Proceedings of the Twenty-Sixth International Florida Artificial Intelligence Research Society Conference

Learning Policies in Partially Observable MDPs with Abstract Actions Using Value Iteration

Hamed Janzadeh and Manfred Huber
The University of Texas at Arlington
701 South Nedderman Drive
Arlington, Texas 76019

Abstract

While the use of abstraction and its benefit in terms of transferring learned information to new tasks has been studied extensively and successfully in MDPs, it has not been studied in the context of Partially Observable MDPs. This paper addresses the problem of transferring skills from previous experiences in POMDP models using high-level actions (options). It shows that the optimal value function remains piecewise-linear and convex when policies are high-level actions, and shows how value iteration algorithms can be modified to support options. The results can be applied to all existing value Iteration algorithms. Experiments show how adding options can speed up the learning process.

Introduction

Today, computers are playing a major role in solving everyday problems and it is no longer possible to accomplish most of our routine tasks without their aid. The way we solve our real-life problems using computers is through first modeling the problems in a mathematical form and second developing computer algorithms to solve the formalized problems. Among the different mathematical models we have to formalize problems, *Markov* processes are very useful for planning or decision making under uncertainty. This makes it important to develop effective methods for mapping problems into these models and for learning efficient solutions.

Partially Observable Markov Decision Processes (POMDPs) provide a broader definition and a better model of the uncertainties in the environment compared to MDPs. As a result, POMDPs are usually better models for formalizing real-world problems and there already exist a number of learning algorithms for these models, including Value Iteration (Smith and Simmons 2012), Policy Gradient (Sutton et al. 2009), Bayesian Learning algorithms (Toussaint, Charlin, and Poupart 2008), etc. However, all of these algorithms are still not effective for problems with more than a few thousand states. On the other hand, real-world problems usually require a huge (if not infinite) number of states if directly modeled by Markov processes.

While developing faster algorithms can be effective for addressing this issue, another approach is to re-evaluate the methods we use to formalize our problems. *Abstraction* and *Transfer Learning* (Taylor and Stone 2009) are interesting topics in AI that take advantage of this concept and help solving more complicated problems using the existing algorithms through systematic down-scaling of the problem size and reusing of the already learned skills.

Unfortunately, abstraction and transfer learning are not well studied for POMDPs. In this paper, we show how transferring knowledge and abstraction is possible in POMDP models through use of high-level actions (options) and we present an algorithm to learn optimal policies for these models created from high-level actions. The learning algorithm is based on the conventional value iteration algorithms for POMDPs. Some variations of these algorithm are among the fastest algorithms that exist for learning in POMDPs.

A new problem is also introduced which is a simplified version of the Robocup Rescue Simulation (Amraii et al. 2004) problem that is very scalable and could be a useful testbed for evaluating abstraction methods in POMDPs.

In the remainder, the next section gives an introduction to the formal definition of POMDPs and the value iteration learning algorithms for these models. Then we explain how value iteration could be modified to learn policies that include high-level actions. In the last section, the evaluation environment and the experimental results are presented.

Related Work

POMDPs

A *Partially Observable Markov Decision Process* (POMDP) is a broader definition of the *Markov Decision Process* (MDP). The difference is that in a POMDP the world's state is not known to the agent; instead, a probabilistic observation corresponding to the state is received from the environment after taking each action. Compared to an MDP, POMDP models are more realistic representations of real-world environments and therefore solving POMDP problems is a step towards addressing more realistic problems.

A finite POMDP is formally defined by the tuple $<\mathcal{S},\mathcal{A},\mathcal{Z},I,T,Z,R,\gamma>$ where:

- $\mathcal{S}$ is a finite set of all the states of the world;
- $\mathcal{A}$ is a finite set of all the actions available to the agent;
- $\mathcal{Z}$ is a finite set of all the observations the agent could receive from the environment;

- $I : D(\mathcal{S})$ is the *initial state probability function* and I_s is the probability of the world starting in state s;
- $T : \mathcal{S} \times \mathcal{A} \to D(\mathcal{S})$ is the *state transition probability function* and $T^{sa}_{s'}$ is the probability of transiting to state s' after taking action a in state s;
- $Z : \mathcal{S} \times \mathcal{A} \to D(\mathcal{Z})$ is the *observation probability function* and Z^{sa}_{z} is the probability of receiving observation z after taking action a and ending up in state s;
- $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the *reward function* and R_{sa} is the reward received after taking action a in state s;
- $\gamma \in [0, 1]$ is the *discount factor*.

