Synthesis of Hierarchical Finite-State Controllers for POMDPs

Eric A. Hansen and Rong Zhou
Department of Computer Science and Engineering
Mississippi State University, Mississippi State, MS 39762
{hansen,rzhou}@cs.msstate.edu

Abstract

We develop a hierarchical approach to planning for partially observable Markov decision processes (POMDPs) in which a policy is represented as a hierarchical finite-state controller. To provide a foundation for this approach, we discuss some extensions of the POMDP framework that allow us to formalize the process of abstraction by which a hierarchical controller is constructed. We describe a planning algorithm that uses a programmer-defined task hierarchy to constrain the search space of finite-state controllers, and prove that this algorithm converges to a hierarchical finite-state controller that is ε-optimal in a limited but well-defined sense, related to the concept of recursive optimality.

Introduction

Hierarchical approaches to planning have a long history. By organizing the construction of a plan at different levels of abstraction, a hierarchical planner can leverage the structure of a planning problem for computational gain, often by decomposing it into related sub-problems that can be solved independently.

In recent years, there has been much interest in hierarchical approaches to planning problems that are formalized as Markov decision processes (Sutton, Precup, & Singh 1999; Parr & Russell 1998; Dietterich 2000). Markov decision processes (MDPs) have been adopted in the AI community as a framework for decision-theoretic planning, as well as for closely-related research in reinforcement learning. With some exceptions (Hernandez & Mahadevan 2001; Pineau, Roy, & Thrun 2001; Theocharous & Mahadevan 2002), most work on hierarchical approaches to MDPs assumes complete observability. In this paper, we discuss an extension of the hierarchical approach to partially observable MDPs (POMDPs). POMDPs model planning problems that include uncertainty about the effects of actions *and* uncertainty about the state of the world – due, for example, to imperfect or unreliable sensors. Although POMDPs provide a general and realistic model for decision-theoretic planning, they are very computationally challenging. Thus, it is especially important to find ways of exploiting problem structure in solving POMDPs, including hierarchical structure.

In this paper, we consider an approach to hierarchical planning that leverages a programmer-defined task hierarchy to decompose a POMDP into a number of smaller, related POMDPs. To solve these sub-problems, we use a policy iteration algorithm for POMDPs that represents a policy as a finite-state controller (Hansen 1998b). We show how to combine the controllers that represent solutions to sub-problems into a hierarchical finite-state controller. We prove that our planner converges to a hierarchical finite-state controller that is ε-optimal in a limited but well-defined sense, related to the concept of recursive optimality introduced by Dietterich (2000) to evaluate hierarchical policies for completely observable MDPs. To the best of our knowledge, this is the first optimality proof for a hierarchical approach to POMDPs. To provide a foundation for these results, we discuss some extensions of the POMDP framework, including a theory of indefinite-horizon POMDPs and some elements of a theory of partially observable semi-Markov MDPs, that allow us to formalize the process of abstraction by which a hierarchical controller is constructed. This foundation is similar to the foundation for hierarchical approaches to completely observable MDPs.

The paper is organized as follows. We begin with a brief review of relevant background about POMDPs and previous work on hierarchical decomposition of MDPs. Then we discuss how to represent a policy for a POMDP as a hierarchical finite-state controller, and in particular, how to model the effects of abstract actions that take the form of finite-state controllers. We next develop a theory of indefinite-horizon POMDPs and discuss the theory of partially observable semi-Markov MDPs. On this foundation, we develop a planning algorithm that finds hierarchical finite-state controllers for POMDPs, and analyze its convergence. We present preliminary computational results and discuss a number of topics for future work, including exploration of the complementary relationship between hierarchical decomposition and state abstraction.

Background

We begin with a brief review of relevant background about POMDPs and previous work on hierarchical approaches to MDPs.

POMDPs and policy iteration

We consider a discrete-time POMDP defined as a tuple $M = (S, A, Z, R, \mathcal{P})$, where: S, A, and Z are finite sets of states, actions, and observations, respectively; R is a reward function, where $r(s, a)$ represents the reward received in state $s \in S$ after taking action $a \in A$; and $\mathcal{P}$ is a set of Markovian state transition and observation probabilities, where $P(s'|s, a)$ represents the probability of making a transition from state s to state s' after action a, and $P(z|s', a)$ represents the probability of observing $z \in Z$ after action a results in a transition to state s'. The objective is to optimize some function of the sequence of rewards – for example, the expected total discounted reward over an infinite horizon, given a discount factor $\beta \in [0, 1)$.

Although the state of the process cannot be directly observed, it is possible to maintain a vector of probabilities over states, denoted b and called a *belief state*, where $b(s)$ denotes the probability that the process is in state s. If action a is taken and followed by observation z, the successor belief state, denoted b_z^a, is determined by revising each state probability using Bayes' theorem, as follows,

$$b_z^a(s') = \frac{\sum_{s \in S} P(s', z|s, a)b(s)}{P(z|b, a)} ,$$

where $P(s', z|s, a) = P(s'|s, a)P(z|s', a)$ and $P(z|b, a) = \sum_{s,s' \in S} P(s', z|s, a)b(s)$.

It is well-known that a belief state updated by Bayesian conditioning is a sufficient statistic that summarizes all information necessary for optimal action selection. This gives rise to the standard approach to solving POMDPs. The problem is transformed into an equivalent, completely observable MDP with a continuous, $|S|$-dimensional state space consisting of all possible belief states, denoted $\mathcal{B}$. In this form, a POMDP can be solved by iteration of a *dynamic programming operator* that updates a value function $V : \mathcal{B} \rightarrow \Re$, as follows,

$$V'(b) = \max_{a \in A} \left\{ r(b, a) + \beta \sum_{z \in \mathcal{Z}} P(z|b, a)V(b_z^a) \right\} ,$$

where $r(b, a) = \sum_{s \in S} b(s)r(s, a)$.

