An integrated approach to hierarchy and abstraction for POMDPs

Joelle Pineau and Sebastian Thrun

August 2002
CMU-RI-TR-02-21

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

. This research has been sponsored by NSF's ITR, Robotics, and CAREER program, and DARPA's MARS Program (contract N66001-01-C-6018), DARPA's CoABS Program (contract F30602-98-2-0137), and DARPA's MICA Program (contract F30602-01-C-0219), all of which are gratefully acknowledged. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing official policies or endorsements, either expressed or implied, of the United States Government or any of the sponsoring institutions.

Abstract

This paper presents an algorithm for planning in structured partially observable Markov Decision Processes (POMDPs). The new algorithm, named *PolCA* (for *Pol*icy-*C*ontingent *A*bstraction) uses an action-based decomposition to partition complex POMDP problems into a hierarchy of smaller subproblems. Low-level subtasks are solved first, and their partial policies are used to model abstract actions in the context of higher-level subtasks. At all levels of the hierarchy, subtasks need only consider a reduced action, state and observation space. The reduced action set is provided by a designer, whereas the reduced state and observations sets are discovered automatically on a subtask-per-subtask basis. This typically results in lower-level subtasks having few, but high-resolution, state/observations features, whereas high-level subtasks tend to have many, but low-resolution, state/observation features. This paper presents a detailed overview of PolCA in the context of a POMDP hierarchical planning and execution algorithm. It also includes theoretical results demonstrating that in the special case of fully observable MDPs, the algorithms converges to a recursively optimal solution. Experimental results included in the paper demonstrate the usefulness of the approach on a range of problems, and show favorable performance compared to competing function-approximation POMDP algorithms. Finally, the paper presents a real-world implementation and deployment of a robotic system which uses PolCA in the context of a high-level robot behavior control task.

Keywords: Markov decision process, POMDP, reinforcement learning, hierarchical planning, abstraction, robot control, dialogue systems

1. Introduction

The problem of planning under uncertainty has received significant attention in the scientific community over the past few years. Many real-world problem domains, such as robotics and human computer interfaces, are characterized by significant degrees of uncertainty. It has long been recognized that considering uncertainty during planning and decision-making is imperative for the design of robust computer systems.

A long-established approach for planning under uncertainty is that of *Markov decision processes (MDPs)*. MDPs allow for stochastic action effects, described by probabilistic state transition functions. The result of MDP planning is not just a sequence of actions, as would be the case in classical STRIPS-like planning (Fikes & Nilsson, 1971). Rather, it is a *policy* for action selection, which prescribes the choice of action for any possible state that might be encountered during execution. In large domains, computing such policies can be computationally challenging. For example, techniques based on *value iteration*—a highly popular approach for MDP policy computation—often require time quadratic in the number of states.

To overcome this problem, various researchers have developed techniques that exploit intrinsic properties of the domain. The two dominant paradigms for large-scale MDP problem solving are based on *function approximation* and *structural decomposition (abstraction)*. Function approximation techniques generalize across states and actions when calculating policies, thereby reducing the number of states or actions that have to be considered. A seminal example of this approach is Tesauro's (1995) TD-Gammon program, whose integration of multilayer perceptrons into Sutton's (1988) TD family of algorithms led to a policy that achieved world-class backgammon playing strength. Further examples include domains with continuous state spaces where exact methods are inapplicable, such as certain robot path planning problems (Moore & Atkeson, 1995). The other, equally important class of efficient algorithms is based on structural decomposition of the domain. The idea here is that the planning problem can be decomposed into a collection of smaller problems that can be solved separately. This divide-and-conquer strategy can lead to considerable computational savings. A seminal example of the structural decomposition approach is Dietterich's (2000a) MAX-Q algorithm. MAX-Q decomposes the planning problem into a hierarchy of smaller subproblems. Each such subproblem is defined over a reduced state and action space, possesses a subgoal (or, more generally, sub-reward function), and also requires a termination condition. By restricting subproblems to smaller state and action spaces, they can be solved more efficiently. This results in significant speed-ups when solving complex problems. We note that the literature on structural decomposition in MDPs offers a range of alternative algorithms for improved planning through structural decomposition (Singh, 1992; Dayan & Hinton, 1993; Kaelbling, 1993; Dean & Lin, 1995; Boutilier et al., 1997; Meuleau et al., 1998; Singh & Cohn, 1998; Sutton et al., 1999; Parr & Russell, 1998; Wang & Mahadevan, 1999; Andre & Russell, 2001).

All these approaches, however, are limited in that they apply to MDPs only. That is, they can only cope with stochastic action effects but *not* with stochastic or imperfect sensors. MDPs rely on the assumption that the state of the world (or a sufficient statistic thereof) can be sensed reliably and without noise, at any point in time. This is clearly not the case in many real-world problems. In robotics, for example, sensor limitations are pervasive,

and the seemingly 'simple' problem of recovering the state from sensor measurements is the key subject of research in entire subfields (Borenstein, Everett, & Feng, 1996; Cox & Wilfong, 1990). Similarly, research on natural language dialog systems typically seeks to devise techniques for recovering state information through conversing with a person (Asoh et al., 1997; Torrance, 1994). It is those problems, where the state of the world is only partially measurable, that motivate the research described in this paper.

