Belief State Value Approximation

In section 3, we described how it was possible to exactly compute the value of a belief state using dynamic programming (equation 1). In this section, we instead explain how we estimate the value of a belief state for our online approach by using a look-ahead search. The main idea is to construct a tree where the nodes are belief states and where the branches are a combination of actions and observations (see Figure 2). To do so, we have defined a new function that takes as parameters a belief state b and a 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.

$$\delta(b, d) = \begin{cases} U(b) & \text{, if } d = 0 \\ R(b) + \gamma \max\limits_{a} \sum\limits_{o \in \Omega} (P(o \mid b, a) & \\ \quad \times \delta(\tau(b, a, o), d - 1)) & \text{, if } d > 0 \end{cases} \tag{10}$$

where $R(b)$ is computed using equation 7 and $P(o \,|\, b, a)$ using equation 9.

```
1: Function RTBSS(b, d , rAcc)
   Inputs: b: The current belief state.
           d: The current depth.
           rAcc: Accumulated rewards.
   Statics: D: The maximal depth search.
           bestValue: The best value found in the search.
           action: The best action.
2: if d = 0 then
3:    finalValue ← rAcc + γ^D × U(b)
4:    if finalValue > bestValue then
5:       bestValue ← finalValue
6:    end if
7:    return finalValue
8: end if
9: rAcc ← rAcc + γ^(D−d) × R(b)
10: if Prune(b, rAcc, d) then
11:    return −∞
12: end if
13: actionList ← Sort(b, A)
14: max ← −∞
15: for all a ∈ actionList do
16:    expReward ← 0
17:    for all o ∈ Ω do
18:       b' ← τ(b, a, o)
19:       expReward ← expReward + γ^(D−d) × P(o | a, b)×
                      RTBSS(b', d − 1, rAcc)
20:    end for
21:    if (d = D ∧ expReward > max) then
22:       max ← expReward
23:       action ← a
24:    end if
25: end for
26: return max
```

Algorithm 1: The RTBSS algorithm.

When $d = 0$, we are at the bottom of the search tree. In this situation, the value of this belief state is given by a utility function $U(b)$. This function gives an idea of the real value of this belief state (if the function $U(b)$ was perfect, their would be no need for a search). This utility function has to be defined for each problem.

When $d > 0$, the value of a belief state at a depth of $D-d$ is simply the immediate reward for being in this belief state added to the maximum discounted reward of the subtrees underneath this belief state.

Finally, the action to perform in a certain belief state is given by:

$$\pi(b, D) = \underset{a}{\operatorname{argmax}} \sum_{o \in \Omega} P(o \mid b, a)\delta(\tau(b, a, o), D - 1) \quad (11)$$

4.2 RTBSS Algorithm

We have elaborated an algorithm, called RTBSS (Real-Time Belief State Search), that is used to construct the search tree and to find the best action (see Algorithm 1). Its name is due to the fact that it does a search in the belief state space online while satisfying a real-time constraint. 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 to a depth of D. This is represented by the Prune function at line 10. The heuristic function returns an estimation of the best utility value that could be found if the search was pursued. Thus, if according to the heuristic, it is impossible to find a value that is better than the lower bound by continuing the search to a depth of D, the algorithm backtracks and explores another action. If not, the search continues because there is a non-zero probability that the best solution is hiding somewhere in the sub-tree.

Moreover, the heuristic function used in the Prune function must be defined for each problem. Note that such heuristic should overestimate the true value in order to guarantee that all the pruned sub-trees do not contain the best solution.

To link the algorithm with the equations presented, notice that the line 19 of Algorithm 1 corresponds to the last part of equation 10, where δ is replaced by RTBSS. Also, the function $\tau(b, a, o)$ at line 18 returns the new belief state if o is perceived after the agent has done action a in belief state b. Note that the Sort function at line 13 sorts the actions 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 depth of the search. The complexity of the search is in the worst case of: $O((\mid A \mid \times \mid \Omega \mid)^D)$. In this case, no pruning is done, consequently with a good heuristic, it is possible to do much better.

Therefore, our algorithm is efficient if the number of actions and observations is kept small. Otherwise, the search cannot be done deep enough since the branching factor becomes too big. If there are many observations, it is possible to use a sampling of the observations in order to explore only the most probable observations [14]. However, for this paper, we only considered exploring all the observations.

