A Hierarchical Approach to POMDP Planning and Execution

Joelle Pineau JPINEAU@CS.CMU.EDU
Nicholas Roy NICKR@RI.CMU.EDU
Sebastian Thrun THRUN@CS.CMU.EDU
Robotics Institute, School of Computer Science, 5000 Forbes Ave, Pittsburgh, PA 15213 USA

Abstract

This paper presents a hierarchical approach to POMDPs which takes advantage of structure in the problem domain to find modular policies for complex tasks. We use a decomposition based on partitioning the action space into specialized groups of related actions. We illustrate the appropriateness of the approach by providing empirical results for three contrasting domains.

1. Introduction

The vast majority of AI planning has focused on situations where the state of the environment is fully observable (Russell & Norvig, 1995). In many real-world applications, however, this is far from true. Partially Observable Markov Decision Processes (POMDPs) (Sondik, 1971) provide a general planning and decision-making framework for acting optimally in partially observable domains, and as such have gained much attention (AAAI, 1998). However the computational cost of finding an optimal policy for the agent significantly limits the use of this approach, thus preventing the successful application of POMDPs to more complex problems.

Many real-world domains have structure that can be exploited to find good policies for complex problems. The idea of leveraging structure to address large problems has been explored in Markov Decision Processes (MDPs) to solve complex problems (Singh, 1992; Dayan & Hinton, 1993; Dietterich, 2000). Unfortunately, none of these solutions is directly applicable to POMDPs, since they all assume that the state of the environment is observable; moreover, transitioning between different sub-modules is conditioned on the state of the environment.

The use of structure in POMDPs is more recent, and preliminary attempts (Castanon, 1997; Wiering & Schmidhuber, 1997) typically make strict assumptions about prior knowledge of low-level tasks and ordering, which are substantially restrictive. More recently, memory-based approaches to hierarchical POMDPs have been proposed (Hernandez-Gardiol & Mahadevan, 2001), however the amount of training data required for these exceeds what is available in many problems, especially for the dialogue management domains we are most interested in.

The single idea underlying our approach is to decompose the domain based on *actions*. In many task domains, the space of actions naturally decomposes into a hierarchy of actions that characterizes the applicability of groups of actions in different situations. Consider, for example, a mobile robotic assistant. The actions involved with navigation (move, turn, stop, ...) are fundamentally different from actions concerned with people interaction (speak, honk horn, display, ...). Our approach differs from others that adopt a state-based decomposition of problems.

2. Review of POMDPs

This section establishes the basic terminology used throughout the paper, by providing a brief overview of the essential concepts in POMDPs (see (Kaelbling et al., 1998) for a detailed discussion.)

A POMDP consists of a set of states $S = \{s_1, \dots, s_n\}$, a set of actions $A = \{a_1, \dots, a_m\}$ that the agent can execute, and a set of observations $O = \{o_1, \dots, o_k\}$ that can be perceived by the agent. The dynamics of the model are described by the state transition probability distribution $p(s'|a,s)$[1], the observation probability distribution $p(o|s,a)$[2], and the reward function $R : S \times A \longrightarrow \Re$, which maps states and actions into numerical rewards.

[1]The probability that the state at time $t+1$ is s', assuming that the state at time t is s and the agent executed action a.

[2]The probability that the agent observes o when the world is in state s after executing a.

At any given point in time, the system is assumed to be in some state s_t. In general, it is not possible to determine the current state with complete certainty. Instead, a *belief*[3] distribution is maintained to succinctly represent the history of the agent' interaction (both applied and perceived) with the domain:

$$b_t = Pr(s_t | o_t, a_t, o_{t-1}, a_{t-1}, ..., o_0, a_0) \quad (1)$$

There exist two interesting problems in POMDPs: 1) state tracking, and 2) policy optimization.

State Tracking. To operate in its domain and apply a belief-conditioned policy, an agent must constantly update its belief vector:

$$b'(s') = Pr(s'|o, a, b) = \frac{O(s', a, o) \sum_{s \in S} T(s, a, s') b(s)}{Pr(o|a, b)} \quad (2)$$

where the denominator is simply a normalizing factor. For most domains, this problem is trivial compared to that of computing a useful action selection policy.