Belief States

In an MDP model, the agent can make decisions based on the current state of the world which is known to it. Therefore, the policy can be defined as a mapping from the world states to actions. However, in a POMDP the world state is not known by the agent and it could thus not define policies by mapping states to actions. To address this, one way to define POMDP policies as a mapping from *belief states* to actions (Kaelbling, Littman, and Cassandra 1998).

A belief state b is a probability distribution over the world states. Given a belief state b, we use b_s to represent the probability of state s. The belief state starts being equal to the initial state probability function, I, given by the POMDP model and is updated over time to track the world state's probability using the observations the agent receives. Formally, a belief state b after taking action a and receiving observation z is updated to belief state b', or $\mathcal{T}(b, a, z)$, as defined below:

$$\mathcal{T}_{s'}(b, a, z) = b'_{s'} = \frac{\sum_s b_s T^{sa}_{s'} Z^{s'a}_{z}}{\sum_{s''} \sum_s b_s T^{sa}_{s''} Z^{s''a}_{z}} \qquad (1)$$

Value Iteration

The *optimal value function* for a POMDP model can be defined recursively over the belief state space using the Bellman equations (Kaelbling, Littman, and Cassandra 1998):

$$\begin{aligned} V^*_n(b) &= \max_a \sum_s b_s \Big(R_{sa} + \\ &\quad \gamma \sum_{s'} \sum_z T^{sa}_{s'} Z^{s'a}_{z} V^*_{n-1}(\mathcal{T}(b, a, z)) \Big), \end{aligned} \qquad (2)$$

where, $V^*_n(b)$ is the value of the optimal policy for belief state b in the n^{th} iteration.

A major problem with this representation of the value function is that it is defined over an infinite space of belief states, making it impossible to learn using conventional value iteration algorithms or even to represent the value function. This way, a single iteration of the value iteration algorithm would require an infinite amount of time and exact representation of the value function needs infinite space.

In 1973 (Smallwood and Sondik 1973) showed that the optimal value function given in Equation 2 has properties that make it possible to solve it using value iteration algorithms. In particular, they showed that the optimal value function is *piece-wise linear* and *convex* over the space of belief states and can be represented by a finite set of vectors. Figure 1 shows a piece-wise linear and convex value function for a sample POMDP with two states. Each of the vectors representing a linear function is called an α*-vector* and the value of the optimal policy at each belief state is equal to the maximum value for all of the linear functions at that belief state. This is formally expressed in Equation 3:

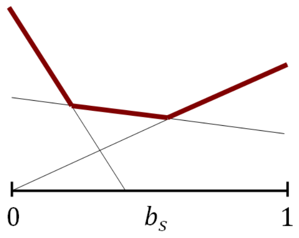

Figure 1: A sample optimal value function for a two state POMDP problem.

$$V^*_n(b) = \max_l \Big[\sum_s b_s \alpha^l_s(n) \Big], \qquad (3)$$

where, $\alpha^l_s(n)$ is the s^{th} element of the l^{th} α-vector at iteration n. The size of each α-vector is equal to the number of states. α-vectors can be calculated recursively using the α-vectors of the previous iteration:

$$\alpha^l_s(n) = R_{sa} + \gamma \sum_{s'} T^{sa}_{s'} Z^{s'a}_{z} \alpha^{l'}_{s'}(n-1) \quad \forall a, z, l'. \qquad (4)$$

One problem with this method of value iteration is that the number of α-vectors grows very fast after each iteration. This could make the algorithm computationally very expensive or even infeasible if it is implemented the exact way. However, there exists significant work aimed at addressing this, resulting in different variations of the value iteration algorithm that are computationally less expensive. Most of these algorithms work by approximating the α-vectors through either sampling or by defining bounds on the value function (Smith and Simmons 2012)(Pineau et al. 2003).

