An Online POMDP Algorithm Used by the PoliceForce Agents in the RoboCupRescue Simulation

Sébastien Paquet, Ludovic Tobin, and Brahim Chaib-draa

DAMAS Laboratory, Laval University
{spaquet, tobin, chaib}@damas.ift.ulaval.ca

Abstract. In the RoboCupRescue simulation, the *PoliceForce* agents have to decide which roads to clear to help other agents to navigate in the city. In this article, we present how we have modelled their environment as a POMDP and more importantly we present our new online POMDP algorithm enabling them to make good decisions in real-time during the simulation. Our algorithm is based on a look-ahead search to find the best action to execute at each cycle. We thus avoid the overwhelming complexity of computing a policy for each possible situation. To show the efficiency of our algorithm, we present some results on standard POMDPs and in the RoboCupRescue simulation environment.

1 Introduction

Partially Observable Markov Decision Processes (POMDPs) provide a very general model for sequential decision problems in a partially observable environment. A lot of problems can be modelled with POMDPs, but very few can be solved because of their computational complexity (POMDPs are PSPACE-complete [1]), which motivates the search for approximation methods. Recently, many approximation algorithms have been developed [2, 3, 4, 5, 6] and they all share in common the fact that they try to solve the problem offline. While these algorithms can achieve very good performances, they are still not applicable on large multiagent problems, like the RoboCupRescue simulation.

This paper presents a novel idea for POMDPs that, to our knowledge, have never received a lot of attention. The idea is to use an online approach based on a look-ahead search in the belief state space to find the best action to execute at each cycle. By doing so, we avoid the overwhelming complexity of computing a policy for every possible situation the agent could encounter. We present results showing that it is possible to achieve relatively good performances by using a very short amount of time online. To make our solution even better, we have incorporated a factored representation in order to speed up the computation time and reduce the amount of memory required.

In this article, we first describe the formalism of our RTBSS (Real-Time Belief Space Search) algorithm, followed by some results on standard POMDPs, then we present an adaptation of our method for the RoboCupRescue simulation environment and some results showing its efficiency.

A. Bredenfeld et al. (Eds.): RoboCup 2005, LNAI 4020, pp. 196–207, 2006.

2 POMDP

In this section we briefly describe POMDPs [7, 8] and then we introduce factored POMDPs. 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.

The agent also needs to choose an action to do in function of its current belief state. This action is determined by the policy π, which is a function that maps a belief state to the action the agent should execute in this belief state. To construct a policy, the agent has to evaluate the expected reward of a belief state. To do so, the agent can use the following value function of a belief state for an horizon of t:

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

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

$R(b)$ is the expected reward for the belief state b and 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:

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

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:

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

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

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

$$b = \left(\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.25 \\ 0.05 \\ 0.50 \\ 0 \\ 0.10 \\ 0.10 \end{bmatrix}, \begin{bmatrix} 0.10 \\ 0.30 \\ 0.15 \\ 0.45 \end{bmatrix} \right) \qquad b = \left(\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} P(X_2 = x_1 \wedge X_3 = x_1) \\ P(X_2 = x_1 \wedge X_3 = x_2) \\ \cdots \\ \cdots \\ \cdots \\ P(X_2 = x_6 \wedge X_3 = x_4) \end{bmatrix} \right)$$

Fig. 1. Belief states without dependance (left) and with dependance (right)

2.1 Factored POMDP

The traditional POMDP model is not suited to big environments because it requires explicitly enumerating all the states. However, most environments can be described by a set of different features which allows representing them much more compactly. Let $X = \{X_1, X_2, \ldots, X_M\}$ be the set of M random variables that fully describe a system state. We can then define a state by assigning a value to each variable: $s = \{X_1 = x_1, \ldots, X_M = x_M\}$ or more compactly $s = \{x_i\}_{i=1}^{M}$.

With such a factored representation it is then possible to compactly represent the transition and observation functions as a dynamic Bayesian network [9]. Also, if the state variables can be partitioned into probabilistically independent subsets, then the joint distribution representing the belief state can be compactly represented by the product of the marginal distributions of each subset [4]. Therefore, by maintaining the probabilities on variables instead of states, it is much easier to update the belief state and use it for approximation methods.

Figure 1 shows an example of two belief states. On the left, the variables are all independent, thus there is one vector of probabilities for each variable. On the right, the last two variables are dependent and thus there are only two vectors. Moreover, even if all variables are dependent, it is still possible to factorize the belief state with minimal degradation of the solution's quality. Some methods have been developed to automatically generate groups of variables that offer an interesting compromise between compactness of the representation and the performance of the agent [4, 10]. It is important to mention that the factorization is not necessary for our method, but if it can be used, it can accelerate the calculations, thus helping our algorithm to search deeper.