Computing a Policy. The goal of POMDP problem solving is to select actions so as to maximize reward collection. The set of selected actions is commonly referred to as the policy. The policy is a function of the belief state b:

$$\pi : B \longrightarrow A \quad (3)$$

It can be computed using value iteration (Sondik, 1971), which assigns a value $V(b)$ to each combination of belief state b:

$$V : B \longrightarrow \Re \quad (4)$$

After convergence, the value is the sum of all (possibly discounted) future payoffs R the agent expects to receive up to time T, if the current belief is b. The literature provides a collection of algorithms for computing the exact value function—and thereby the optimal policy—for finite horizon POMDPs (Cassandra et al., 1997; Kaelbling et al., 1998). However exact algorithms are bounded by a double exponential computational growth in the planning horizon, and in practice can be exponential. This points to the need for more efficient algorithms.

3. Hierarchical POMDPs

The fundamental idea behind our approach is the decomposition of a model-based POMDP problem based on an action hierarchy. Assuming a given POMDP problem, the model designer hierarchically partitions its original action set such that it spans a collection of hierarchically-related smaller POMDPs, which we refer to as *subtasks* (we use notation $\mathbf{P}_i$ for a given POMDP subtask). Each action is assigned to one or more subtasks, where each subtask independently learns a policy over its subset of actions using existing POMDP solving algorithms. High-level subtasks generally learn policies over the selection of lower-level subtasks; and low-level subtasks are responsible for the selection of primitive actions.

[3]The probability that, at time t, the agent is in state s_t, given the history $\{o_t, a_t, o_{t-1}, a_{t-1}, ..., o_0, a_0\}$.

3.1 Action Hierarchy

The defining element of our approach is the action hierarchy. Figure 1 illustrates the basic concept of an action hierarchy. Formally, an action hierarchy is a tree, where each leaf is labeled by an action $a \in A$. Each action $a \in A$ (henceforth called *primitive actions*) must be attached to at least one leaf. The internal leaves are called *abstract actions* (we use a bar, as in $\bar{a}_i$, to indicate that an action is abstract.) Each $\bar{a}_i$ is in fact an abstraction of the actions in the nodes directly below it in the hierarchy (e.g. $\bar{a_2}$ is an abstraction of a_1, a_2, a_3.)

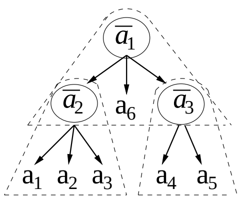

Figure 1. General Form Action Hierarchy

3.2 Task Decomposition

A key step towards hierarchical problem solving is to translate the action hierarchy into a collection of POMDPs that individually are smaller than the original POMDP, yet collectively define a complete policy.

In our approach, each internal node in the action hierarchy together with its immediate children defines a subtask (shown as a triangle in Figure 1.) Each subtask $\mathbf{P}_i$ constitutes a separate POMDP, defined over the full state space S and observation space O, but where its set of applicable actions is limited to its immediate children in the action hierarchy. Policy optimization for that subtask is limited to this action subset. Figure 1 shows a problem that has been divided into three subtasks: $\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3$, with respective action sets: $\mathbf{P}_1 : \{\bar{a_2}, a_6, \bar{a_3}\}$, $\mathbf{P}_2 : \{a_1, a_2, a_3\}$, $\mathbf{P}_3 : \{a_4, a_5\}$.

3.3 Computing Local Policies

Our approach independently optimizes a local action policy[4] for each subtask. We note that only the subtasks with exclusively primitive actions (e.g. $\mathbf{P}_2$, $\mathbf{P}_3$) contain well-defined POMDPs [5], whereas subtasks containing abstract actions are so far ill-defined since the original (flat) POMDP does not provide meaningful parameters conditioned on these abstract actions. Our hierarchical approach recursively makes use of the policies of lower-level subtasks to parameterize the abstract actions, and therefore we proceed in a bottom-up manner to find a local policy for each subtask, from the leaves of the hierarchy, to the root.

Subtasks with only primitive actions are solved first, using any of the existing algorithms (we currently use the algorithm described in (Cassandra et al., 1997)).