Although the space of belief states is continuous, Smallwood and Sondik (1973) proved that the dynamic-programming (DP) operator preserves the piecewise linearity and convexity of the value function. This means the value function can be represented by a finite set of $|S|$-dimensional vectors of real numbers, $\mathcal{V} = \{v_1, v_2 \ldots, v_{|\mathcal{V}|}\}$, such that the value of each belief state b is defined as follows:

$$V(b) = \max_{v_i \in \mathcal{V}} \sum_{s \in S} b(s)v_i(s). \qquad (1)$$

This representation of the value function allows the DP operator to be computed exactly and several algorithms have been developed for this (Cassandra, Littman, & Zhang 1997). In turn, the DP operator is the core step of two exact algorithms for solving infinite-horizon POMDPs – value iteration and policy iteration. We briefly review a policy iteration algorithm due to Hansen (1998b) that plays a central role in this paper.

Several researchers have pointed out that a policy for a POMDP can often be represented as a finite-state controller (FSC) and the value function of a FSC can be computed exactly by solving a system of linear equations (Sondik 1978; Kaelbling, Littman, & Cassandra 1998). Indeed, there is an elegant, one-to-one correspondence between the nodes of a FSC and the vectors in a piecewise-linear and convex representation of its value function. Given this method of policy evaluation, Hansen (1998b) showed how to interpret the DP operator for POMDPs as a method of policy improvement that transforms a FSC into an improved FSC. Starting with an initial FSC, the algorithm creates a series of improved FSCs by repeating the following two steps.

The *policy evaluation* step computes the value function of a FSC by solving a system of linear equations,

$$v_i(s) = r(s, \alpha(i)) + \beta \sum_{s' \in S, z \in Z} P(s', z|s, \alpha(i))v_{\delta(i,z)}(s') , \qquad (2)$$

where i is the index of a node of the FSC, $\alpha(i)$ is the action associated with node i, and $\delta(i, z)$ is the index of its successor node if z is observed. Note that each node i is associated with a unique vector v_i.

The *policy improvement* step computes the DP operator and uses the updated value function to improve the FSC by leveraging the fact that each vector corresponds to a potential new node of the FSC. The node is associated with an action, and for each possible observation, a transition to a node of the current FSC. The potential new nodes are used to modify the current FSC as follows. If a node is identical to a node of the current FSC, no change is made. Otherwise, the node is *added* to the FSC. If the vector corresponding to the added node pointwise dominates a vector corresponding to any node in the current FSC, the dominated node is *merged* into the new node. Finally, the algorithm *prunes* from the FSC any node that does not have a corresponding vector in the updated value function, as long as it is not reachable from a node that does have a corresponding vector. (Merging and pruning nodes allows a FSC to shrink as well as grow in size.) The algorithm monotonically improves a FSC and is guaranteed to converge to an ε-optimal FSC after a finite number of iterations. If the DP operator cannot improve a FSC, the FSC is optimal. Given a starting belief state for a POMDP, the initial node of the FSC is selected as the node corresponding to the vector that optimizes Equation (1).

Empirical results indicate that policy iteration is between 10 and 100 times faster than value iteration, due to the fact that it takes fewer iterations to converge and the complexity of policy evaluation is negligible compared to the complexity of the dynamic programming operator for POMDPs. Policy iteration plays a central role in this paper not only because of its convergence guarantees, but because it represents a policy as a FSC, and this representation allows us to extend the concepts of action abstraction and action decomposition to POMDPs in a natural way.

Hierarchical approaches to MDPs

Several approaches to hierarchical decomposition of completely observable MDPs have been developed (Sutton, Precup, & Singh 1999; Parr & Russell 1998; Hauskrecht *et al.* 1998; Dietterich 2000). We briefly review some common elements of these approaches. This will guide us in developing a hierarchical approach to POMDPs.

All of these approaches treat closed-loop policies that represent complex behaviors as abstract actions. To ensure that an abstract action terminates and returns control to a higher level, it is associated with a set of terminal states. As soon as a terminal state is entered, the abstract action stops. Because the closed-loop policy corresponding to an abstract action can take a varying and indefinite number of steps to finish execution, an MDP with abstract actions is formalized as a semi-Markov decision process – a generalization of the MDP model that allows actions with durations that vary probabilistically.

The closed-loop policy corresponding to an abstract action can be hand-crafted, or it may correspond to a policy that is the result of solving a lower-level MDP. In the latter case, the lower-level MDP is typically an indefinite-horizon (or episodic) MDP for which a policy eventually terminates. In some cases, a programmer-defined task hierarchy is used to decompose an MDP into a collection of hierarchically-related sub-MDPs at various levels of abstraction. Each sub-MDP has an action set that contains primitive actions of the original MDP, or abstract actions that correspond to policies for sub-MDPs.

Commitment to hierarchical decomposition of a policy may cause sub-optimality by restricting the space of policies that is considered in solving an MDP. Thus, alternative definitions of optimality have been introduced. A policy is said to be *hierarchically optimal* if it is best among all policies that are consistent with the constraints of the hierarchical decomposition. A policy is said to be *recursively optimal* if every MDP in the task hierarchy has an optimal policy given policies for its subtasks that are recursively optimal. The distinction pertains to whether optimization is relative to, or independent of, the calling context. To achieve hierarchical optimality, policies for subtasks must be be context-dependent, that is, dependent on policies at higher levels of the hierarchy. Dietterich (2000) introduced the concept of recursive optimality to allow policies to be context-free. Recursive optimality only guarantees that policies for subtasks are locally optimal with respect to the context in which they are executed. Thus, the concept of recursive optimality is not as strong as the concept of hierarchical optimality. Nevertheless, it is believed to have advantages. For example, Dietterich claims that it allows greater state abstraction.

Although there has been some interesting work on hierarchical planning for POMDPs, it relies heavily on approximation (Hernandez & Mahadevan 2001; Pineau, Roy, & Thrun 2001; Theocharous & Mahadevan 2002). We would like to develop a hierarchical approach to POMDPs that allows some guarantee about the quality of a policy, similar to the limited forms of optimality defined for hierarchical approaches to MDPs. After a discussion of hierarchical FSCs, we develop some extensions of the POMDP framework that will allow us to do this. Then we describe a hierarchical planner that is guaranteed to converge to a limited form of recursive optimality.

Hierarchical finite-state controllers

In our hierarchical approach to POMDPs, we represent a policy as a FSC. Given this representation, a policy that reflects the action decomposition corresponding to a task hierarchy can be represented as a hierarchical FSC. In preparation for defining a hierarchical FSC, we define a FSC as follows.