Options

In 1998 (Precup, Sutton, and Singh 1998) introduced abstract actions for *Semi-MDP* models which they call *options*. An option is an already learned policy for performing a subtask that could be transferred and used in solving other problems as an additional action in the MDP model. For example, when a robot learns how to go from one room to another, it could use that as one of its skills or actions when learning to plan for another problem like cleaning the house.

Learning the optimal policy for MDPs when options are added is very similar to the original case. Each option is a regular policy that uses lower-level actions to execute. For each option, after being learned, some statistics including the expected reward and transition probability functions are calculated and preserved. The expected reward function, R_{so}, defines the expected sum of the discounted rewards that could be obtained if executing option o in state s before the option terminates. The state transition function $T^{so}_{s'}$ gives the probability of option o terminating in state s' if executed in state s. These functions look very similar to those of regular actions, and with slight modifications to the algorithms and the model, options can be added to the model's set of actions and used for learning of future problems.

The Markov processes the options could be used in are called Semi-Markov, because options violate the Markov assumption by taking arbitrarily long to execute. This prevents the probability of the next time-step's state to depend only on the action taken and the current state.

Hierarchical Policy for POMDPs

In this section we show how an optimal policy made of high-level actions could be calculated for POMDPs by making some modifications to the value iteration algorithm.

Generalized Belief State Update

One of the main pre-requisites of the value iteration algorithm for POMDPs is to have a belief state, because the value function is defined over the belief state space. As a result, it is very important to show how the belief state is updated if options are available in the POMDP model.

Here, we define a more general version of the belief state update to handle options in the value function. In the conventional form, the belief state is updated once after each action is executed and an observation is received. An option, however, is a high level action and executes a sequence of lower-level actions before terminating. Therefore, in the generalized form we define a belief update over a sequence or trajectory of action-observation pairs we call a *history*.

A history h is a trajectory of action-observation pairs:

$$h = < a_1, z_1, a_2, z_2, ..., a_L, z_L >, \quad (5)$$

where a_t is the action taken and z_t is the observation obtained at time-step t. L is the length of the history. Also, h_k is a prefix of h which contains the first k pairs.

Now, we show how a belief state is updated. $b' = \mathcal{T}(b, h)$ is the updated belief state b after experiencing history h:

$$\mathcal{T}_{s'}(b, h_k) = \frac{\sum_s \mathcal{T}_s(b, h_{k-1}) T^{sa_k}_{s'} Z^{s'a_k}_{z_k}}{\sum_{s''} \sum_s \mathcal{T}_s(b, h_{k-1}) T^{sa_k}_{s''} Z^{s''a_k}_{z_k}}. \quad (6)$$

If we expand this recursive function we see that the belief state function is still linear in the initial belief state. We use $\psi^h_{ss'}$ to simplify the equations in the later sections:

$$\begin{aligned}
\mathcal{T}_s(b, h_1) &= \frac{\sum_{s_0} b_{s_0} T^{s_0 a_1}_{s} Z^{s a_1}_{z_1}}{\sum_{s_0} \sum_{s_1} b_{s_0} T^{s_0 a_1}_{s_1} Z^{s_1 a_1}_{z_1}}, \\
\mathcal{T}_s(b, h_2) &= \frac{\sum_{s_0} \sum_{s_1} b_{s_0} T^{s_0 a_1}_{s_1} Z^{s_1 a_1}_{z_1} T^{s_1 a_2}_{s} Z^{s a_2}_{z_2}}{\sum_{s_0} \sum_{s_1} \sum_{s_2} b_{s_0} T^{s_0 a_1}_{s_1} Z^{s_1 a_1}_{z_1} T^{s_1 a_2}_{s_2} Z^{s_2 a_2}_{z_2}}, \\
&\vdots \\
\mathcal{T}_s(b, h_k) &= \frac{\sum_{s_0, s_1, ... s_{k-1}} b_{s_0} T^{s_0 a_1}_{s_1} Z^{s_1 a_1}_{z_1} ... T^{s_{k-1} a_k}_{s} Z^{s, a_k}_{z_k}}{\sum_{s_0, s_1, ... s_k} b_{s_0} T^{s_0 a_1}_{s_1} Z^{s_1 a_1}_{z_1} ... T^{s_{k-1} a_k}_{s_k} Z^{s_k, a_k}_{z_k}} \\
&= \frac{\sum_{s_0} b_{s_0} \psi^{h_k}_{s_0 s}}{\sum_{s'} \sum_{s_0} b_{s_0} \psi^{h_k}_{s_0 s'}}. \quad (7)
\end{aligned}$$