5. EXPERIMENTS: STANDARD POMDPS

In this section we present the results we have obtained on two problems: *Tag* [17] and *RockSample* [19]. 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, depending on the problem.

We also note that our algorithm is faster than the popular approximation method Q_{MDP} and achieves a better result. Q_{MDP} consists in solving the underlying MDP as if it was totally observable.

Moreover, the most interesting results are obtained when the problem becomes bigger. If we look at the *RockSample* problems, RTBSS is really close on the first three smaller problems, but it is much better on the biggest problem. RTBSS is better because HSVI has not had time to converge in the allowed time. This shows an important advantage of our approach on big environments.

However, the *Tag* and *RockSample* problems do not totally do justice to our algorithm because we can affirm that it is better to apply an offline algorithm in order to have a better solution, even if we need to wait a few hours or days before having the solution. However, each time the environment slightly changes, we have to wait another few

Problem	Reward	Offline Time(s)	Online Time(s)
Tag (870s,5a,30o)			
Q_{MDP}	-16.75	11.8	-
RTBSS	-10.56	0	0.23[1]
PBVI [17]	-9.18	180880	-
BBSLS [5]	$\backsim$ -8.3	$\backsim$100000	-
BPI [18]	-6.65	250	-
HSVI [19]	-6.37	10113	-
Perséus [21]	-6.17	1670	-
RockSample[4,4] (257s,9a,2o)			
RTBSS	16.2	0	0.1[1]
PBVI [19][2]	17.1	$\sim$ 2000	-
HSVI [19]	18.0	577	-
RockSample[5,5] (801s,10a,2o)			
RTBSS	18.7	0	0.1[1]
HSVI [19]	19.0	10208	-
RockSample[5,7] (3201s,12a,2o)			
RTBSS	22.6	0	0.1[1]
HSVI [19]	23.1	10263	-
RockSample[7,8] (12545s,13a,2o)			
RTBSS	20.1	0	0.2[1]
HSVI [19]	15.1	10266	-

Table 1: Comparison of our approach. The reward for RTBSS is the average reward over 1000 simulations.

hours or days. Thus, if the environment is not exactly the same from one execution to another, the offline approaches become really expensive. We have used those two environments because they were popular and because they enabled us to compare the RTBSS's performances with the best offline performances. Our results show that even if they have all the time they want, offline approaches have some difficulties to catch up with the performances obtained by our RTBSS algorithm on big environments.

Another huge advantage of our algorithm is its adaptability to environment changes. Let's suppose that we have the *RockSample* problem but at each new execution in the environment, the initial position of the rocks changes or the shape of the grid changes. With offline algorithms, it would require recomputing a new policy for the new configuration while our algorithm could be applied right away. Therefore, our RTBSS algorithm is more suited to environments in which the initial configurations can change and when the agent has to be deployed rapidly. For instance, in rescue operations, agents have to be deployed immediately, they do not have the time to learn a good policy.

6. EXPERIMENTS: ROBOCUPRESCUE

In this section, we present results in a mush more complete environment: the RoboCupRescue simulation. This environment consists of a simulation of an earthquake happening in a city [12]. The goal of the agents (representing firefighters, policemen and ambulance teams) is to minimize the damages caused by a big earthquake, such as civilians buried, buildings on fire and roads blocked. In this dynamic environment, there are a lot of uncertainties that complicate the work of the agents.

[1]It corresponds to the average time taken by the algorithm at each time it is called to find an action.

[2]PBVI was presented in [17], but the result on *RockSample* was published in [19].

For this article, we consider only the policeman agents. Their task is to clear the most important roads as fast as possible, which is crucial to allow the other rescuing agents to perform their tasks. However, it is not easy to determine how the policemen should move in the city because they do not have a lot of information. They have to decide which road to prioritize and they have to coordinate themselves so that they do not try to clear the same road.

In this section, we present how we applied our approach in the RoboCupRescue simulation. In fact, we are interested in only a subproblem which can be formulated as: Having a partial knowledge of the roads that are blocked or not, the buildings in fire and the position of other agents, which sequence of actions should a policeman agent perform?