It has long been recognized that a more general framework for addressing planning under sensor limitations is *partially observable Markov decision processes (POMDPs)* (Sondik, 1971; Kaelbling, Littman, & Cassandra, 1998; Monahan, 1982). POMDPs are just like MDPs with one important difference: policies are defined over *information states*, instead of world states, since the latter ones are not directly observable. The space of information states is the space of all beliefs a system might have regarding the world state. Information states are easily calculated from the measurements of noisy and imperfect sensors. In POMDPs, information states are typically represented by probability distributions over world states. Unfortunately, the space of all probability distribution over states is much larger than the state space, and for this reason planning in POMDPs is computationally much harder than in MDPs. In fact, the best known bound for *exact* solution is doubly exponential in the planning horizon, or singly exponential in the cases where the starting state is known. This enormous complexity is arguably the most important obstacle towards applying POMDPs successfully in practice.

The computational complexity of POMDP planning makes the need for fast approximations in POMDPs more prevalent than in MDPs. Unfortunately, the design of efficient POMDP algorithms has received relatively little attention from the scientific community compared to MDPs. Function approximation approaches have been proposed in several papers (White, 1991; Lovejoy, 1991; Littman, Cassandra, & Kaelbling, 1995; Parr & Russell, 1995; Brafman, 1997; Thrun, 2000; Roy & Thrun, 2000; Hauskrecht, 2000; Bayer Zubek & Dietterich, 2000; Bonet & Geffner, 2001), consistently demonstrating significantly faster planning in problems often too hard for exact planners. However, we still lack effective methods that employ structural decompositions of the domain.

This paper describes one such algorithm: a hierarchical POMDP planning algorithm called PolCA. Similar to Dietterich's MAX-Q algorithm, this new algorithm decomposes the planning problem into a hierarchy of subproblems. Each subproblem is treated as a separate POMDP, and solved separately of the other subproblems. At the lowest level of problem solving, PolCA solves a collection of POMDP problems characterized by a reduced action space. The reduced action space leads to significant computational savings, since the number of actions has a major impact on the computational costs of planning. In high-level subproblems, the action set contains *abstract actions*, which subsume lower-level subtasks and their policy. In addition to the reduced action sets, each subproblem uses only a subset of the state and observation spaces. This reduction follows directly from the reduced action set: when eliminating an action, it is also natural to ignore those state/observation features that are only affected by this specific action. Furthermore, the state/observation abstraction in high-level subtasks is delayed until lower-level subtasks have been solved, thus the name of the algorithm, PolCA=**Pol**icy-**C**ontingent **A**bstraction, which represents the fact that abstraction in high-level subtasks is contingent on the policy of lower-level subtasks. As we

will show in this paper, this aspect of our algorithm is a key to obtaining significant state abstraction.

When solving complex problems, many hierarchical MDP approaches calculate and then combine partial value functions to reconstitute the full value function. In contrast, our hierarchical POMDP approach focuses instead on reconstituting a complex policy by combining subpolicies. By moving away from value function reconstitution, we are able to circumvent the subtask termination conditions required by virtually all hierarchical MDP algorithms. This is particularly important given the nature of POMDPs, where state perception is incomplete, rendering the determination of a state-based termination condition impossible. The result is an algorithm that can be applied to a range of structured POMDP problems. Moreover, by solving subpolicies exactly we are able to successfully address partially observable planning problems requiring non-trivial information-gathering sequences. Many such problems are beyond the scope of existing function-approximation POMDP solutions.

In addition to stating the basic PolCA algorithm, this paper offers systematic experimental results in four different domains. These domains possess natural structure which facilitates their hierarchical decompositions. In particular, one task is an instance of an *information-contingent* POMDP, where a successful policy requires the use of multiple information-gathering actions, thus putting it beyond the reach of many function-approximation POMDP approaches. In addition, we also describe a high-level robot control task, which is unique in that a service robot deployed in a real world application (that of assistant to an elderly person) is controlled using explicit POMDP-style representations of uncertainty.

Our algorithm is not the first hierarchical POMDP approach (Hernandez-Gardiol & Mahadevan, 2001; Theocharous, Rohanimanesh, & Mahadevan, 2001; Wiering & Schmidhuber, 1997). Earlier algorithms, however, typically make significant assumptions regarding the ability to fully observe the completion of subproblems, which we do not require.

We nonetheless note that our approach makes several assumptions. The most important being the reliance on a human designer to provide the structural decomposition beforehand. A decade of research on the much simpler MDP paradigm has shown that finding good decompositions automatically is extremely difficult (Pickett & Barto, 2002; Hengst, 2002; Ryan, 2002; McGovern & Barto, 2001; Thrun & Schwartz, 1995). Moreover, it has often been argued (Russell & Norvig, 1995) that providing structural decompositions provides an opportunity to bring to bear background knowledge that a human designer might naturally possess to speed up planning. A second assumption concerns the fact that in certain tasks, artificial sub-reward functions have to be provided. This property is shared with a rich body of work on MDPs (though exceptions exist), and should be thought of as another opportunity to bring to bear background knowledge a human designer might have. Given these assumptions, in extensive comparisons with other state-of-the-art algorithms, we found that PolCA outperforms others in information-contingent tasks, while exhibiting comparable performance in goal seeking tasks.

2. Partially Observable Markov Decision Processes

2.1 Probabilistic Model

We begin our exposition with a brief introduction to partially observable Markov decision processes (POMDPs), setting forth the notation used throughout this paper. This introduction is similar to that by Kaelbling et al. (1998).

Formally, a POMDP is characterized by seven distinct quantities, denoted S, A, Ω, b_0, T, O, and R.

- *States.* The state of the world is denoted s, with the set of all states denoted by $S = \{s_0, s_1, \ldots\}$. The state at time t is denoted s_t, where t is a discrete time index. The state is not directly observable in POMDPs, where an agent can only compute a belief over the state space S.