To solve subtasks with abstract actions (e.g. $\mathbf{P}_1$), we need to infer model parameters $p(s'|s,\bar{a})$, $p(o|s,\bar{a})$ and $R(s,\bar{a})$ for all abstract actions. Let $\bar{a}_i$ be such an abstract action. Since we are traversing the hierarchy in a bottom-up fashion, we already have calculated a POMDP solution for the subtask spanned by $\bar{a}_i$ (i.e. $\mathbf{P}_i$). Let π_{P_i} be the policy calculated for subtask $\mathbf{P}_i$. Then we define:

$$p(s'|s,\bar{a}_i) := p(s'|s,\pi_{P_i}(s)) \tag{5}$$

$$p(o|s,\bar{a}_i) := p(o|s,\pi_{P_i}(s)) \tag{6}$$

$$R(s,\bar{a}_i) := R(s,\pi_{P_i}(s)) \tag{7}$$

where all the right-hand side terms are defined in the original model, and all the left-hand side terms are those needed to model the abstract action. In other words, we model $\bar{a}_i$ on a state-by-state basis, using the action chosen by the policy of subtask $\mathbf{P}_i$. These definitions lead to a fully parameterized subtask, which can then be solved using any POMDP algorithm (we use incremental pruning). Clearly, this definition of parameters constitutes an approximation. Consider for example subtask $\mathbf{P}_1$, which assumes parameters inferred for abstract actions $\bar{a}_2$, without having access to the full parameterization of actions $\{a_1, a_2, a_3\}$.

One important assumption of this approach is that each subtask contains some local reward information, without which local policies cannot be meaningfully optimized. This is inconsistent with some single-goal problems where partial progress is not rewarded, however the variety of problems presented in the experimental sections suggests that many complex POMDP problems meet this assumption.

[4]A local action policy is a policy which is defined over a given action subset.

[5]A well-defined POMDP is one for which all parameters (e.g. $T(s'|s,a), O(o|s,a), R(s,a)$) are defined

3.4 Calculating the Policy

We are left with the task of constructing a global policy using the set of local policies produced for the subtasks. We first notice that all policies in the hierarchy are defined over the entire belief space, and assume that no abstraction is applied during belief tracking. Thus the agent is in possession of the full belief space.

In practice, the global policy is generated only at execution time. To generate an action, the agent traverses the tree from the top to a leaf. At each level, the agent queries the local policy based on the current belief, and the action proposed by the policy is either primitive or abstract. If the action is a primitive action, it is directly executed by the agent. If the action is an abstract action, the agent queries the policy of the subtask spanned by this action. It is trivial to show that this recursive algorithm always generates a primitive action.

The recursive action selection (and hierarchy traversal) is repeated at each time step. This differs from many hierarchical MDP algorithms where an agent 'remains' in a subtask until a so-called terminal state is reached. The difference is a consequence of the partial state observability in POMDPs, which suggests that we cannot guarantee detectability of terminal states.

3.5 State and Observation Abstractions

In general, the number of linear pieces representing an exact POMDP value function is recursively given by: $|\Gamma_t| = O(|A||\Gamma_{t-1}|^{|O|})$ (with $\Gamma_0 = |A|$), which can be enumerated in time: $O(|S|^2|A||\Gamma_{t-1}|^{|O|})$. The hierarchical algorithm, as described so far, reduces the computational complexity of computing POMDP policies, specifically for large planning horizons. Each subtask possesses a reduced action set, which factors in as an exponential factor in the overall complexity. These savings are partially offset by the fact that we now have to compute many policies, not just one.

Fortunately in many applications (including the ones discussed below), there is an opportunity to further reduce computational costs. We consider domains where the difference between certain states is only relevant if a specific action is available (e.g. robot location may be irrelevant for subtasks that do not involve navigation.) In this case, some state features may be safely ignore within certain subtasks (e.g. robot location in dialogue subtasks). Consequently the state set can be reduced to include only relevant state features, and related observations [6]. This is done on a subtask basis, where each can ignore those state features that

[6]Currently done by hand, but soon to be automatic.

are irrelevant to its small action subset. Subtasks can therefore be defined over smaller state and observation spaces without influencing the policy optimization. The resulting computational savings can be tremendous (several orders of magnitude).