2.2 Formalization

More formally, to take advantage of the factored representation of the belief state, we define a function $\omega : B \to \mathbb{P}\, S$:

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

This function returns, according to a belief state, all the states the agent could be in. We know that a state is impossible if one of the variables has a probability of zero according to b. Moreover, if the variables are ordered approximately according to their certainty (see Figure 1), this subset of states can be rapidly constructed because each time we encounter a variable with a probability equal

to zero, we can immediately exclude all the corresponding states. The following equation can then be computed much more rapidly than equation 2:

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

In fact, the less uncertainty the agent has, the smaller the subset of possible states is and the faster the computation of equation 7 is compared to equation 2.

Now that we consider only the states that an agent can be in, we would also like to have a function that returns the states that are reachable from a certain belief state. To do so, 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 a that was performed and the observation perceived o and returns all the states that the agent can reach:

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

The probability of making an observation ($P(o|a, b)$) can also be expressed using α and ω:

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

3 Online Decision Making

Instead of computing a policy offline, we adopted an online approach where the agent rather performs a local search at each step in the environment, thus avoiding a lot of computations. The advantage of such a method is that it can be applied to problems with a huge state space.

3.1 Belief State Value Approximation

In section 2, we described how it was possible to exactly compute the value of a belief state (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 $\delta : B \times \mathbb{N} \rightarrow \mathbb{R}$ that looks like equation 1, but that is based on a depth-first search instead of dynamic programming. 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.

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

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

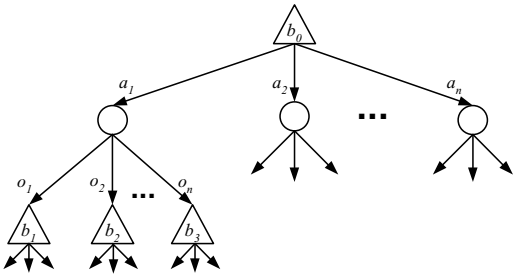

Fig. 2. A search tree

When $d = 0$, we are at the bottom of the search tree. In this situation, we need a way to estimate the value of the current belief state. To do so, we need a utility function $U(b)$, that gives an estimation of the real value of this belief state (if the function $U(b)$ was perfect, their would be no need for a search). 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 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 agent's policy which returns the action the agent should do in a certain belief state is defined as:

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

3.2 RTBSS Algorithm

We now describe our RTBSS (Real-Time Belief Space Search) algorithm 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. 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, the algorithm backtracks and explores another action. If not, the search continues because there is a non-zero probability that the best solution hides somewhere in the sub-tree.

Moreover, the heuristic function used in the PRUNE function must be defined for each problem. Also, to work well it has to always overestimate the true value.

Algorithm 1. The RTBSS algorithm

```
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(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
```

If it does not, it would have the effect of pruning some parts of the tree that might hide the best solution. On the other hand, if the heuristic always overestimates, we are guaranteed 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 actions are sorted at line 13. The purpose of this is to try the actions that are the most promising first because it generates more pruning early in the search.

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 our algorithm is in the worst case of: $O((|A| \times |\Omega|)^D)$. This is if no pruning is possible, thus with a good heuristic, it is possible to do much better. As we can see, the complexity of our algorithm depends on the number of actions and

Table 1. Comparison of our approach

Problem	Reward	Time (s)
Tag (870s,5a,30o)		
Q_{MDP}	-16.75	11.8
RTBSS	-10.56	0.23[1]
PBVI [3]	-9.18	180880
BBSLS [2]	$\backsim$ -8.3	$\backsim$100000
BPI [4]	-6.65	250
HSVI [5]	-6.37	10113
Perséus [6]	-6.17	1670
RockSample[4,4] (257s,9a,2o)		
RTBSS	16.2	0.1[1]
PBVI [5][2]	17.1	$\sim$ 2000
HSVI [5]	18.0	577
RockSample[5,5] (801s,10a,2o)		
RTBSS	18.7	0.1[1]
HSVI [5]	19.0	10208
RockSample[5,7] (3201s,12a,2o)		
RTBSS	22.6	0.1[1]
HSVI [5]	23.1	10263
RockSample[7,8] (12545s,13a,2o)		
RTBSS	20.1	0.2[1]
HSVI [5]	15.1	10266

observations, but not on the number of states. Therefore, even if the environment has a huge state space, our algorithm would still be applicable, if the number of actions and observations are kept small.

4 Experiments on Standard POMDPs

This section presents the results obtained in two problems: *Tag* [3] and *RockSample* [5]. 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. Our algorithm 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 generally not too important.

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 instance of the problem. RTBSS is better because HSVI has not had time to converge. This shows the advantage of our approach on large environments.

[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 [3], but the result on *RockSample* was published in [5].

Another huge advantage of our algorithm is that if the environment changes, we do not need to recompute a policy. Let's suppose that we have the RockSample problem but at each new simulation, the initial position of the rocks changes. With offline algorithms, it would require recomputing a new policy for the new configuration while our algorithm could be applied right away.

5 Experiments on RoboCupRescue

The RoboCupRescue simulation environment consists of a simulation of an earthquake happening in a city [11]. 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.

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?