- *Observations.* To infer a belief regarding the world's state s, the agent can take sensor measurements. The set of all measurements, or observations, is denoted $\Omega = \{o_0, o_1, \ldots\}$. The observation at time t is denoted o_t. Observation o_t is usually an incomplete projection of the world state s_t, contaminated by sensor noise.

- *Actions.* To act in the world, the agent is given a set of actions, denoted $A = \{a_0, a_1, \ldots\}$. Actions affect the state of the world. Choosing the right action as a function of history is the core problem in POMDPs.

POMDPs are instances of Markov processes, that is, the world state s_t d-separates (Pearl, 1988) the future from the past. For notational convenience, it is commonly assumed that actions and observations are alternated over time. This assumption does not restrict the general expressiveness of the approach.

To fully define a POMDP, we have to specify the probabilistic laws that describe state transitions and observations. These law are given by the following distributions:

- The *initial state probability distribution*

$$b_0(s) \quad := \quad Pr(s_0 = s) \tag{1}$$

  is the probability that the domain is in state s at time $t = 0$. This distribution is defined over all states in S.

- The *state transition probability distribution*

$$T(s, a, s') \quad := \quad Pr(s_t = s' \mid s_{t-1} = s, a_{t-1} = a) \tag{2}$$

  is the probability of transitioning to state s', given that the agent is in state s and selects action a, for any (s, a, s'). Since T is a conditional probability distribution, we have $\sum_{s' \in S} T(s, a, s') = 1, \forall (s, a)$. As our notation suggests, T is time-invariant, that is, the stochastic matrix T does not change over time. For time-variant state transition probabilities, the state s must include a time-related variable.

- The *observation probability distribution*

$$O(s, a, o) \quad := \quad Pr(o_t = o \mid s_{t-1} = s, a_{t-1} = a) \tag{3}$$

  is the probability that the agent will perceive observation o upon executing action a in state s. This conditional probability is defined for all (s, a, o) triplets, for which $\sum_{o \in S} O(s, a, o) = 1, \forall (s, a)$.

Finally, we have to define the objective of action selection, which is the function being optimized in the planning process. In POMDPs and MDPs alike, it is common to assume that the agent is given a reward function:

- The *reward function.* $R : S \times A \longrightarrow \Re$, assigns numerical values to state-action pairs. The goal of the agent is to maximize its reward over time. Mathematically, this is commonly defined by an infinite sum of the form:

$$E[\sum_{\tau=t}^{\infty} \gamma^{\tau - t} R_t] \tag{4}$$

  where R_t is the reward at time t, $E[\,]$ is the mathematical expectation, and γ with $0 \leq \gamma < 1$ is a *discount factor* which ensures that the sum in Equation 4 is finite. It is easy to see that the reward function R is more general than goal functions, in which the agent seeks to arrive at one or more goal states.

These items together, the states S, actions A, observations Ω, reward R, and the three probability distributions T, O, and b_0, define the probabilistic world model that underlies each POMDP.

2.2 Belief Computation

The key characteristic that sets POMDPs aside from many other probabilistic models (like MDPs) is the fact that the state s_t is not directly observable. Instead, the agent can only perceive observations $\{o_1, \ldots, o_t\}$, which convey incomplete information about the world's state.

At first glance, it might appear that the agent has to maintain a complete trace of all observations and all actions it ever executed to select an optimal action. However, a well-known fact is that this history can be summarized via a *belief distribution* (Boyen & Koller, 1998), which is the following posterior probability distribution:

$$b_t(s) \quad := \quad Pr(s_t = s \mid o_t, a_{t-1}, o_{t-1}, \ldots, a_0). \tag{5}$$

Because the belief distribution b_t is a sufficient statistic for the history, it suffices to condition the selection of actions on b_t, instead of on the ever-growing sequence of past observations and actions. Furthermore, the belief b_t at time t is calculated *recursively*, using only the belief one time step earlier, b_{t-1}, and the most recent action a_{t-1} and observation o_t:

$$\begin{aligned} b_t(s) \quad &:= \quad f(b_{t-1}, a_{t-1}, o_t) \\ &:= \quad c \cdot \sum_{s'} O(s', a_{t-1}, o_t)\, T(s', a_{t-1}, s)\, b_{t-1}(s') \end{aligned} \tag{6}$$

where c is a normalizing constant.

This equation is equivalent to the decades-old Bayes filter (Jazwinski, 1970), and is commonly applied in the context of hidden Markov models (Rabiner, 1989). Its continuous generalization forms the basis of Kalman filters (Kalman, 1960).

It is interesting to consider the nature of belief distributions. For finite state spaces, which will be assumed throughout this paper, the belief is a continuous quantity, defined over the simplex describing the space of all distributions over the state space S. For very large state spaces, calculating the belief update (Equation 6) can be computationally challenging. Recent research has led to efficient techniques for belief state computation that exploits structure of the domain, sometimes expressed by Bayes networks (Boyen & Koller, 1998; Poupart & Boutilier, 2000). However, we note that by far the most complex aspect of POMDP planning is the generation of a policy for action selection, which will be described further below. For example in robotics, calculating beliefs over state spaces with 10^6 states is easily done in real-time (Burgard et al., 1999). In contrast, calculating optimal action selection policies exactly appears to be infeasible for environments with more than 10^2 states (Kaelbling et al., 1998), not directly because of the size of the state space, but because of the complexity of the optimal policies. For this reason, this paper exclusively addresses the computational complexity involved in policy generation (planning), assuming that the state spaces at hand are small enough that the belief can be calculated exactly.