POMDP Options

Similar to regular policies, a POMDP option can be defined either by a Finite State Controller (FSC) or by a finite set of linear functions represented by vectors (we name β-vectors). In this paper, we will focus mainly on options defined using a set of linear equations, because the higher level policy is defined this way, too. However, it is possible to use the FSC options in the proposed value iteration algorithm, as well.

An option needs a termination method; otherwise, it will never stop execution after being selected. We will use a deterministic termination function by adding a termination action to the option's action set. This way, the option's policy will select termination in those areas of the belief state simplex where termination should happen. When learning the option, the reward function has to be manipulated such that the termination action is selected when the sub-task is done. Similarly, initialization has to ensure that an option is not selected in a belief state outside its initialization set.

In the case of FSC options, the solution for handling initialization and termination is much simpler: there will be additional memory states that define those conditions.

Value Iteration for Abstract Policies

Here we discuss the conditions under which the value function for an optimal POMDP policy using options will remain piecewise linear and convex. Then we will explain how value iteration can be modified to learn with options.

First, we need to re-define the POMDP value function from Equation 2 for the case where options are available. The value function could be re-defined using the generalized form of the belief state update function we just explained:

$$V^*_n(b) = \max_o \Big(R^o_b + \sum_{h \in H} P^{o,b}_h \gamma^L V^*_{n-1}(\mathcal{T}(b, h)) \Big), \quad (8)$$

where, $V^*_n(b)$ is the value of the optimal policy in belief state b in iteration n, h is a history, H is the finite set of possible limited horizon histories, and L is the length of history h.

$P^{o,b}_h = P(h|o, b)$ is the probability of history h happening if option o starts execution in belief state b:

$$\begin{aligned}
P^{o,b}_h &= \big(1 - \nu^o(b)\big) \sum_{s_0} b_{s_0} \\
&\quad \times \pi^o(b, a_1) \big(1 - \eta^o(b)\big) \sum_{s_1} T^{s_0 a_1}_{s_1} Z^{s_1 a_1}_{z_1} \\
&\quad \vdots \\
&\quad \times \pi^o(\mathcal{T}(b, h_k), a_k) \eta^o(\mathcal{T}(b, h_k)) \sum_{s_k} T^{s_{k-1} a_k}_{s_k} Z^{s_k, a_k}_{z_k} \\
&= \pi^o(b, a_1) ... \pi^o(\mathcal{T}(b, h_k), a_k) \big(1 - \nu^o(b)\big) \\
&\quad \big(1 - \eta^o(b)\big) ... \big(1 - \eta^o(\mathcal{T}(b, h_{k-1}))\big) \eta^o(\mathcal{T}(b, h_k)) \\
&\quad \times \sum_{s_0, ..., s_k} \Big[b_{s_0} T^{s_0 a_1}_{s_1} Z^{s_1 a_1}_{z_1} ... T^{s_{k-1} a_k}_{s_k} Z^{s_k, a_k}_{z_k} \Big] \\
&= A^o_{h_k} \times W^b_{h_k} \quad (9)
\end{aligned}$$

where, $\pi^o(b, a)$ is the probability of action a being selected by option o in belief state b:

$$\pi^o(b,a) = \begin{cases} 1 & \text{if } a = arg\max_i \sum_s b_s \beta_s^{o,i}, \\ 0 & \text{otherwise.} \end{cases} \quad (10)$$

$\nu^o(b)$ is the probability of belief state b not being an initial state for option o and $\eta^o(b)$ is the probability of option o terminating in belief state b. So, if *init* represents the virtual initialization action and *term* is the termination action, then $\nu^o(b) = \pi^o(b, init)$ and $\eta^o(b) = \pi^o(b, term)$.