5.1 RoboCupRescue Viewed as a POMDP

We present how we modelled the problem of 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 of the system can be described by approximately 1500 random variables, depending on the simulation.

- *Roads*: There are approximately 800 roads in a simulation and each road can either be blocked or cleared.
- *Buildings*: There are approximately 700 buildings in a simulation. We consider that a building can 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.

5.2 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. 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.

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 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.

Moreover, we have defined a dynamic reward function that gives a reward for clearing a road based on the positions of the fires and the other agents. This enables the agent to efficiently compute its estimated rewards based on its 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 positions 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 positions 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

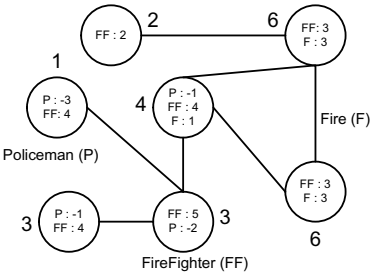

Fig. 3. Reward function's graph

the graph would receive a reward of 4, 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. To do so, the agent propagates negative rewards around the other policeman agents, thus they all 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 rewards, 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 help the firefighter because there is already a policeman agent near it. Consequently, agents are coordinating themselves simply by propagating negative rewards.

5.3 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. We have compared it with our last approach for the policemen in the RoboCupRescue simulation. In this approach agents were clearing roads according to some priorities. Each agent received a sector for which it was responsible at the beginning of the simulation. Afterwards, agents were clearing 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. The approach presented in this paper 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. On the graph, we show a 95% confidence interval that suggest that our algorithm allows more stable performances.

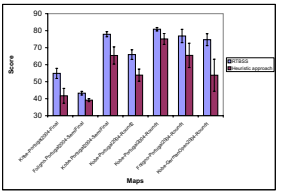

Fig. 4. Scores

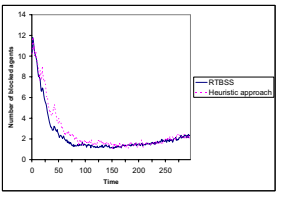

Fig. 5. Agents blocked

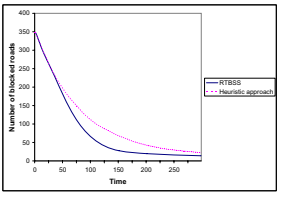

Fig. 6. Roads blocked

Figure 5 shows a comparison of the number of agents that are blocked at each cycle. The results show that our method allows prioritizing the most important roads since on average, there are one or two fewer blocked 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. Another result showing the efficiency of our approach is that it helped us to finish second at the 2004 international competition.

6 Related Work

The first works that come to mind are those approximation methods that aim to solve larger problems. Particularly, we compared our approach with 5 other algorithms [2, 3, 4, 5, 6] used to find an approximate solution. These approaches are very interesting because they achieve good solution quality. In addition, some of these algorithms can bound their distance from the optimal solution. However, they can only be applied to relatively small environments because they all proceed offline and require a lot of computation time on big problems.

For POMDPs, very few researchers have explored the possibilities of online algorithms. [12] 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.

7 Conclusion and Future Work

This paper presents 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. To reinforced our claims, we showed results on two problems considered to be large in the POMDP literature. Those results show that RTBSS becomes better as the environment becomes bigger, compared to state of the art POMDP approximation algorithms.

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, and we presented results showing its efficiency.

In our future work, we would like to improve our online algorithm by reusing the information computed previously in the simulation. We also want to combine offline and online strategies, because it might be possible to precompute some information offline that could be useful online. In short, RTBSS represents a good step towards making POMDP applicable in real life applications.

References

1. Papadimitriou, C., Tsisiklis, J.N.: The complexity of markov decision processes. Mathematics of Operations Research **12** (1987) 441–450
2. Braziunas, D., Boutilier, C.: Stochastic local search for POMDP controllers. In: The Nineteenth National Conference on Artificial Intelligence (AAAI-04). (2004)
3. 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
4. Poupart, P.: Exploiting Structure to Efficiently Solve Large Scale Partially Observable Markov Decision Processes. PhD thesis, University of Toronto (2005) (to appear).
5. 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)
6. 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
7. Aberdeen, D.: A (revised) survey of approximate methods for solving partially observable markov decision processes. Technical report, National ICT Australia (2003)
8. Kaelbling, L.P., Littman, M.L., Cassandra, A.R.: Planning and acting in partially observable stochastic domains. Technical Report CS-96-08, Brown University (1996)
9. Boutilier, C., Poole, D.: Computing optimal policies for partially observable decision processes using compact representations. In: Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96), Portland, Oregon, USA, AAAI Press / The MIT Press (1996) 1168–1175
10. Boyen, X., Koller, D.: Tractable inference for complex stochastic processes. In: In Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence. (1998) 33–42
11. Kitano, H.: Robocup rescue: A grand challenge for multi-agent systems. In: Proceedings ICMAS 2000, Boston (2000)
12. Geffner, H., Bonet, B.: Solving large POMDPs using real time dynamic programming (1998)