2.3 Computing a Policy

The central objective of the POMDP is to compute a *policy* for selecting actions. A policy is of the form

$$\pi(b) \quad \longrightarrow \quad a, \tag{7}$$

where b is a belief distribution and a is the action chosen by the policy π.

Of particular interest is the notion of *optimal policy*, which is a policy that maximizes the expected future discounted cumulative reward:

$$\pi^*(b_t) \quad := \quad \operatorname*{argmax}_{\pi} E_\pi[\sum_{\tau=t}^{\infty} \gamma^{\tau-t} R_t]. \tag{8}$$

Clearly, computing an optimal policy is challenging. This is because π is defined over a high-dimensional continuous space, the space of all belief distributions.

The most straightforward approach to finding optimal policies remains the value iteration approach, where iterations of dynamic programming are applied to compute increasingly more accurate values for each belief state b. Let V be a value function that maps belief states to values in $\Re$. Beginning with the initial value function:

$$V_0(b) \quad = \quad \max_a \sum_{s \in S} b(s) R(s, a) \tag{9}$$

then the t-th value function is constructed from the $(t-1)$-th by virtue of the following recursive equation:

$$V_t(b) \quad = \quad \max_a \left[\sum_{s \in S} b(s) R(s, a) + \gamma \sum_{b'} Pr(b' \mid a, b) V_{t-1}(b') \right]. \tag{10}$$

Here, the conditional probability of the belief b' is given by

$$Pr(b' \mid a, b) \quad = \quad \sum_{o \in \Omega} I_{b'=f(b,a,o)} \sum_{s \in S} O(s, a, o)\ b(s). \tag{11}$$

The function I denotes the indicator function that is 1 if its argument is true and 0 otherwise, and the function f is the belief updating function defined in Equation 6. A remarkable result by Sondik (1971) shows that for a finite-horizon problem (i.e., one that terminates in finite time), the value function is a piecewise linear, convex, and continuous function of the belief, composed of finitely many linear pieces.

After t iterations, the value V_t is the expected sum of all (possibly discounted) future pay-offs the agent receives in the next t time steps, for any belief state b. The following one-step look-ahead policy is reminiscent of the update rule (Equation 10):

$$\pi^*_{t+1}(b) \quad := \quad \operatorname*{argmax}_a \left[\sum_{s \in S} b(s) R(s, a) + \gamma \sum_{b'} Pr(b' \mid a, b) V_t(b') \right]. \tag{12}$$

It maximizes, in expectation, the discounted sum of the next $t + 1$ time steps. Thus, it is optimal under the planning horizon $t + 1$. Bounded-time POMDP problems, thus, can be solved exactly with an appropriate choice of the horizon $t + 1$. If the environment is such that the agent might not be able to bound the planning horizon in advance, the policy $\pi^*_{t+1}(b)$ is an approximation to the optimal one whose quality improve with the planning horizon t. Thus, POMDPs with finite state, action, and observation spaces can be solved effectively by selecting an appropriate t.

Unfortunately, the best known exact algorithms for computing the optimal value function appear, in the worst case, to require time doubly exponential in the planning horizon t (Kaelbling et al., 1998); or singly exponential in cases where the initial state is known. More specifically, a single step of value iteration may require space and time on the order of

$$|\Gamma_t| = O(|A||\Gamma_{t-1}|^{|\Omega|}) \tag{13}$$

where $|A|$ is the number of actions, $|\Omega|$ is the number of observations, Γ_{t-1} represents the set of linear components necessary to represent the value function at horizon $t - 1$. In practice, $|\Gamma_t|$ often appears to grow singly exponentially in t, even with clever mechanisms for pruning unnecessary linear functions when computing the value function V_t. This enormous computational complexity is a serious impediment towards applying POMDPs to practical problems. As remarked in the introduction, recent research has led to function approximation-based approaches (White, 1991; Lovejoy, 1991; Littman et al., 1995; Parr & Russell, 1995; Brafman, 1997; Thrun, 2000; Roy & Thrun, 2000; Hauskrecht, 2000; Bayer Zubek & Dietterich, 2000; Bonet & Geffner, 2001), yet we lack effective methods that employ structural decompositions to lower the computational burden imposed by POMDPs.

3. Hierarchical Task Decompositions

The key idea of PolCA is to reduce the complexity of planning by hierarchically decomposing POMDP problems. The basic idea behind PolCA's structural decomposition is analog to that in the hierarchical MDP literature: if the overall task is such that it naturally maps

into a hierarchy of subtasks, a planner should take advantage of such structure, by solving individual subtasks separately, rather than jointly. The computational savings of such a methodology arise from the fact than solving N subtasks is usually computationally more efficient that solving a single task that is N times as large. In MDPs, problem solving usually requires time quadratic in the size of the state space, which gives an indication of the savings one might attain through an optimal decomposition. In POMDPs, the complexity of calculating policies is much larger: it may be doubly exponential in the planning horizon in the worst case, and is singly exponential in most practical problems. Thus, the potential savings one may attain through the structural decomposing of a POMDP problem are much larger—as will be demonstrated empirically in our experimental results section.

One common MDP strategy, defining subtasks via partitioning the state space, is not applicable when decomposing POMDPs where special attention has to be paid to the fact that the state is not fully observable. For this reason, but also because action reduction has a greater impact than state reduction on planning complexity in POMDPs (Equation 13), PolCA relies on a structural decomposition of the task/action space. This decomposition is expressed by a *task hierarchy*, illustrated in Figure 1. The state reduction then follows from the action reduction, in the form of subtask-specific abstraction.