6.1 RoboCupRescue viewed as a POMDP

We present how we modelled the RoboCupRescue as a POMDP, from the point of view of a policeman agent. The different actions an agent can do are: *North*, *South*, *East*, *West* and *Clear*. A state can be described by approximately 1500 random variables, depending on the simulation:

- *Roads*: There are approximately 800 roads in a simulation and they can either be blocked or cleared.
- *Buildings*: There are approximately 700 buildings in a simulation and they can either be on fire or not.
- *Agents position*: An agent can be on any of the 800 roads and there's usually 30-40 agents.

If we estimate the number of states, we obtain $2^{800} \times 2^{700} \times 800^{30}$ states. However, a strong majority of them are not possible and will not ever be reached. The state space of RoboCupRescue is too important to even consider applying offline algorithms. We must therefore adopt an online method that allows finding a good solution very quickly.

6.2 Application of RTBSS on RoboCupRescue

This section presents how we have applied RTBSS to this complex environment. In RoboCupRescue, the online search in the belief state space represents a search in the possible paths that an agent can take. In the tree, the probability to go from one belief state to another depends on the probability that the road used is blocked. One specificity of this problem is that we have to return a path to the simulator, thus the RTBSS algorithm has been modified to return the best branch of the tree instead of only the first action.

Furthermore, the key aspect of our approach is that we consider many variables of the environment to be static during the search in order to minimize the number of states considered. For example, suppose the agent has to evaluate a belief state. If we consider the general problem, the agent would have to iterate over a lot of states, because many states are possible, thus the set returned by the equation 6 would still be very big.

Our idea is to reduce this subset of states even more. To do so, the agent considers that some parts of the environment are static during its search in the tree. In the RoboCupRescue, all variables are considered static except the position of the agent and the variables about the roads. For the other variables, like the position of the other agents and the position of the fires, the agent considers that they keep the last value observed. Consequently, all those fixed

variables are represented in the belief state by a vector containing only zeros except for the last value observed which has a probability of one. Therefore, the function ω (equation 6) only returns a small subset of states.

More precisely, the beliefs are only maintained for the road variables. Those variables are the most important for the agent decisions. In other words, the agent focuses on the more important variables, maintaining beliefs as precisely as possible, and it abstracts the other variables by considering that they are fixed and it relies only on its observations to maintain them.

The fixed variables are not ignored, they are considered during the search, but we are not trying to update them. For example, if a firefighter agent is considered to be on road r_3, it will stay there during the whole search. We know that in practice it moves, but to simplify the search, we consider that it stays at the same position. We update the value of the fixed variables only when the agent perceives a new value. In our model, we consider the observations to be both the direct agent's observations and the information received by messages. We are in a cooperative multiagent system, therefore all agents have complete confidence in the information received from the other agents.

In the RoboCupRescue, we could have tried to estimate the position of all other agents by considering their position, their tasks and their speed, but since they are all independent agents, the estimated probability would have been really approximate. Moreover, the computation time to maintain good approximations would have been too important. Consequently, the price to pay is too important compared to the gain we could have made.

In complex dynamic multiagent environments, it is often better to rely on observations than to try to predict everything. There are just too many things moving in the simulation. Therefore, the agent should focus on the more important parts of the environment. To efficiently take all the unpredicted parts of the environment into consideration, the agent can shorten its loop of observation and action to keep its belief state up-to-date. This can be done because our RTBSS algorithm can find an action very quickly. Consequently, the agent makes frequent observations, thus it does not need a model for the less important parts of the world, because they do not have time to move a lot between observations.

6.3 Dynamic Reward Function

Moreover, we have defined a dynamic reward function that gives a reward for clearing a road that depends on the position of the fires and the other agents. This enables the agent to efficiently compute its estimated rewards based on its current belief state without having to explicitly store all rewards for all possible states.

A policeman agent needs to assign a reward to each road in the city, which are represented as nodes in a graph (see Figure 3). The reward values change in time based on the position of the agents and the fires, therefore the agent needs to recalculate them at each turn. To calculate the reward values, the agent propagates rewards over the graph, starting from the rewarding roads, which are the position of the agents and the fires. For example, if a firefighter agent is on road r_1 then this road would receive a reward of 5, the roads adjacent to r_1 in the graph would receive a reward of 4, the roads adjacent to the roads adjacent to r_1 would receive 3,

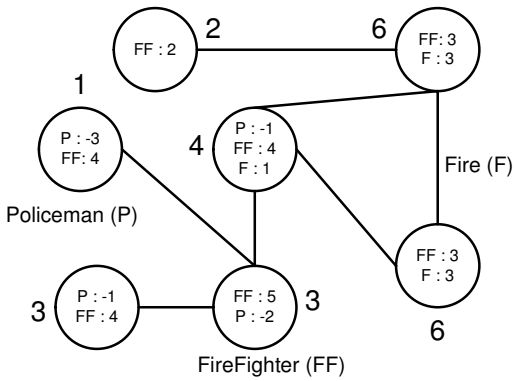

Figure 3: Reward function's graph.

and so on. Also, we add rewards for all roads in a certain perimeter around a fire.