In Equation 9, we have split the history probability into two terms: the *agent effect* $A_h^{o,b}$ and the *model effect* W_h^o as introduced in (Shelton 2001). The Agent Effect is the product of all action, initialization and termination probability terms defined by the policy, while the Model Effect is the product of all initial state, state transition and observation terms defined by the world model. Splitting the equation this way, will help us simplify the math later.

R_b^o in Equation 8 is the expected reward after executing option o in belief state b. Assuming that an option is a regular POMDP policy, an option's utility, or its expected reward function, is piecewise linear in b and can be represented with a set of β-vectors, each representing a linear equation.

$$R_b^o = \max_i \sum_s b_s \beta_s^{o,i} \quad (11)$$

Theorem: The optimal value function $V_n^*(b)$ in Equation 8 is piecewise linear and convex, given the option's initialization and termination functions are deterministic and modeled using the β-vectors:

$$V_n^*(b) = \max_l \Big(\sum_s b_s \alpha_s^l(n) \Big) \quad (12)$$

Proof: First, the claim is true for $n = 0$ which is the end of our finite horizon, because the total reward we could get at the end of time is zero and $V_0^*(b) = 0$ is linear.

Now, using induction we can show $V_t^*(b)$ is piecewise-linear and convex in b for all values of t. Assuming the hypothesis holds for $t = n - 1$, then we prove this applies to $t = n$ as well. By using Equation 12 for $t = n-1$, we have:

$$V_{n-1}^*(\mathcal{T}(b,h)) = \max_l \Big(\sum_s \mathcal{T}_s(b,h) \alpha_s^l(n-1) \Big) \quad (13)$$

Plugging Equations 13 and 11 into Equation 8, we get:

$$V_n^*(b) = \max_o \Big(\Big[\max_i \sum_{s_0} b_{s_0} \beta_{s_0}^{o,i} \Big] + \sum_{h \in H} P_h^{o,b} \gamma^L \Big[\max_l \sum_s \mathcal{T}_s(b,h) \alpha_s^l(n-1) \Big] \Big) \quad (14)$$

Finding the location of a particular belief state inside the belief state simplex, we can select the vector that provides the largest value for the given belief state (Smallwood and Sondik 1973). Calling this vector α^*, we can remove the $\max$ operation of the α-vectors from Equation 14. The same argument applies to the β-vectors of the option:

$$V_n^*(b) = \max_o \Big(\sum_{s_0} b_{s_0} \beta_{s_0}^{o,*} + \sum_{h \in H} P_h^{o,b} \gamma^L \sum_s \mathcal{T}_s(b,h) \alpha_s^*(n-1) \Big) \quad (15)$$

In Equation 15, moving $P_h^{o,b}$ inside the inner sum and multiplying it with $\mathcal{T}_s(b,h)$ will cause the model effect W_h^b from Equation 9 to be cancelled out with the normalizing factor (the denominator) in Equation 7. Applying these changes to Equation 15 will result in:

$$\begin{aligned} V_n^*(b) &= \max_o \Big(\sum_{s_0} b_{s_0} \beta_{s_0}^{o,*} + \sum_{h \in H} \gamma^L \sum_s \Big[A_h^o \sum_{s_0} b_{s_0} \psi_{s_0 s}^{h_k} \Big] \alpha_s^*(n-1) \Big) \\ &= \max_o \Big(\sum_{s_0} b_{s_0} \Big[\beta_{s_0}^{o,*} + \sum_s \alpha_s^*(n-1) \big(\sum_{h \in H} \gamma^L A_h^o \psi_{s_0 s}^{h_k} \big) \Big] \Big) \quad (16) \\ &= \max_o \Big(\sum_{s_0} b_{s_0} \Big[\beta_{s_0}^{o,*} + \sum_s \alpha_s^*(n-1) \hat{\Psi}_{s_0 s}^o \Big] \Big), \quad (17) \end{aligned}$$

where, $\hat{\Psi}_{s_0 s}^o = \sum_{h \in H} \gamma^L A_h^o \psi_{s_0 s}^{h_k}$.