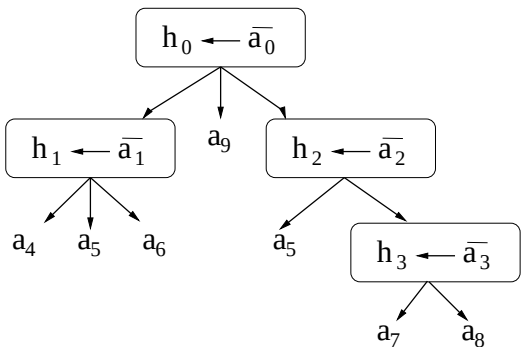


Figure 1: General form of a task hierarchy, $\mathbf{H}$, which structurally decomposes the action space of a POMDP. This hierarchy subsumes a problem that has been divided into four subtasks, $\{\mathbf{h}_0, \mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3\}$, with respective action sets: $A_{h_0} : \{\bar{a}_1, a_9, \bar{a}_2\}$, $A_{h_1} : \{a_4, a_5, a_6\}$, $A_{h_2} : \{a_5, \bar{a}_3\}$, $A_{h_4} : \{a_7, a_8\}$. Actions $\{a_4, a_5, a_6, a_7, a_8, a_9\}$ are from the original action set A. Actions $\{\bar{a}_0, \bar{a}_1, \bar{a}_2, \bar{a}_3\}$ are abstract actions introduced to represent structural constraints.

Before defining task hierarchies formally, let us first discuss the intuition behind them. We conjecture that task hierarchies reflect a natural decomposition of a complex task into simpler subtasks. Formally, a task hierarchy is represented by a tree. At the top level, the root of the tree represents the overall task, as defined by the POMDP. At the bottom level, each individual leaf corresponds to an individual action $a \in A$, which we will henceforth refer to as *primitive action*. Such primitive actions represent the lowest level of policy choice. In between, all internal nodes in the tree represent *subtasks*. Subtasks are defined

over limited sets of other subtasks and/or primitive actions, as specified by their children in the tree. Thus, each internal node in the hierarchy has a double interpretation. Relative to its children, it specifies a task that involves a limited set of subtasks and/or primitive actions. Relative to tasks higher up in the hierarchy, it specifies an *abstract action*, namely the action of invoking this very subtask. We use a bar (as in $\bar{a}$) to denote an abstract action.

Subtasks in PolCA have a second important characteristic, apart from being defined over a limited number of other tasks and actions. Often, the value of a given subtask only depends on a subset of all state features, and similarly, only a subset of all observations are relevant. Examples of such situations will be presented in the experimental results section of this paper. To accommodate such situations, the task hierarchy specifies at each node a subset of states and observations that are relevant to the respective subtask.

Our hierarchical approach relies on a few important assumptions related to domain knowledge. Like most structured MDP approaches, we assume that the hierarchical subtask decomposition is provided by a designer. This constitutes prior knowledge brought to bear on the domain to facilitate planning. In addition, we assume that we are given a POMDP model of the original flat (non-hierarchical) problem.

In the following sections, we will provide formal definition of a task hierarchy, and illustrate how it alters planning and plan execution in POMDPs. As argued below, the notion of task hierarchies raises two fundamental questions: how can we exploit task hierarchies in POMDP planning, and how can we use them afterwords, during plan execution. The first question is non-trivial in that nodes in the hierarchy represent tasks relative to their children, but actions relative to their parents. This raises important issues, for example how to tie value functions between subtasks through abstract actions. The second question is also non-trivial in that a decision has to be made at plan execution time as to which subtask is responsible for selecting the final primitive action to be executed by the agent.

Though PolCA was developed specifically with the goal of solving POMDPs, it can also address the specific case of MDP problem solving. In the next section, we start by introducing the simpler MDP formulation of PolCA, which shares some similarities with earlier algorithms such as MAXQ (Dietterich, 2000a) and ALisp (Andre & Russell, 2001). In the following section, we present the complete (and more general) POMDP formulation.

4. Hierarchical MDPs

The Markov decision process (MDP) is a special case of the POMDP, where the current state of the world is assumed to be *fully-observable* at every time step. An MDP is defined to be a 4-tuple $M = \{S, A, T, R\}$, where S, A, T, R have the same meaning as in POMDPs (namely S is the state set, A is the action set, T defines transition probabilities and R defines costs/rewards.) Under the full-observability assumption of MDPs, a unique observation is emitted by each state. Thus, it is not necessary to consider observation probabilities during planning, and belief tracking is trivial.

4.1 Structural assumptions

The proposed hierarchical MDP algorithm requires a set of structural assumptions, which are similar to those of earlier hierarchical MDP approaches. Formally, we assume we are

given a task hierarchy H, which as for POMDPs is defined by a tree (Figure 1). Tree nodes are either internal (labeled h_i) or leaf nodes.

Each *leaf node* contains a primitive action $a \in A$. Individual actions $a \in A$ may be affiliated with multiple leaves in the task hierarchy. However, each primitive action $a \in A$ should be associated with at least one leaf in the tree—otherwise it will never be chosen and therefore should have been eliminated from the MDP in the first place.

Each *internal node* h corresponds to a well-defined MDP subtask containing:

- $A_h = \{a_k, a_l, \ldots, \bar{a}_p, \bar{a}_q, \ldots\}$, the set of actions allowed in subtask h. Based on the hierarchy, there is one action for each immediate child of h. This set can include both primitive actions $\{a_k, a_l, \ldots\}$ and abstract actions $\{\bar{a}_p, \bar{a}_q, \ldots\}$ (corresponding to lower-level subtasks). The set of actions is completely specified by the given subtask.