What is interesting with this reward function is that it can be used to coordinate the policeman agents. The coordination is necessary, because we do not want all agents to go to the same road. Therefore, we have to maintain some degree of dispersion among the policeman agents. To do so, the agent propagates negative rewards around the other policeman agents. In other words, the policeman agents repulse each other. With this simple modification of the reward function, we were able to disperse efficiently, thus dynamically coordinate up to fifteen agents acting in a really dynamic environment.

Figure 3 shows an example of a reward graph. The nodes represent the roads and the reward source is identified in each node. The big number over a node is the total reward, which is the sum of all rewards identified in the node. As we can see, roads around the firefighter agent receive positive reward, while roads around the policeman agent receive negative rewards. Therefore, the agent would want to go to roads near the fire and not necessarily go to the firefighter because there is already a policeman agent near it. Consequently, agents are coordinating themselves simply by propagating negative rewards. This is a nice way to coordinate agents in an online multiagent POMDP.

6.4 Results and Discussion

In such a huge problem as RoboCupRescue, it was impossible to compare our approach with other POMDP algorithms. Therefore, we compared our algorithm RTBSS with an heuristic method for the policeman agents.

To demonstrate the efficiency of RTBSS, we have compared it with our last approach for the policemen, which was an intuitive approach in which agents cleared roads according to some priorities. Each agent received a sector for which it was responsible at the beginning of the simulation. Agents cleared roads in this order: roads asked by the other agents, roads around refuges and fires and finally, all the roads in their sector.

The results that we have obtained on 7 different maps are presented in Figure 4. By using the approach we presented in this paper, it improved the average score by 11 points. This difference is very important because in competitions, a few tenths of a point can make a big difference. For example, at the 2004 international competition, our DAMAS-Rescue team missed the first place by 0.4 points. Furthermore, on

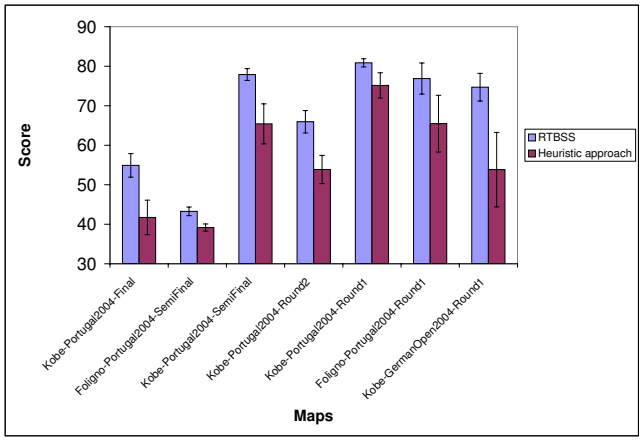

Figure 4: Scores obtained on seven different simulations.

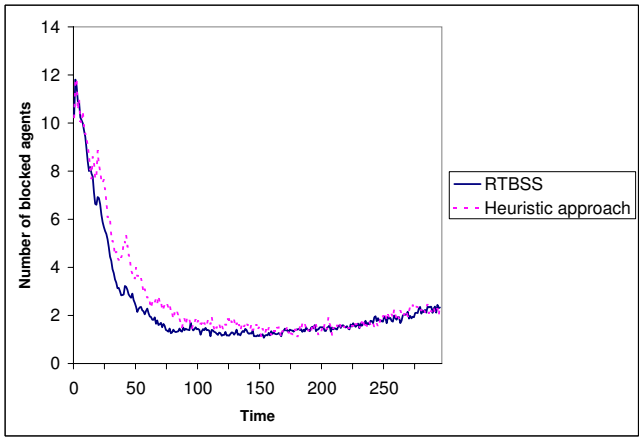

Figure 5: Number of agents blocked.

the graph we show a 95% confidence interval that suggest that our algorithm allows more stable performances.