In Equation 16, A_h^o is a product of option o's initialization, termination and action probability functions. The action probability $\pi^o(b,a)$ takes on a value of either 0 or 1 for each belief state point b depending on the result of the *argmax* operation in Equation 10 and this value is constant over the whole belief state area wherein the corresponding β-vector has a maximal value. This is because the policy function is piecewise linear. As a result, the action probability terms in A_h^o are constants. The $\nu^o(b)$ and $\eta^o(b)$ functions are constants, too, because they are also defined using β-vectors[1]. Obviously, $\beta^{o,*}$, α^* and ψ^h are independent of b and are constants, too. As a result, all of the terms inside the brackets in Equation 16 are constants and thus the equation is piecewise linear and convex in b, completing the proof. □

In practice we do not need to go through all possible histories to calculate the value of $\Psi_{ss'}^o$ as explained in Equation 17. Instead, we could use a sample set of histories to approximate the expected value for this parameter and we could pre-compute that for each option beforehand.

Another method to expedite the calculations is to approximate the value function by running the value iteration algorithm using a finite set of sampled belief points that properly cover the belief simplex (Pineau et al. 2003). We could also go one step further and instead of sampling the belief points that cover the entire simplex, sample those that are more likely to be reached from the current belief state (Smith and Simmons 2012). This will enable us to approximate the value function more accurately for the current belief state.

Evaluation

The Ambulance Problem

The Robocup Rescue Simulation (Amraii et al. 2004) is one of the well-known AI test-beds that provides a comprehensive environment for research on challenging problems such

[1]Removing this restriction and using arbitrary functions of the belief state for initialization and termination will prevent the value function from remaining piecewise linear and convex.

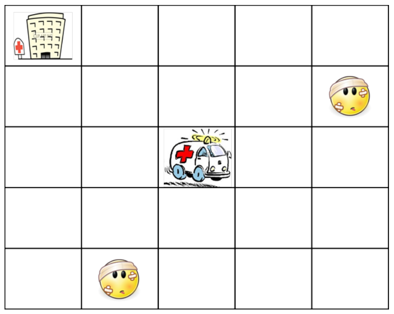

Figure 2: A sample ambulance problem with two civilians.

as decision making under uncertainty, multi-agent planning, realtime learning, etc. The scalability of this environment is unique among most existing official AI problems. One can easily create different subsets of this problem from a simple single-agent environment with few states to a complex multi-agent environment with an infinite number of states.

In our research we also needed a problem that could easily be scaled from a simple case used for benchmarking the algorithms in absence of abstraction to more complex cases that could better show the advantage of the abstraction methods. Most of the existing problems however are designed to be simple and solvable using the existing algorithms. We therefore created a simplified version of the Robocup Rescue Simulation problem which is scalable from a very simple version to complex versions for the future algorithms.

In our *ambulance problem*, there exists a rectangular grid world on which there are an ambulance agent, a set of civilians and one of the grid cells defined as the refuge. The civilians have experienced a disaster and need to be carried to the refuge for protection and cure. The ambulance agent should find the civilians, load them, carry them to the refuge one by one and unload them there. The ambulance does not know its own location and the civilians'. It can see the civilians if they are in the same cell and can hear their voice probabilistically based on their distance. The ambulance also observes when it hits a wall which is considered to be the edges of the grid-world. The agent has actions to move one step *Up*, *Down*, *Right* or *Left* on the grid world and also can *Load* or *Un-load* a civilian. A civilian can be loaded if it is in the same cell as the ambulance, and can be un-loaded everywhere. There is a reward of +1000 to un-load a civilian at the refuge and a -1 for all other actions. A civilian can not be loaded from or escape from the refuge.