- $S_h = \{z_h(s_0), z_h(s_1), \ldots\}$, the set of states relevant to subtask h. The function $z_h()$ defines a state abstraction, mapping *states* to *clusters of states*, such that planning can occur over clusters of states. We assume that the abstraction preserves sufficient resolution to solve h. The function $z_h()$ might be the identity function, in which case a subtask will comprise all states, that is S_h=S. The state abstraction is found automatically as part of the algorithm.

- $G_h = \{s_i, s_j, \ldots\}$, the set of terminal states for subtask h, where $\{s_i, s_j, \ldots\} \in S$. Subtask h is terminated upon reaching state $s \in G_h$, at which point control is returned to the higher-level subtask which initiated h.[1]

- $\bar{R}_h(s)$, the pseudo-reward function specifying the relative desirability of each terminal state. By definition, $\bar{R}_h(s) = 0, \forall s \notin G_h$. In practice, $\bar{R}_h(s)$ is often set to zero for goal-achieving terminal states, and to a large negative value for non-goal terminal states.

Unlike A_h, G_h, and $\bar{R}_h$, which are all provided by a designer, we assume that the clustered set S_h is automatically found for each subtask from the original set S. Finally, PolCA also requires a parameterized model of the MDP domain: $M = \{S, A, T, R\}$. This is in contrast with other hierarchical reinforcement learning algorithms which rely on exploration to acquire the model. Nonetheless we require it to perform the automatic state abstraction.

4.2 Planning Algorithm

For each subtask, the hierarchical planning algorithm optimizes a corresponding *local policy*:

Definition 1: Given h, a subtask with action set A_h, we say that π_h, the policy defined over action subset A_h, is a LOCAL POLICY.

Table 1 describes our hierarchical MDP planning algorithm. It computes the set of local policies (one per subtask) using four simple steps, each of which is explained in further details below.

1. We use notation G_h for terminal states (both goal and non-goal), instead of the more common T_h, in an attempt to avoid confusion with the notation for transition probabilities $T(s, a, s')$.

PLAN-PolCA$_{MDP}(M,\ H)$	0
STEP 1: Re-formulate structured state space: $H \cdot S$	1
For each subtask $h \in H$, following a bottom-up ordering:	2
STEP 2: Set parameters: $T(h \cdot s, a, h \cdot s')$ and $R(h \cdot s, a), \forall s \in S, a \in A_h$	3
STEP 3: Minimize states: $S \rightarrow S_h$	4
STEP 4: Solve subtask: $M_h \rightarrow \pi_h^*$	5
end	6
end	7

Table 1: Main planning function. The function is called using the parameterized MDP model M as the first argument and the hierarchy H as the second argument.

4.2.1 STEP 1—*Re-formulate structured state space*

Because steps 2-4 apply to each subtask separately, it is highly likely that any given world state will have different clustering assignments or final policy for different subtasks. Consequently, step 1 reformulates the state space by adding one more state variable to reflect the *hierarchy state*. This idea was first suggested in the HAM framework (Parr & Russell, 1998).

The new state space $H \cdot S$ is equal to the cross product between the original state space $S = \{s_0, \ldots, s_n\}$ and the hierarchy state $H = \{h_0, \ldots, h_m\}$. The final structured state space is $H \cdot S = \{h_0 \cdot s_0, \ldots, h_0 \cdot s_n, \ldots\ldots, h_m \cdot s_0, \ldots, h_m \cdot s_n\}$.

4.2.2 STEP 2—*Set parameters*

The purpose of the second step is to appropriately translates the transition and reward parameters specified in $M = \{S, A, R, T\}$ to the structured problem representation. This involves translating parameters from the original state space S, to the structured state space $H \cdot S$. This is typically straight-forward for any parameter conditioned on *primitive actions*. However it also involves inferring parameters for the newly-introduced abstract actions, a non-trivial operation.

Given a subtask h, with action set $A_h = \{a_k, a_l, \ldots, \bar{a}_p, \bar{a}_q, \ldots\}$, where $\{a_k, a_l, \ldots\}$ are from A, the set of primitive actions, and $\{\bar{a}_p, \bar{a}_q, \ldots\}$ invoke corresponding lower-level subtasks $\{h_p, h_q, \ldots\}$, then Equations 14–17 translate the parameters from the original MDP to the structured state space.

CASE 1 - PRIMITIVE ACTIONS (full-line transition arrows in Figure 2): $\forall a_k \in A_h, a_k \in A$ and $\forall (s, s') \in S$,

$$T(h \cdot s, a_k, h \cdot s') := T(s, a_k, s') \tag{14}$$

$$R(h \cdot s, a_k) := \begin{cases} \bar{R}_h(s), & \text{if } s \in G_h \\ R(s, a_k), & \text{otherwise.} \end{cases} \tag{15}$$

CASE 2 - ABSTRACT ACTIONS (dotted-line transition arrows in Figure 2): $\forall \bar{a}_p \in A_h, \bar{a}_p \notin A$ and $\forall (s, s') \in S$,[2]

$$T(h \cdot s, \bar{a}_p, h \cdot s') := T(s, \pi^*_{h_p}(s), s') \tag{16}$$

$$R(h \cdot s, \bar{a}_p) := \begin{cases} \bar{R}_h(s), & \text{if } s \in G_h \\ R(s, \pi^*_{h_p}(s)), & \text{otherwise.}^3 \end{cases} \tag{17}$$

One should note that Equations 16–17 depend on $\pi^*_{h_p}$, the final policy of subtask h_p. This is an important feature of this algorithm. It is the reason we traverse the hierarchy from the bottom up, that way, each subtask is parameterized only after all subtasks below it in the hierarchy have been solved. This hierarchical ordering has important implications for state abstraction, for example significantly increasing clustering possibilities in subtasks with multiple terminal states. This will be shown in the experimental section.