Definition 1 *A finite-state controller (FSC) is a tuple* (Q, α, δ, T)*, where* Q *is a set of controller states (hereafter called nodes), with* q_0 *a distinguished initial node;* $\alpha : Q \rightarrow A$ *is a mapping from nodes to actions;* $\delta : Q \times Z \rightarrow Q$ *is a node transition function; and* $T \subseteq Q$ *is a set of terminal nodes.*

We include terminal nodes because it is important for FSCs to terminate in order to allow a sub-controller to pass control back to a higher-level controller.

Representing a policy as a FSC allows us to adopt the following concept of action abstraction.

Definition 2 *An abstract action is a FSC that is treated as a single action. Executing an abstract action generates a sequence of actions of the FSC that begins with the action associated with its initial node, continues with a series of actions that depends on the sequence of observations received, and ends with the action associated with one of its terminal nodes.*

Since we use A to refer to a set of primitive actions, we use $\overline{A}$ to refer to an action set that may contain abstract actions. We define a hierarchical FSC by allowing the action associated with a node of a FSC to correspond to an abstract action, i.e., another FSC.

Definition 3 *A hierarchical FSC is a tuple* $(Q, \overline{\alpha}, \delta, T)$ *defined as above, except that* $\overline{\alpha} : Q \rightarrow \overline{A}$ *is a mapping from nodes to actions that may be abstract actions. We require a hierarchical FSC to have a finite number of "levels" and the lowest-level FSCs in the hierarchy have only primitive actions.*

Of course, a hierarchical FSC is a special case of a FSC. Any hierarchical FSC can be "expanded" into a conventional or "flat" FSC by replacing its abstract actions with their corresponding FSCs.

In this paper, we assume that a hierarchical FSC is a policy for a POMDP. All the actions at its lowest level are *primitive actions* from the action set of the POMDP, and these are associated with observations from the observation set of the POMDP. (Abstract actions are either associated with a null observation, as we explain in a moment, or with a kind of *abstract observation*, as discussed at the end of the paper.) Given a reward function and transition and observation probabilities for this POMDP, we now describe how to compute a reward function and transition and observation probabilities for an abstract action in a hierarchical FSC. The advantage of representing a policy for a POMDP as a FSC is that when we treat this policy as an abstract action, we can compute an exact model of its effects, as follows.

Reward function The reward function of an abstract action is computed by the policy evaluation method for a FSC. For each terminal node $t \in T$ of the FSC, an $|S|$-dimensional vector v_t is set equal to the immediate reward vector for the action associated with the terminal node. For each non-terminal node i of the FSC, an $|S|$-dimensional vector v_i is computed by solving the system of linear equations given by Equation (2). The immediate reward function of the abstract action is the vector v_0 that corresponds to the initial node q_0. It represents the expected cumulative reward received by starting the FSC in the initial node and continuing until a terminal node is reached.

State transition probabilities The state transition probabilities for an abstract action are computed by the standard method of computing absorption probabilities in a Markov chain. This is based on the same insight used by policy evaluation: a FSC for a POMDP fixes a Markov chain with a state space that is the cross-product of Q and S. The absorbing states of the chain are those associated with a terminal node of the FSC.

Let $P_i(\tilde{s}|s)$ denote the probability of making a transition from state s to state $\tilde{s}$ of the POMDP by executing a FSC beginning from node i and continuing until termination. $P_i(\tilde{s}|s)$ can be computed by assuming that for each terminal node $t \in T$ of the FSC,

$$P_t(\tilde{s}|s) \quad = \quad \begin{cases} 1 & \text{if } s = \tilde{s} \\ 0 & \text{if } s \neq \tilde{s}, \end{cases}$$

and by computing state transition probabilities for each non-terminal node i by solving the following system of $|Q - T||S|^2$ linear equations:

$$P_i(\tilde{s}|s) = \sum_{s' \in S, z \in Z} P(s', z|s, \alpha(i)) P_{\delta(i,z)}(\tilde{s}|s'). \qquad (3)$$

The state transition probabilities of the abstract action corresponding to this FSC are set equal to $P_0(\tilde{s}|s)$, where q_0 is the initial node of the FSC.

For any node i and starting state s, $\sum_{\tilde{s}} P_i(\tilde{s}|s) = 1$ only if the FSC corresponding to the abstract action is guaranteed to terminate. By analogy to the concept of a *proper policy* (Bertsekas 1995), we define a *proper FSC* as follows.

Definition 4 *A proper FSC eventually terminates with probability one.*

Later, we give conditions under which a FSC that represents a policy for a POMDP is guaranteed to terminate. However, this method of computing state transition probabilities can be used even when termination is not guaranteed.

Observation probabilities Now we face an interesting question. How do we associate an observation with an abstract action? What information is provided by executing an abstract action? One possibility is to consider the entire execution history of the FSC corresponding to the abstract action as an observation. But this is clearly infeasible, since the observation space could be infinite! Another possibility is to consider the last observation received by the FSC (i.e., the observation received after the action associated with a terminal node) as the observation associated with the abstract action. This may turn out to be useful, and would allow us to compute observation probabilities for an abstract action in a way that is similar to how we compute state transition probabilities. We discuss this again near the end of the paper.

But for now, we assume that a null observation is associated with each abstract action. Although this is not a necessary assumption, it will help to simplify our exposition. It also conveys an intuition about the modularity of a hierarchical policy. By associating a null observation with an abstract action, all of the history information accumulated in the course of executing the FSC is treated as local information that is unavailable at the next higher level of control. Note that memory of what happened before the sub-controller was invoked is preserved and still available at the higher level. At the end of the paper, we discuss how to relax the assumption that a null observation is associated with an abstract action.