We have defined two very simple options for this environment to run our experiments. The *Search* option finds a cell that contains a civilian and then terminates. The *Carry* option moves until it reaches the refuge cell and terminates. It is obvious that the optimal policy is to repeat the sequence of Search, Load, Carry and Unload actions. However, it is not that simple for the learning algorithm because it needs to evaluate all sequences of actions, including primitive actions and options and all different possible trajectories resulting from executing each option. In these experiments we modeled the options as FSCs with three memory states each.

Implementation

Using the value function we derived in the previous section, it is easy to adopt any of the existing POMDP value iteration algorithms to support learning with the high-level actions. All variations of POMDP value iteration use different methods to address the problem of increasing complexity due to the fast growing number of α-vectors. We do not address this problem here and have based our implementation on the Point-based value iteration methods with the updated *backup function* as explained in Algorithm 1. Please refer to the HSVI paper (Smith and Simmons 2012) for more details.

Algorithm 1 ρ = backup(Γ, b)

```
  $\beta^{o,*} \leftarrow arg\max_{\beta^o}(\beta^o.b)$
  $\rho^o_s \leftarrow \beta^{o,*}_s + \max_{\alpha \in \Gamma}(\sum_{s'} \alpha_{s'} \hat{\Psi}^o_{ss'})$
  $\rho \leftarrow arg\max_{\rho^o}(\rho^o.b)$
```

Experiments

We used the ambulance problem to run experiments and evaluate the learning algorithm.

Two different scales of the ambulance problem are used. In the first there is a 3x3 grid world, one civilian and 162 states. The second one has a grid world of size 4x4 with one civilian and 512 states. The parameters defining the state are the ambulance location, the civilians' location and the load status. There is one *Found-civilian* and four *Hit-wall* (for different sides) observations. We did not use the *Heared-civilian* observation here. There are also 6 primitive actions: *Up*, *Down*, *Right*, *Left*, *Load* and *Un-load*. The reward to rescue a civilian is 1000 and the discount factor is 0.95.

Figure 3 depicts the computation times of the algorithm for the two problems, each in three different settings. In the first case (circles), all of the primitive actions are used and there are no high-level actions available. This is the normal POMDP value iteration algorithm. In the second case (rectangles), the Search and Carry options are added to the list of available actions. All of the primitive actions are also preserved. This will increase the number of actions to 8. In the last setting (triangles), we add the two options and removed the four primitive move actions (Up, Down, Left and Right). The picture on the top shows the results for the 3x3 problem while the bottom is the result from the 4x4 problem.

The results show that adding options improves the learning speed even though it makes the model more complex by increasing the number of actions. It also shows that as the number of states is increasing, the effectiveness of the high-level actions in speeding up the learning algorithm also increases. In the ambulance-3x3 problem, the utility of the optimal policy reaches 90% of the maximum value using options two times faster than in the case of primitive actions only. This is almost three times faster in the ambulance-4x4 problem which has almost three times as many states. The reason is that options propagate more information about future rewards in each iteration and will let the value function be updated faster (Precup, Sutton, and Singh 1998).

It is obvious form the experiments that after new, high-level skills are added, the learning speed can be increased if

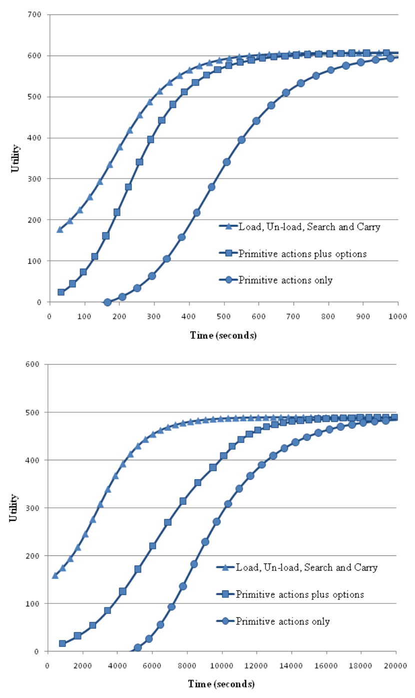

Figure 3: Computation times for learning an ambulance-3x3 problem (top) and an ambulance-4x4 problem (bottom).

less useful primitive actions are removed. We do not address the problem of which actions to remove in this paper; however, this has been studied for MDP models (Asadi 2006).