4.2.3 STEP 3—*Minimize states*

The goal of this step is to learn a minimization function $z_h(s)$ mapping individual states to clusters of states. State abstraction (also called state clustering and model minimization) is used to reduce the size of the planning problem, thereby accelerating solving. Automatic state abstraction is done on a subtask-per-subtask basis, using an MDP model minimization algorithm by Dean and Givan (1997).

To infer $z_h(s) \to c$, the function mapping states $\{h \cdot s_0, h \cdot s_1, \ldots\}$ to the (expanding) set of clusters $C_h = \{c_0, c_1, \ldots\}$:

I - INITIALIZE STATE CLUSTERING: Let $z_h(s_i) = z_h(s_j)$ if

$$R(h \cdot s_i, a) = R(h \cdot s_j, a), \forall a \in A_h \tag{18}$$

II - CHECK STABILITY OF EACH CLUSTER: A cluster $c \in C_h$ is deemed stable iff

$$\sum_{s' \in c'} T(h \cdot s_i, a, h \cdot s') = \sum_{s' \in c'} T(h \cdot s_j, a, h \cdot s'), \quad \forall (s_i, s_j) \in c, \forall c' \in C_h, \forall a \in A_h \tag{19}$$

III - IF A CLUSTER IS UNSTABLE, THEN SPLIT IT: Let

$$c \to \{c_k, c_{k+1}, \ldots\} \tag{20}$$

such that Step II is satisfied (with corresponding re-assignment of $z_h(s), \forall s \in c$). This is typically done by evaluating several cluster splits and greedily choosing the split that most improves stability.

In summary, in Part I a set of overly-general clusters are proposed; Parts II and III are then applied repeatedly, gradually splitting clusters according to salient differences in model parameters, until there are no intra-cluster differences. This algorithm exhibits the following desirable properties (which we state without proof, see Dean and Givan (1997), Dean, Givan, and Leach (1997) for details):

1. All states in a given cluster have the same value.

2. To clarify the notation in Case 2, consider $\bar{a}_p$, an abstract action in subtask h; $\bar{a}_p$ subsumes subtask h_p and $\pi^*_{h_p}(s)$ is the policy of h_p at state s.

2. Planning over clusters converges to the optimal solution.

3. The algorithm can be relaxed to allow approximate (ϵ-stable) state abstraction.

4. Assuming an MDP with a factored state space, all steps can be implemented such that we can avoid fully enumerating the state space.

This last point is of particular interest: avoiding full state space enumeration is paramount for large problems, both during state abstraction and during planning/solving.

Though not part of the original model minimization algorithm by Dean and Givan (1997), it is also possible to compute an abstraction of $z(s,a)$ (abstracting over $Q(s,a)$), instead of just $z(s)$ (which abstracts over $V(s)$). This is often used when hand-crafting abstraction functions in hierarchical MDPs (Dietterich, 2000a; Andre & Russell, 2002). The advantage of abstracting Q instead of V is that we can allow different state abstractions under different actions, potentially resulting in an exponential reduction in the number of clusters. To abstract Q, we fix any policy that visits every state-action pair and make a new MDP whose states are the state-action pairs of M and whose transitions are given by our fixed policy. We then run the clustering algorithm on this new MDP.

4.2.4 Step 4—*Solve subtask*

The purpose of Step 4 is to learn the value function and policy for subtask h. Since the state clustering step described in Step 3 ensures that all states in a cluster share the same value, it is most convenient to fold value function updates directly into the clustering algorithm. Thus, state minimization augmented with value updates becomes:

- I-III - Same as Equations 18–20
- IV - Update value of each cluster:

$$V(c) = \max_a [R(c,a) + \gamma \sum_{c' \in C_h} T(c,a,c')V(c')], \quad \forall c \in C_h \tag{21}$$

$$\pi(c) = \operatorname*{argmax}_a [R(c,a) + \gamma \sum_{c' \in C_h} T(c,a,c')V(c')], \quad \forall c \in C_h \tag{22}$$

In this case, the algorithm terminates when $V()$ converges. For non-hierarchical MDPs, and under certain conditions, this is equivalent to Boutilier, Dearden, and Goldszmidt's (2000) decision-tree representation of MDP value functions.

Finally, if desired, the optimized approximate value function for the problem can be retrieved by simply looking at the value function of the top subtask: $V_{\pi^*_{h_0}}(h_0 \cdot s)$.

Figure 2 illustrates the entire process for a simple 4-state, 2-subtask problem. The procedure starts with a given model of the 4-state problem, and a given action-based hierarchy containing two subtasks with respective action sets: $A_{h_0} = \{a_2, \bar{a}_1\}, A_{h_1} = \{a_3, a_4\}$. The structured state space augments the 4-state problem by adding a subtask identifier variable $h = \{h_0, h_1\}$. Subtask h_1, being the lower-level one, is addressed first. It starts by undergoing parameterizing, where parameters conditioned on actions $\{a_3, a_4\}$ are transposed from the original model to this subtask. Next, the model minimization procedure reduces

S_{h_1} from four states to two clusters, because of symmetry in the transition probabilities. Finally, value iteration yields the final policy for this subtask, where $\pi(h_1 \cdot s) = a_3, \forall s \in S$. Subtask h_0 is address next. In the parameterization step, it transposes parameters from the original model to describe a_2 and uses the policy of h_1 to model abstract action $\bar{a}_1$. Clustering on subtask h_0 reduces the subtask from four states to three. Finally, value iteration yields the final policy for h_0, as illustrated in the bottom right corner of Figure 2. By generating policies for each subtasks, we have achieved a full planning solution to this problem.