Mean duration times We consider one last aspect of the model of an abstract action – the mean number of primitive time steps an abstract action takes to execute. This information is needed to do appropriate discounting in a semi-Markov decision processes that includes abstract actions, if we use a discount factor. The mean duration of an abstract action, measured in number of primitive time steps, is computed by solving the following system of $|Q - T||S|$ linear equations,

$$N_i(s) = n(s, \alpha(i)) + \sum_{s' \in S, z \in Z} P(s', z|s, \alpha(i)) N_{\delta(i,z)}(s'),$$

where $N_i(s)$ is the mean number of primitive steps until termination of the FSC, starting from non-terminal node i, and $n(s, \alpha(i))$ is the mean duration of action $\alpha(i)$. We assume that $n(s, \alpha(i))$ is equal to 1 if $\alpha(i)$ is a primitive action. For a terminal node t, $N_t(s) = n(s, \alpha(t))$ for all $s \in S$. The mean duration time of an abstract action, conditioned on the underlying state s, is $N_0(s)$, where q_0 is the initial node of the FSC corresponding to the abstract action. Although this system of equations assumes a proper FSC, it can be appropriately modified for a FSC that does not terminate with probability one.

Indefinite-horizon POMDPs

Our definition of a FSC includes a set of terminal nodes. In a framework for hierarchical control, this is important because terminal nodes allow a sub-controller to stop and pass control back to a higher-level controller. In this section, we discuss how to represent and guarantee termination of a sub-controller, and more generally, of a policy for a POMDP. This question has received very little attention in the literature on POMDPs, but is crucial in a hierarchical approach to control.

Of course, for finite-horizon POMDPs, a policy terminates after a fixed number of time steps. But for most planning problems, the number of time steps needed to achieve an objective cannot be fixed in advance, and a more flexible model is needed. If we formalize a sub-POMDP as an infinite-horizon POMDP, however, a policy may never terminate and control may never be passed back to a higher level. In work on hierarchical control of completely observable MDPs, each sub-MDP is formalized as an indefinite-horizon MDP. This allows a policy to terminate after an indefinite number of time steps, and under certain conditions, it is possible to ensure eventual termination with probability one. Indefinite-horizon MDPs include the class of stochastic shortest-path problems (Bertsekas 1995). Such problems include a non-empty set of terminal states. When one of these states is entered, a policy stops execution. Because terminal states can be viewed as goal states, stochastic shortest-path problems are typically used to model decision-theoretic planning problems that involve goal achievement (or, in the case of hierarchical control, subgoal achievement).

It is not obvious how to extend the theory of stochastic shortest-path problems to POMDPs, however. In a stochastic shortest-path problem, termination of a policy requires recognition that a terminal state has been reached. But in a POMDP, the underlying state of the environment is not directly observable! A special-case theory of stochastic shortest-path POMDPs has been developed that assumes a POMDP contains a special observation that reliably indicates when a terminal state has been reached (Patek 2001). But assuming that terminal states are completely observable is not a general solution. We know of no other attempt to characterize indefinite-horizon POMDPs.[1]

In this section, we propose a framework for indefinite-horizon POMDPs that we believe is general and natural. We do so by thinking about the problem in a slightly different way. In the traditional framework of stochastic shortest-path problems, termination of a policy is associated with a set of terminal states. Because we view termination as a decision, we think it is more natural (or at least, *as* natural) to associate termination of a policy with a set of *terminal actions*, instead of a set of terminal states. We define a terminal action as a special action such that as soon as it is taken and its associated reward is received, execution of the policy terminates. This leads to the following definition.

Definition 5 *An indefinite-horizon POMDP is a POMDP with an action set that includes one or more terminal actions.*[2]

The advantage of this approach to defining indefinite-horizon POMDPs is obvious: it does not require recognition of the underlying state of the process. Of course, we can define the state-dependent reward function for a terminal action in such a way that a large reward (or no reward) is received for stopping in a state that corresponds to a goal state, and no reward (or a large penalty) is received otherwise. Thus, we can still identify goal states in a useful way. However, we no longer require that termination of a policy depends on recognition that a particular goal state has been reached. Instead, we model goal-achievement tasks by defining an indefinite-horizon POMDP in such a way that a policy terminates when the expected value of doing so is optimized, based on a belief distribution over underlying states, including goal states, and a tradeoff between the cost of continued plan execution and the reward for reaching a goal state. In this framework, it may be optimal to stop even if it is not completely certain that the underlying state is a goal state.

This definition of an indefinite-horizon POMDP not only allows partially observed goal states, it fits nicely with our definition of a FSC, which includes a set of terminal nodes. If a policy for an indefinite-horizon POMDP is represented by a FSC, the terminal nodes of the FSC are exactly those nodes that are associated with a terminal action.

Now that we have described a mechanism by which a policy for a POMDP can terminate, we consider whether we can guarantee that it terminates. It is important to note that our hierarchical approach does not require that each sub-controller eventually terminates, although it is important that sub-controllers usually terminate. Our method of modelling abstract actions (represented by FSCs), and of planning with them, can be used even if some sub-controller has a non-zero probability of not terminating, although a discount factor may be required in this case to ensure that the reward function for the abstract action is finite. However, we can identify an important class of POMDPs for which termination can be guaranteed. Consider the class of undiscounted, indefinite-horizon POMDPs for which every non-terminal action incurs a negative reward. We call this class of problems *negative POMDPs*.

Theorem 1 *For negative POMDPs, every FSC found by policy iteration terminates (i.e., is a proper FSC).*

This follows from the fact that a policy that doesn't terminate will have unbounded negative value, and a single-node FSC that always selects a terminal action, or a better policy than this (which must also terminate), can be found with a single iteration of the DP operator.

How do we adapt a DP algorithm for POMDPs so that

[1]It is interesting to note that many examples of POMDPs in the literature are essentially indefinite-horizon POMDPs. Examples include the well-known tiger-behind-a-door problem (Kaelbling, Littman, & Cassandra 1998) and many maze-navigation problems. Because there has been no theory of indefinite-horizon POMDPs, they are formalized as infinite-horizon POMDPs with one or more "reset actions." A reset action is similar to a terminal action in that it represents the last step of a plan, but in these examples, it resets the problem to its starting state. Instead of stopping, the policy repeatedly solves the same problem. Because it accumulates the reward acquired in solving the problem repeatedly over an infinite horizon, it requires a discount factor. But an optimal policy with discounting may not be the same as an optimal policy for an indefinite-horizon problem that is not discounted.