Figure 5 shows a comparison of the number of agents that are blocked at each cycle. As we mentioned above, one of the goals of the policeman agents is to clear the roads so that other agents can navigate freely in the city. The fewer agents that are blocked, the better the performances are. The results show that our method allows prioritizing the most important roads since on average, there are one or two fewer blocked agents. This means that those agents save civilians instead of waiting for policeman agents. Furthermore, Figure 6 shows the number of roads that are blocked at each cycle in the simulation. We see that RTBSS allows the policeman agents to clear the roads faster. Briefly, with RTBSS agents clear the most important roads faster than with the heuristic approach.

7. RELATED WORK

We compared our approach with 5 others used to find an approximate solution (see Table 1). These approaches are very interesting because they achieve good solution quality. However, those approaches are still limited to relatively small environments because they all proceed offline which

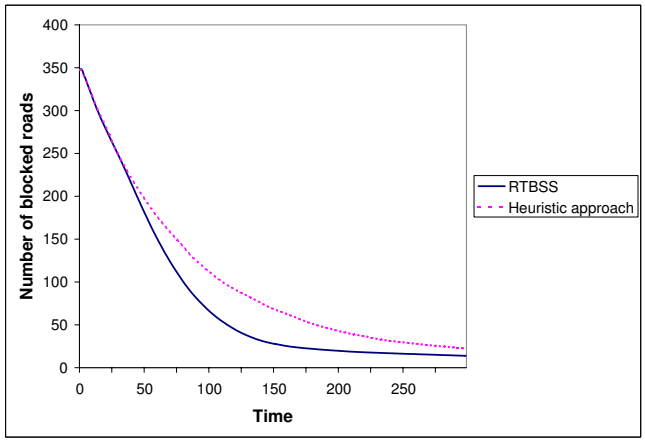

Figure 6: Number of blocked roads.

requires a lot of computation time on big problems.

Other researchers have worked on online approaches. For example, [14] have also used an online exploration of the belief state space, but they were not using any pruning techniques in order to accelerate the algorithm. In addition, [22] developed the BI-POMDP algorithm which expands a tree similar to us, but with a AO* approach to which he adds bounds calculation to choose the order in which the nodes have to be expanded. For his bounds, the underlying MDP has to be solved offline, which is not admissible for us, as stated in the Motivations section. [11] have used an exploration of trajectories in a tree structure to test different policies. Their objective is to choose a policy in a class of policies using a simulator of the POMDP. In our case, we do not consider the availability of a simulator to test different policies. We instead consider that the agent is working directly in the environment.

Another online algorithm is BEL-RTDP which learns some heuristics values for the belief states visited by successive trials in the environment [6]. The main differences of this approach is that it does not search in the belief state tree and it needs offline time to calculate the starting heuristic based on the Q_{MDP} approach. Also, since it learns heuristic values for each belief states visited, it needs to discretize the belief state space to have a finite number of belief states.

Other methods use an heuristic search to solve POMDPs, but in their case, the search is performed in the space of the policies, represented as finite state machines [7, 15]. Thus, they also try to build a policy offline which differs from our approach. Another offline algorithm that uses search to solve MDPs is LAO*, which returns a complete policy represented as a cyclic graph [8].

It is also possible to draw similarities with the work in the probabilistic planning domain [2, 3]. In planning a look-ahead search is often used to explore the belief space until a goal is found. However, in POMDPs the agent is not necessarily looking for a goal but it is rather trying to maximize its expected rewards. In addition, classical planning does not take into account the execution. With POMDPs, after each executed action, the agent perceives a new observation that influences its future. The plan, or policy, produced then needs to take into consideration all possible observations which is not the case in planning.

8. CONCLUSION AND FUTURE WORK

In this paper, we have described an approach that can be used online to act quickly in POMDPs with a huge state space. The main advantage of our method is that it can be applied to problems where offline algorithms would take too much time. The main idea consists of using an online approach in order to avoid computing a full policy offline.

We showed results on two standard POMDP problems. On the smaller problems, RTBSS is not far from the other algorithms, but it is much faster. On the largest problem, RTBSS becomes better because the offline algorithm was not able to converge. This shows a great advantage of our algorithm on larger problems. The larger the environment is, the better RTBSS is compared to offline approaches.