We also do not explain in this paper how options should be extracted from previous experiments. One way to achieve this is to find the useful policies that might not directly perform a complete task but are repeatedly used as a subset of other policies (Asadi and Huber 2007).

We have used the Cassandra POMDP learning tools (Cassandra 1999) to parse and load the POMDP files for benchmarking with existing value iteration algorithms and also as a base to add our implementation. All experiments were run on a PC with an Intel Pentium 4 CPU (3.40GHz) running a Redhat Enterprise 5 GNU/Linux operating system.

Conclusion

In this paper we provided a learning algorithm based on value iteration for POMDP models that enables the use of high-level actions, or options. Using options makes it possible to transfer knowledge gained from past experiences to solve other problems. We show that the optimal value function remains piece-wise linear and convex after options are added and it is possible to benefit from the existing value iteration algorithms to solve POMDPs. We have also introduced a new problem for benchmarking abstraction in POMDPs that we called the ambulance problem. Our experiments show that adding options makes the learning algorithm faster and more effective when the number of states is increased. In this work we are not addressing how sub-goals could be defined for learning options and how to reduce the complexity by removing less useful primitive actions from the model. The presented algorithm also relies on having the model's distribution functions (i.e. state transition and observation functions) which is a requirement for all other value iteration algorithms; however, in another line of research we are working on a different learning algorithm that enables learning model-less POMDP problems using options.

References

Amraii, S.; Behsaz, B.; Izadi, M.; Janzadeh, H.; Molazem, F.; Rahimi, A.; Ghinani, M.; and Vosoughpour, H. 2004. Sos 2004: An attempt towards a multi-agent rescue team. In *Proceedings of the 8th RoboCup International Symposium*.

Asadi, M., and Huber, M. 2007. Effective control knowledge transfer through learning skill and representation hierarchies. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, 2054–2059.

Asadi, M. 2006. *Learning State and Action Space Hierarchies for Reinforcement Learning Using Action-Dependent Partitioning*. Ph.D. Dissertation, University of Texas at Arlington.

Cassandra, A. 1999. Tonys pomdp page. *website http://www. cs. brown. edu/research/ai/pomdp*.

Kaelbling, L.; Littman, M.; and Cassandra, A. 1998. Planning and acting in partially observable stochastic domains. *Artificial intelligence* 101(1):99–134.

Pineau, J.; Gordon, G.; Thrun, S.; et al. 2003. Point-based value iteration: An anytime algorithm for pomdps. In *International joint conference on artificial intelligence*, volume 18, 1025–1032. LAWRENCE ERLBAUM ASSOCIATES LTD.

Precup, D.; Sutton, R.; and Singh, S. 1998. Theoretical results on reinforcement learning with temporally abstract options. *Machine Learning: ECML-98* 382–393.

Shelton, C. 2001. Policy improvement for pomdps using normalized importance sampling. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, 496–503. Morgan Kaufmann Publishers Inc.

Smallwood, R., and Sondik, E. 1973. The optimal control of partially observable markov processes over a finite horizon. *Operations Research* 21(5):1071–1088.

Smith, T., and Simmons, R. 2012. Point-based pomdp algorithms: Improved analysis and implementation. *arXiv preprint arXiv:1207.1412*.

Sutton, R.; Maei, H.; Precup, D.; Bhatnagar, S.; Silver, D.; Szepesvári, C.; and Wiewiora, E. 2009. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th Annual International Conference on Machine Learning*, 993–1000. ACM.

Taylor, M., and Stone, P. 2009. Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research* 10:1633–1685.

Toussaint, M.; Charlin, L.; and Poupart, P. 2008. Hierarchical pomdp controller optimization by likelihood maximization. *Uncertainty in AI (UAI)*.