[2]Obviously, it is possible to define indefinite-horizon completely observable MDPs in a similar way. In fact, this results in a framework that is more general than the theory of stochastic shortest-path problems. Any stochastic shortest-path problem can be defined using terminal actions instead of terminal states, by restricting the set of states in which a terminal action can be taken to the terminal states. In addition, this framework lets us define MDPs for which a policy can terminate in *any* state, if doing so maximizes expected value. This may have some advantages in the setting of complete observability, but we do not discuss them here.

it can solve an indefinite-horizon POMDP? The modification is simple. Because a terminal action is the last action executed by a policy, we do not optimize over terminal actions when using the DP operator to create an updated set of vectors. Instead we simply include the reward vectors for each terminal action in the current set of vectors from which the DP operator constructs an updated set of vectors. Essentially, this allows new FSC nodes to link to terminal nodes of a FSC, without allowing terminal nodes to have successor nodes.

A bound on the sub-optimality of a solution found by DP is computed in the standard way by multiplying the Bellman residual, $r = \max_{b \in \mathcal{B}} |V'(b) - V(b)|$, which represents the maximum error per time step, by a term u that represents an upper bound on the expected number of time steps until termination. In the discounted case, $u = \frac{\beta}{1-\beta}$ represents the expected time until termination under the interpretation of the discount factor as a non-zero probability of "terminating" each time step (Bertsekas 1995). In the case of negative POMDPs, an upper bound on the expected time till termination is $u = \lceil \frac{\nu}{\xi} \rceil + 1$, where $\nu = \max_{s \in S, t \in T} v_t(s) - \min_{s \in S, t \in T} v_t(s)$ is the greatest difference of reward for terminating in one state rather than another; ξ is zero minus the largest reward (i.e., it is the smallest cost) for any non-terminal action; and one is added to account for the execution time of a terminal action. In either case, the sub-optimality of the FSC found by policy iteration for POMDPs is bounded by $\varepsilon \leq r \cdot u$, and we have the following result.

Theorem 2 *Policy iteration converges to an ε-optimal FSC after a finite number of iterations.*

For discounted, infinite-horizon POMDPs, this result is a straightforward extension of the similar result for value iteration (Hansen 1998a). We claim that it also holds for negative POMDPs, and will give the proof in a separate paper.

Partially observable semi-Markov decision processes and weak optimality

Most work on hierarchical control of completely observable MDPs adopts the framework of semi-Markov decision processes (SMDPs). Thus, it is natural to expect a well-founded approach to hierarchical control of POMDPs to adopt the framework of partially observable semi-Markov decision processes (POSMDPs). We now consider this possibility.

First, we note that a DP algorithm for completely observable SMDPs does not need to consider the actual time interval between actions, and it only needs to consider the expected time interval (or mean action duration) under the discounted optimality criterion. In that case, the mean action duration is relevant because it affects the degree of discounting. The partially observable case is much more complicated than this. Consider the formula for updating the belief state in a POSMDP,

$$b_{z,t}^a(s') = \frac{P(z|s',a) \sum_{s \in S} P(t,s'|s,a) b(s)}{P(z,t|b,a)},$$

where $b_{z,t}^a$ denotes the belief state that results from taking action a resulting in observation z after interval t; $P(t,s'|s,a)$ is the joint probability distribution of transition interval t and next state s', given that action a is taken in current state s; and

$$P(z,t|b,a) = \sum_{s' \in S} \Big[P(z|s',a) \sum_{s \in S} P(t,s'|s,a) b(s) \Big].$$

The fact that the resulting belief state is conditioned on the actual transition interval, as well as the action and resulting observation, means that the transition interval must be considered in optimization, even when there is no discounting. The DP operator for POSMDPs must sum over all possible transition intervals, as well as all possible observations, in order to exactly update a value function, as follows:

$$V'(b) = \max_{a \in A} \left\{ r(b,a) + \sum_{z \in Z} \sum_{t=0}^{\infty} \beta^t P(z,t|b,a) V(b_{z,t}^a) \right\}.$$

As a result, except for special cases, the DP operator for POSMDPs cannot be computed exactly.[3] Therefore, we propose the following concept of limited optimality.

Definition 6 *A weak (ε)-optimal policy for a POSMDP is a policy that is (ε)-optimal among all policies that are* not *conditioned on the transition interval (i.e., the duration of actions).*

It is straightforward to prove that an exact DP algorithm for POMDPs converges to a weak ε-optimal policy for a POSMDP. In the discounted case, it must still consider expected action durations to do appropriate discounting. But in any case, it ignores the possibility that a policy is conditioned on the actual duration of actions. This means that our policy iteration algorithm searches in a space of FSCs for which the node transition function is conditioned on the observation only, not both the observation and the duration of the action.

For hierarchical POMDPs that contain abstract actions corresponding to FSCs, adoption of this criterion of weak (ε)-optimality seems reasonable for a second reason: we have already chosen to ignore other information that could be relevant in optimization. To update the belief state correctly, we would need to consider not only the duration of an abstract action, but the entire history of observations received in the course of executing the abstract action. This is the only way to ensure exact belief updates and true optimality. We chose to ignore this information when we decided to return a null observation (or limited information) from an abstract action. Although we could use the full history information to do exact belief updates, it is clearly infeasible to enumerate over all possible histories in an optimization

[3]We know of only one paper that discusses the possibility of computing exact solutions for POSMDPs, and it focuses on the finite-horizon case (White 1976). If action durations are discrete, then – because the finite horizon bounds the duration of an action – the set of possible action durations is finite and can be enumerated. It follows that the optimal value function is piecewise linear and convex and can be found by DP. This result does not extend (in any natural way) to the indefinite or infinite horizon case.

algorithm – just as it is infeasible to enumerate all possible action durations.

Therefore, we extend our definition of weak optimality to POSMDPs with abstract actions as follows.

Definition 7 *A weak (ε)-optimal policy for a POSMDP with abstract actions is a policy that is (ε)-optimal among all policies that are not conditioned on the duration of actions or any history information about the execution of the abstract action except that explicitly modelled by its observation function.*

We adopt this definition not only because it seems unavoidable, but because it seems reasonable in the context of hierarchical decomposition. It coincides with our intuition that in a well-designed task hierarchy, memory that is needed to solve a subtask is (mostly) local to that subtask, and can be safely ignored at a higher level of abstraction.