We have also applied our algorithm in a complex multiagent environment, the RoboCupRescue simulation. We showed how we have slightly modified our basic RTBSS algorithm to take the specificity of multiagent systems more into consideration. In addition, we introduced a dynamic reward function, which is really useful to coordinate agents in dynamic environments. Then, we presented results showing the efficiency of RTBSS for the policemen tasks in the RoboCupRescue simulation.

In our future work, we would like to improve our online algorithm by reusing the information computed previously in the simulation. This would allow being able to explore in greater depth without using more computation time.

9. REFERENCES

[1] D. Aberdeen. A (Revised) Survey of Approximate Methods for Solving Partially Observable Markov Decision Processes. Technical report, National ICT Australia, 2003.

[2] B. Bonet and H. Geffner. Planning with Incomplete Information as Heuristic Search in Belief Space. In *Proc. 5th International Conf. on Artificial Intelligence Planning and Scheduling*, pages 52–61, Breckenridge, Colorado, 2000. AAAI Press.

[3] C. Boutilier, T. Dean, and S. Hanks. Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

[4] X. Boyen and D. Koller. Tractable Inference for Complex Stochastic Processes. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 33–42, Madison, 1998. Morgan Kaufmann Publishers.

[5] D. Braziunas and C. Boutilier. Stochastic Local Search for POMDP Controllers. In *The Nineteenth National Conference on Artificial Intelligence (AAAI-04)*, 2004.

[6] H. Geffner and B. Bonet. Solving Large POMDPs Using Real Time Dynamic Programming. In *Working Notes Fall AAAI Symposium on POMDPs*, pages 61–68, 1998.

[7] E. A. Hansen. Solving POMDPs by Searching in Policy Space. In *Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 211–219, Madison, Wisconsin, 1998.

[8] E. A. Hansen and S. Zilberstein. LAO: a Heuristic Search Algorithm that Finds Solutions with Loops. *Artificial Intelligence*, 129(1-2):35–62, 2001.

[9] M. Hauskrecht. Value-Function Approximations for Partially Observable Markov Decision Processes. *Journal of Artificial Intelligence Research*, 13:33–94, 2000.

[10] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. Technical Report CS-96-08, Brown University, 1996.

[11] M. Kearns, Y. Mansour, and A. Y. Ng. Approximate Planning in Large POMDPs via Reusable Trajectories. In S. Solla, T. Leen, and K.-R. Muller, editors, *Advances in Neural Information Processing Systems 12*. MIT Press, 2000.

[12] H. Kitano. RoboCup Rescue: A Grand Challenge for Multi-Agent Systems. In *Proceedings of ICMAS 2000*, Boston, MA, 2000.

[13] M. L. Littman. *Algorithms for Sequential Decision Making*. PhD thesis, Brown University, 1996.

[14] D. McAllester and S. Singh. Approximate Planning for Factored POMDPs using Belief State Simplification. In *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 409–416, San Francisco, CA, 1999. Morgan Kaufmann Publishers.

[15] N. Meuleau, K.-E. Kim, L. P. Kaelbling, and A. R. Cassandra. Solving POMDPs by Searching the Space of Finite Policies. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 417–426, San Francisco, 1999. Morgan Kaufmann Publishers.

[16] C. Papadimitriou and J. N. Tsisiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[17] J. Pineau, G. Gordon, and S. Thrun. Point-Based Value Iteration: An Anytime Algorithm for POMDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 1025–1032, Acapulco, Mexico, 2003.

[18] P. Poupart. *Exploiting Structure to Efficiently Solve Large Scale Partially Observable Markov Decision Processes*. PhD thesis, University of Toronto, 2005.

[19] T. Smith and R. Simmons. Heuristic Search Value Iteration for POMDPs. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence(UAI-04)*, Banff, Canada, 2004. AUAI Press.

[20] E. J. Sondik. *The Optimal Control of Partially Observable Markov Processes*. PhD thesis, Stanford University, 1971.

[21] M. T. J. Spaan and N. Vlassis. A Point-Based POMDP Algorithm for Robot Planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2399–2404, New Orleans, Louisiana, 2004.

[22] R. Washington. BI-POMDP: Bounded, Incremental Partially-Observable Markov-Model Planning. In *Proceedings of the 4th European Conference on Planning*, volume 1348 of *Lecture Notes in Computer Science*, pages 440–451, Toulouse, France, 1997. Springer.