4. Experimental Evaluation

To demonstrate our approach in practice, we evaluated our algorithm on three different domains. We generated policies for all problems using three approaches: a conventional POMDP algorithm, our hierarchical POMDP algorithm (referred to as H-POMDP) and an MDP-solution, which solves the problem as an MDP and during execution uses the most likely state heuristic to map belief states to states. Policy computations were performed using the incremental pruning algorithm for POMDPs, and value iteration for MDPs (all computations were performed on a 400MHz Pentium II). All tasks were evaluated using 5000 runs to show performance over time.[7]

The first task considered is the parts manufacturing problem introduced in Sondik's thesis (Sondik, 1971). We selected this problem specifically because it was not constructed to exhibit structure and can therefore illustrate the generality of the approach. Furthermore, we considered seven different hierarchical decompositions of this problem, to better study whether the algorithm is highly susceptible to a good design of the action hierarchy. Figure 2a shows one of the action hierarchy considered for this domain.

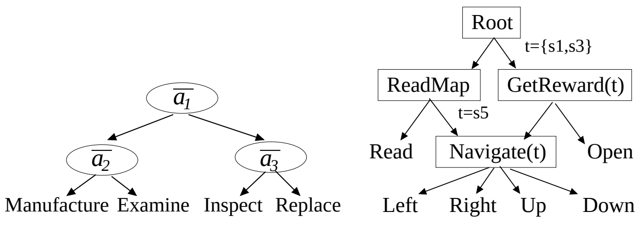

Figure 2. Action hierarchies for (a) Manufacturing task and (b) Taxi navigation task

Figure 3 illustrates the average reward for each of the seven decompositions (in decreasing order of performance). There is a clear grouping, where the first four decompositions yield near-optimal policies, whereas the last three are much weaker, though still as good as the MDP heuristic[8]. We currently have no intuitive explanation for the performance difference, however this is the object of ongoing work.

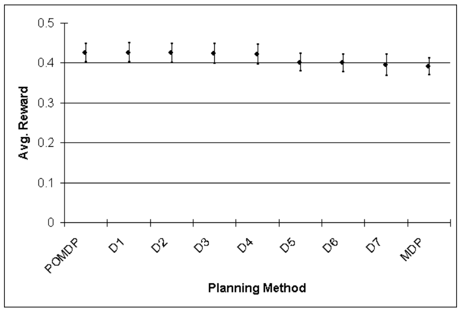

Figure 3. Manufacturing Task Results

The second task is a modified version of Dietterich's taxi task (Dietterich, 2000) with noisy perception. The problem is simple enough to be solvable using conventional POMDP techniques. Figure 2b shows our action hierarchy for this domain.

The third task is a more challenging one. It arises from a robot-interface domain where a robot has to perform diverse tasks involving motion and dialogue exchanges, which can exhibit significant uncertainty, in large part due to poor speech recognition. POMDPs are currently our best solution for this high-level robot control problem. Figure 4 shows our action hierarchy, which decomposes the action space along the natural divisions of various conversational goals.

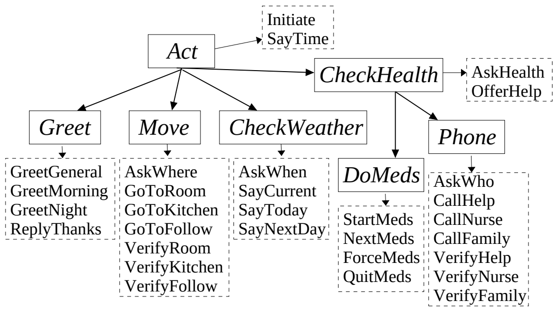

Figure 4. Dialogue Problem Action Hierarchy

Table 1 presents results for the two more complex problems, showing policy computation times, and average reward per action. For the taxi task, performance with the hierarchical POMDP (H-POMDP) approaches that of the conventional POMDP (goal is always reached, but occasionally requires an extra action), and clearly exceeds that of the greedy MDP heuristic, which is unable to take advantage of uncertainty-reducing actions. Results for the robot problem show that even though the exact POMDP solver is unable to find a policy (and had only completed one iteration after 24 hrs), our hierarchical approach (H-POMDP) was able to obtain a policy in

[7]The results reported were obtained using a simulated user due to the large number of experiments necessary to gain significance. Experiments are currently underway to verify the performance of the robot policy with real users.