MAXQ Decomposition

In the previous sections, we developed a theoretical framework for hierarchically-structured POMDPs in which a policy is represented as a hierarchical FSC. We believe that this can provide a foundation for generalizing hierarchical approaches to MDPs from the completely observable to the partially observable case. As an example, we show how one hierarchical approach to completely observable MDPs – Dietterich's (2000) MAXQ approach – can be generalized to POMDPs.

One reason for considering the MAXQ approach is that significant effort has already been made to generalize it to POMDPs (Pineau, Roy, & Thrun 2001; Pineau & Thrun 2002). Pineau et al. use the MAXQ approach to improve the scalability of value iteration for POMDPs. Their hierarchical algorithm has a number of attractive features and shows impressive empirical performance. For example, it has been successfully used to control a mobile robotic assistant in a nursing home. However, their approach relies on several approximations, especially in modelling abstract actions, and it cannot make any guarantees about the quality of the policy it finds.

In this section, we present an alternative generalization of the MAXQ approach that represents abstract actions as FSCs and uses policy iteration as a planning algorithm, instead of value iteration. This lets us to make some limited but well-defined claims about the quality of the policy found by our planner. We begin with a review of the MAXQ approach and Pineau et al.'s generalization, and then describe our alternative.

The key idea of the MAXQ approach is to decompose an MDP into a collection of smaller MDPs based on a programmer-defined task hierarchy. Figure 2 shows a simple example of a task hierarchy. The leaves of the task hierarchy correspond to primitive actions. The internal nodes (represented by boxes) correspond to tasks, or MDPs, for which a policy can be found and treated as an abstract action of the MDP corresponding to the node above it in the task hierarchy. The action set of each MDP contains the primitive and abstract actions represented by its child nodes in the task hierarchy.

Each subtask in this hierarchy represents a sub-goal in a hierarchical decomposition of the original MDP. A policy for the subtask terminates when the sub-goal is achieved. In the completely observable case, termination is associated with a set of terminal states and the sub-goal is modelled by associating rewards with the terminal states. Dietterich even allows the designer of a task hierarchy to create artificial sub-goals if this helps decompose a problem, by defining a *pseudo-reward function* for the MDP corresponding to a subtask.

We follow Pineau et al. in associating each internal node of the task hierarchy with a POMDP. Primitive POMDPs have primitive actions only, and abstract POMDPs have abstract actions (and possibly primitive actions). We also adopt Pineau et al.'s general approach to solving a POMDP represented in this way. We use an exact DP algorithm to compute a policy for each subproblem, traversing the task hierarchy in a bottom-up order. After computing a policy, we treat it as an abstract action in the next higher-level POMDP. This requires computing a reward function and transition model for the abstract action.

Our approach departs from the approach of Pineau et al. in the following ways. We represent a policy as a FSC; we formalize each POMDP in the task hierarchy as an indefinite-horizon POMDP (as defined earlier) to allow termination of a FSC without requiring recognition of an underlying terminal state; and we use policy iteration instead of value iteration to solve a POMDP. Representing a policy as a FSC allows us to compute an exact model for an abstract action. Whereas Pineau et al. treat a policy for a subtask as a single abstract action for which a reward and transition model is approximated, we create one abstract action for each node of the FSC representing the policy. Thus, in our approach, a FSC is converted to a set of abstract actions, one for each of its nodes. Computing models for these abstract actions adds no complexity to the model computation step because we compute a model for all nodes of a FSC simultaneously. However, it does increase the size of the action set of the parent POMDP in the hierarchy. It might seem that this would dramatically increase the complexity of solving the parent POMDP, but in practice it does not. First, the $O(|A||\mathcal{V}|^{|Z|})$ worst-case complexity of the DP operator depends exponentially on the size of the observation set, polynomially on the size of the vector set, and only linearly on the size of the action set. Second, each of our abstract actions is associated with a null observation. Because it is the number of observations that principally affects the complexity of the DP operator, the set of abstract actions (each with a null observation) increases the complexity of the parent POMDP almost negligibly.

We mention one additional element of our generalization of the MAXQ approach. If a programmer specifies a pseudo-reward function for a subtask in the hierarchy, we associate this pseudo-reward function with a *pseudo-action* that is defined to be a terminal action. (Recall that we introduced terminal actions in the framework of indefinite-horizon POMDPs.) Since the pseudo-reward function is artificial, we view the pseudo-action as similarly artificial and assume it takes zero time steps to execute.

```
Repeat until error bound is less than $\varepsilon$
  for each task in hierarchy, in bottom-up order do
    Use policy iteration to improve (or create) FSC for task
    Compute error bound of FSC
    if not the root task, compute model for abstract
      actions corresponding to nodes of FSC
```

Figure 1: MAXQ-Hierarchical Policy Iteration.

Earlier, we showed that for any POMDP that includes abstract actions, policy iteration converges to a weak ε-optimal FSC. This means that an abstract action may be sub-optimal and associated with some bounded error. If we can show how to propagate the error bounds for abstract actions up through the task hierarchy, we can bound the error of a hierarchical FSC.

Theorem 3 *The sub-optimality of a FSC found by policy iteration in solving an abstract POMDP is bounded by*

$$\varepsilon \leq (r + d) \cdot u,$$

where the Bellman residual, r, and the upper bound on the mean time until the policy terminates, u, are defined in the paragraph preceding Theorem 2, and $d = \max_{\bar{a} \in \bar{A}} \varepsilon_{\bar{a}}$ is a bound on the sub-optimality of any abstract action in the action set of this POMDP (where $\varepsilon_{\bar{a}}$ is the error bound associated with abstract action $\bar{a}$).

The idea is that the sub-optimality of a FSC can increase the sub-optimality of any FSC that uses it as an abstract action, and this needs to be considered in calculating error bounds. In effect, the maximum error for any abstract action is added to the residual of a POMDP that includes this abstract action.

The error bound for the lowest-level FSCs is computed by setting d to 0 because the lowest-level FSCs only have primitive actions. The error bound for FSCs that include abstract actions in their action set is computed using Theorem 3, and the error bound for the hierarchical FSC is the error that propagates to the top of the task hierarchy. If the overall error bound is greater than some pre-specified ε, the algorithm can traverse the hierarchy in bottom-up order again to improve the hierarchical FSC and reduce its error bound further. The algorithm may converge more quickly by focusing effort on improving the sub-controller that contributes most to the overall error.

The pseudocode in Figure 1 shows in very simple outline the structure of our planning algorithm, called MAXQ-hierarchical policy iteration. The outer loop, which repeats the traversal of the task hierarchy until a desired error bound is achieved, ensures eventual convergence.

Theorem 4 *MAXQ-hierarchical policy iteration converges to a recursively weak ε-optimal FSC in finite time.*

Note that in extending the MAXQ approach to POMDPs, we have added two qualifications to Dietterich's recursive optimality result for completely observable MDPs. First, we only guarantee ε-optimality because there is no guarantee that policy iteration can find an optimal FSC. Second, we only guarantee weak optimality because (a) the node transition function of a FSC is conditioned on observations only, not the duration of actions, and (b) we choose to ignore (most) history information that is local to the FSC corresponding to an abstract action.

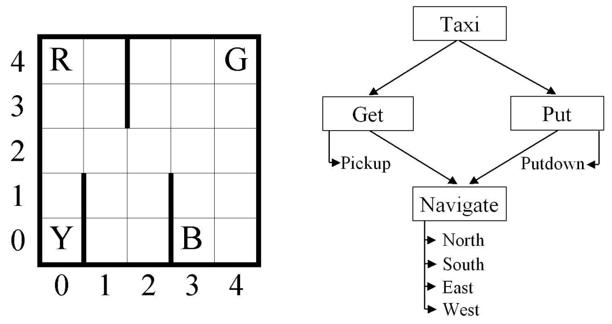

Figure 2: The Taxi domain and its corresponding task hierarchy (Dietterich 2000).

Example

We have implemented the MAXQ-hierarchical policy iteration algorithm and used it to solve several simple POMDPs, including a partially observable version of the Taxi problem used as a motivating example by Dietterich (2000). Figure 2 shows the grid in which the taxi navigates. The taxi can pick up or drop off a passenger in any one of four locations, labelled R, G, Y and B. The problem has three state variables; the location of the taxi (with 25 possibilities), the location of the passenger (with five possibilities corresponding to R, G, Y, B and the taxi) and the destination (with four possibilities), for a total of 500 states. There are six primitive actions; four navigation actions that deterministically move the taxi one square North, South, East or West; a Pickup action; and a Putdown action. There is a reward of -1 for each navigation action. The Pickup action has a reward of -1 when the taxi is at the passenger location, and -10 otherwise. The Putdown action has a reward of +20 when the taxi is at the destination with the passenger in the taxi, and -10 otherwise. We can model this problem as a negative POMDP in which the Putdown action is a terminal action. (Note that we can add pseudo-terminal actions for the pseudo-reward functions in Dietterich's hierarchical decomposition.)

To make this problem partially observable, we assume that the taxi cannot observe its exact location. Instead, all it can observe is the wall placement in all four directions immediately adjacent to its location. With this simple observation function (and using the notation (column, row) to denote a square in the grid), the following sets of states are aliased, i.e., they have the same observation and cannot be distinguished by observation alone; {(0,4), (2,4)}, {(1,4), (4,4)}, {(0,3), (2,3), (0,2), (1,1), (3,1)}, {(1,3), (4,3), (4,2), (2,1), (4,1)}, {(3,3), (1,2), (2,2), (3,2)}, {(1,0), (3,0)}, and {(2,0), (4,0)}. States (0,0), (0,1), and (3,4) each have a unique observation and can be reliably identified by it. The total number of observations is 10. Although the location of the taxi is partially observable, the other two state variables (the location of the passenger and the destination) are completely observable. This creates a POMDP with a hybrid structure (i.e., some state variables are partially observable and some are completely observable) and this can be exploited to solve the problem much more efficiently. In particular, it is unnecessary to optimize the value function for a belief simplex over 500 states, since most of these belief states are impossible give that there is no uncertainty about two of the state variables. Instead, we optimize 20 separate value functions for 20 belief simplices, one for each combination of values for the completely observed variables. The dimension of each simplex is 25, corresponding to the 25 partially observable locations for the taxi. A very simple adaptation of the DP algorithm for POMDPs that exploits this hybrid structure is described by Choi et al. (2001). It allows us to solve this 500-state POMDP without needing to address the issue of state abstraction, a topic we defer for future work.

The MAXQ-hierarchical policy iteration algorithm finds a hierarchical FSC for this problem that is recursively optimal in the weak sense defined earlier. (Although policy iteration is only guaranteed to converge to ε-optimality, we remind the reader that it can detect convergence to optimality.) This takes less than thirty minutes on a Pentium IV 1 GHz processor. The controller has 1094 nodes, and the number of abstract actions created in solving this problem is 1092. The most complex part of the policy is the 433-node sub-controller responsible for navigating to location B, since this location is the most significantly aliased. It is interesting to note that this optimal sub-controller does not guarantee termination in location B, although it guarantees termination. In this respect, it illustrates our indefinite-horizon model.

Extensions

We have outlined an approach to hierarchical planning for POMDPs in which abstract actions are represented as FSCs. To make this approach practical, several developments are still needed. In this section, we briefly discuss two important extensions that we leave for future work.

Task-dependent state abstraction

Currently, exact DP algorithms cannot solve POMDPs that require optimizing a value function for a belief simplex that is defined over more than about 40 or 50 states. Thus, state abstraction is absolutely necessary for scaling up a hierarchical (or any other) approach to solving POMDPs. Preliminary work on state abstraction for POMDPs shows that the DP algorithm can find exact or approximate solutions if the number of *abstract states* can be limited to about the same number (Hansen & Feng 2000; Feng & Hansen 2001).

An important motivation for a hierarchical approach to MDPs is the opportunity to leverage the hierarchical decomposition to perform task-dependent state abstraction. This means that state distinctions that are relevant for achieving one subgoal may be irrelevant for achieving another subgoal, and can be considered or ignored depending on the context. By decomposing a problem hierarchically and solving it on a subtask-by-subtask basis, opportunities for state abstraction may be dramatically increased. This has proved to be a significant advantage of hierarchical approaches to completely observable MDPs (Dietterich 2000; Andre & Russell 2002), and has also been explored for hierarchically-structured POMDPs (Pineau & Thrun 2002). We claim that our hierarchical approach to POMDPs imposes no limit on the overall size of the state space or the number of state distinctions that may be considered at one time or another, as long as the POMDP corresponding to each subtask can be solved by DP. (For similar reasons, it imposes no limit on the overall size of the hierarchical FSC.)