[8]There is no theoretical guarantee that an H-POMDP policy will necessarily be better than an MDP policy

reasonable time. Execution performance with the H-POMDP policy was superior to that obtained with the MDP heuristic, both in terms of average reward per action and goal success rate. The performance difference between the MDP and H-POMDP policies is smaller for this domain since at many time steps the state is fully observable.

Problem Solution	CPU time (secs)	Average reward	% Goal success
Taxi Problem:	$\|S\|$=11, $\|A\|$=6, $\|O\|$=6		
MDP	0.000654	0.0	50
H-POMDP	2.84	12.2	100
POMDP	1119.93	12.5	100
Robot Problem:	$\|S\|$=20, $\|A\|$=30, $\|O\|$=27		
MDP	6.46	53.33	80.0
H-POMDP	77.99	64.43	93.2
POMDP	24hrs+		

Table 1. Performance Results

Table 2 provides a sample interaction in the robot domain, using the H-POMDP policy. It shows how the policy generated using our approach is able to make use of additional information-gathering actions, thereby allowing the system to recover from speech recognition errors (e.g. the third observation ("go_to") is erroneous: the speech recognition extracted "go to big" from the user's "go to room", and our H-POMDP decides that the cost of accidentally moving to the wrong location outweighs the cost of asking for clarification.) The MDP-heuristic policy did not exhibit similar adaptability to poor recognition performance.

Actor	Speech	Observation	Action
Person	"Hello"	hello	
Robot	"Can I help you?"		Initiate
Person	"What time is it?"	time_request	
Robot	"It's now 11:41."		SayTime
Person	"Go to big."	go_to	
Robot	"Where should I go?"		AskWhere
Person	"Go to the."	go_to_room	
Robot	"Should I go to the room?"		VerifyRoom
Person	"Yes to the room."	go_to_room	
Robot	"You want me to go to the room?"		VerifyRoom
Person	"Yes."	yes	
Robot	*Robot goes to room*		GoToRoom

Table 2. An example dialogue from the interface domain.

It is worth mentioning that the domains examined exhibit structure differently. In the first case, there is no structure to speak of. In the second case, a final goal can be satisfied through a sequence of intermediate subgoals. In the third case, the dialogue manager can satisfy alternate goals within a unified domain. Thus, our experiments address different problem setups always using the same algorithm, thereby shedding light onto our approach under different circumstances.

5. Conclusion

We presented a hierarchical POMDP algorithm that can be used to optimize policies for complex POMDP. A bottom-up algorithm was introduced that computes a sequence of POMDP policies, one for each task in the hierarchy. At run-time, the resulting hierarchy of policies is traversed from the top to the bottom, until a primitive action is found. Mild computational savings are achieved through reduced action space. However, in many tasks the action hierarchy gives rise to state and observation abstractions, which can drastically reduce the computational complexity.

Experimental results obtained for two different tasks illustrate reduction in computational complexity of several orders of magnitude with minimal performance loss, when compared to the flat, computationally hard POMDP model. These experiments suggest that our hierarchical approach provides a viable approach for solving complex POMDPs that are otherwise intractable — assuming that the domain possesses structure that can be expressed via an action hierarchy.

References

AAAI (1998). *AAAI Symposium on POMDPs.* www.cs.duke.edu/mlittman/talks/pomdp-symposium.html.

Cassandra, A., Littman, M. L., & Zhang, N. L. (1997). Incremental pruning: A simple, fast, exact method for partially observable markov decision processes. *UAI.*

Castanon, D. (1997). Approximate dynamic programming for sensor management. *Conf. Decision and Control.*

Dayan, P., & Hinton, G. (1993). Feudal reinforcement learning. *NIPS 5.*

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, *13*, 227–303.

Hernandez-Gardiol, N., & Mahadevan, S. (2001). Hierarchical memory-bsed reinforcement learning. *NIPS 13.*

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, *101*, 99–134.

Russell, S., & Norvig, P. (1995). *Artificial intelligence: A modern approach.* Prentice Hall.

Singh, S. (1992). Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, *8*, 323–339.

Sondik, E. (1971). *The optimal control of partially observable markov processes.* Doctoral dissertation, Stanford.

Wiering, M., & Schmidhuber, J. (1997). HQ-learning. *Adaptive Behavior*, *6*.