Besides needing state abstraction in a DP algorithm for POMDPs, we need state abstraction in computing the transition model for an abstract action, which requires solving the system of linear equations given by Equation (3). It is especially critical to use state abstraction in this step because the number of equations is quadratic in the number of states.

Observations for abstract actions

Earlier, we made the simplifying assumption that each abstract action is associated with a null observation. This means the abstract action is treated as a black box, and the history of observations received in the course of executing it is invisible to the higher-level controller.

This is a reasonable assumption for an approach based on hierarchical decomposition. It means that memory used by a sub-controller to solve a sub-problem is local memory, and not needed by the higher-level controller. When this assumption is justified by the structure of a problem, it offers great computational leverage. Nevertheless, there will be times when it is useful for a sub-controller to pass some history information back to the higher-level controller that invoked it. This can be done by associating an observation function with the abstract action. It is obviously infeasible to pass the entire execution history of an abstract action back to the higher-level controller as an observation, since this could make the observation space infinite. However, there are ways to return a useful summary. If the POMDP corresponding to a subtask has two or more terminal actions, for example, one could associate a distinct observation with each. Recall that each terminal action corresponds to a distinct terminal node of the sub-controller. Because the set of terminal nodes partitions all possible histories of the sub-controller, the observation associated with each terminal action could be viewed as a summary of the most relevant information from this history. So, in some sense, this can be viewed as a form of observation abstraction.

As one example of how this could be useful, Dietterich describes a fickle taxi problem in which a passenger changes his destination while the abstract action for navigating to the destination is executing. If the abstract action is represented by a FSC with four terminal nodes, one for each destination, the changed destination could easily be passed back to the higher-level controller as an observation. In this case, the size of the observation space is equal to the number of terminal nodes. The point is not simply that it is possible to return history information from a sub-controller in this way (or others), but that the designer of the task hierarchy can control and limit the information returned. In a well-designed task hierarchy, we expect that only a small amount of relevant history information will be returned from a sub-controller to the higher-level controller that invoked it.

Conclusion

We have described an approach to hierarchical planning for POMDPs in which a policy is represented as a FSC. The key idea of our approach is that representing an abstract action as a FSC allows us to compute an exact transition and reward model for it. In turn, this allows us to create a well-defined abstract POMDP and to provide some theoretical guarantees about the quality of a policy found by our hierarchical algorithm. We have covered a lot of ground in developing this approach, and many of the topics we have raised deserve further elaboration and study. However, we have tried to present enough detail to convince the reader that this approach is plausible and worth investigating further.

We conclude by noting that a FSC is a "plan-like" representation of a policy for a POMDP, in contrast to a representation of a policy as a mapping from belief states to actions. It may be that a policy representation that is closer to traditional AI plan representations makes it easier to leverage techniques – such as hierarchical action decomposition – that have proved helpful in scaling-up traditional AI planning algorithms.

Acknowledgments We thank the anonymous reviewers for helpful comments. This work was supported in part by NSF grant IIS-9984952 and NASA grant NAG-2-1463.

References

Andre, D., and Russell, S. 2002. State abstraction for programmable reinforcement learning agents. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI-02)*, 119–125.

Bertsekas, D. 1995. *Dynamic Programming and Optimal Control, Vols. I and II*. Belmont, MA: Athena Scientific.

Cassandra, A.; Littman, M.; and Zhang, N. 1997. Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In *Proceedings of the 13th Annual Conference on Uncertainty in Artificial Intelligence (UAI-97)*, 54–61.

Choi, S.; Zhang, N.; and Yeung, D. 2001. Solving hidden-mode Markov decision processes. In *Proceedings of the 8th International Workshop on Artificial Intelligence and Statistics*.

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

Feng, Z., and Hansen, E. 2001. Approximate planning for factored POMDPs. In *Proceedings of the 6th European Conference on Planning (ECP-01)*.

Hansen, E., and Feng, Z. 2000. Dynamic programming for POMDPs using a factored state representation. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling (AIPS-00)*, 130–139.

Hansen, E. 1998a. *Finite-Memory Control of Partially Observable Systems*. Ph.D. Dissertation, University of Massachusetts/Amherst.

Hansen, E. 1998b. Solving POMDPs by searching in policy space. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98)*, 211–219.

Hauskrecht, M.; Merleau, N.; Kaelbling, L.; Dean, T.; and Boutilier, C. 1998. Hierarchical solution of Markov decision processes using macro-actions. In *Proceedings of 14th Conference on Uncertainty in Artificial Intelligence (UAI-98)*, 220–229.

Hernandez, N., and Mahadevan, S. 2001. Hierarchical memory-based reinforcement learning. In *Advances in Neural Information Processing Systems 13: Proceedings of the 2000 Conference (NIPS*2000)*, 1047–1053. MIT Press.

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

Parr, R., and Russell, S. 1998. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems 10: Proceedings of the 1997 Conference (NIPS*97)*, 1043–1049. MIT Press.

Patek, S. 2001. On partially observed stochastic shortest path problems. In *Proceedings of the 40th IEEE Conference on Decision and Control (CDC 2001)*, 5050–5055.

Pineau, J., and Thrun, S. 2002. An integrated approach to hierarchy and abstraction for POMDPs. Carnegie Mellon University Technical Report CMU-RI-TR-02-21.

Pineau, J.; Roy, N.; and Thrun, S. 2001. A hierarchical approach to POMDP planning and execution. In *Workshop on Hierarchy and Memory in Reinforcement Learning (ICML 2001)*.

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

Sondik, E. 1978. The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. *Operations Research* 26:282–304.

Sutton, R.; Precup, D.; and Singh, S. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* 112:181–211.

Theocharous, G., and Mahadevan, S. 2002. Approximate planning with hierarchical partially observable Markov decision processes for robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

White, C. 1976. Procedures for the solution of a finite-horizon, partially observed, semi-Markov optimization problem. *Operations Research* 24(2):348–358.