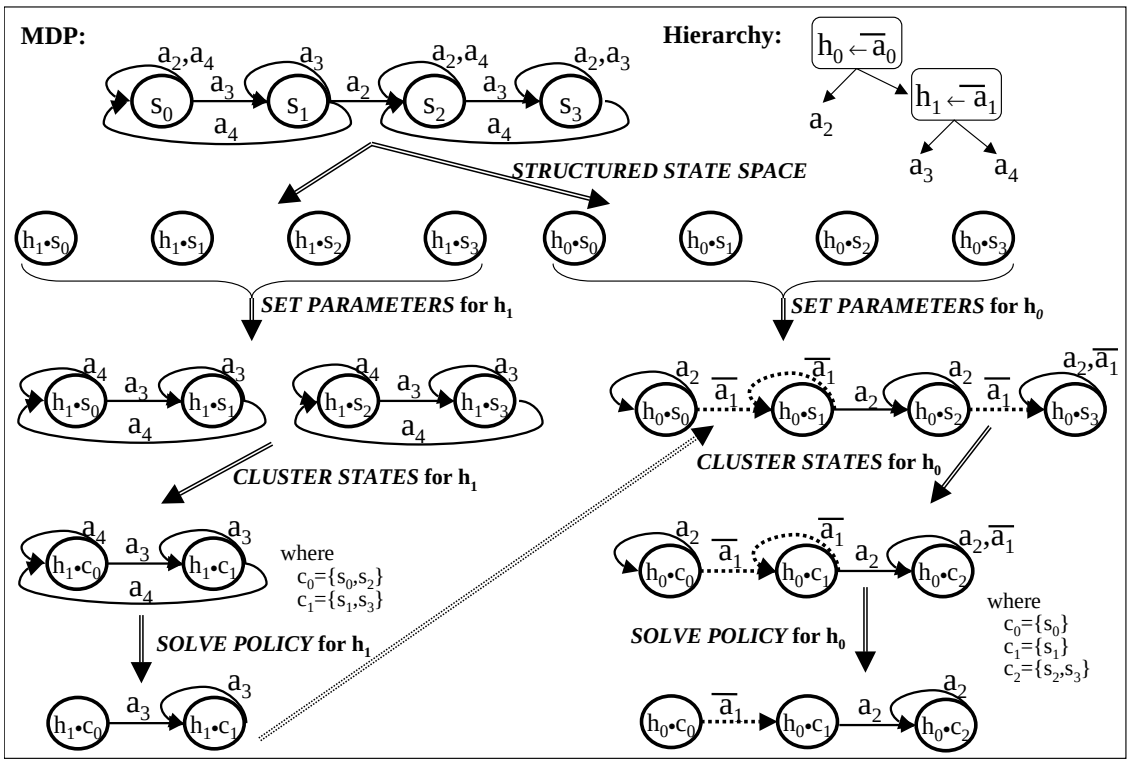


Figure 2: Simple 4-state problem with 2-subtask hierarchy. Non-zero transition probabilities are illustrated in the MDP (top left corner). The subtask hierarchy H is illustrated in the top right corner. Note that the final policy of h_1 (bottom left) is used to model $\bar{a}_1$ in the context of higher-level subtask h_0 (right column). The reward function is not included to keep the example simple.

4.3 Execution Algorithm

It is necessary to specify an execution algorithm that uses the collection of local policies to extract a global policy. The hierarchical execution algorithm maps the current state s_t to a primitive action a_t to be executed by the agent. Rather than pre-computing a global policy explicitly, we propose a recursive online algorithm to generate the next policy action at each time step.

Execution corresponds to a trace down the subtask tree. The algorithm, as described in Table 2, starts by consulting the local policy for the root task; this yields a policy action,

EXECUTE-PolCA$_{MDP}(h,\, s_t)$	0
Let $a_t = \pi_h(s_t)$	1
If a_t is a primitive action	2
Return a_t	3
Else if a_t is an abstract action (i.e. $\bar{a}_t$)	4
Let h_{child} be the subtask spanned by $\bar{a}_t$	5
EXECUTE(h_{child}, s_t)	6
end	7
end	8

Table 2: Recursive execution function. The function is initially called using the root subtask h_0 as the first argument and the current state s_t as the second argument.

either *abstract* or *primitive*. In the case where this is an abstract action, the policy of the corresponding lower-level subtask is consulted, and so on down the hierarchy until a primitive action is selected. Once a primitive action is selected, the execution trace is terminated and the action is applied by the agent.

It is important to emphasize that the full top-down trace through the hierarchy is repeated at every time step. This is a departure from many hierarchical MDP planning algorithms which operate within a given subtask for multiple time steps until a terminal state is reached; this common approach is impractical in POMDPs where we cannot guarantee detection of terminal states. Because it uses a polling approach, consistent with that used by Kaelbling (1993) and Dietterich (2000a), PolCA extends easily to POMDPs.

4.4 Theoretical implications

To better discuss the theoretical performance of our algorithm, we introduce two useful definitions (adapted from Dietterich (2000a)):

Definition 2 - Hierarchical Optimality: Given Π_H, the class of all policies consistent with hierarchy H, then a policy $\pi^* \in \Pi_H$ is said to be HIERARCHICALLY OPTIMAL if it achieves the most reward amongst all policies $\pi \in \Pi_H$.

Definition 3 - Recursive Optimality: Given a subtask h with action set A_h, and Π_h the class of all policies available in h, and assuming fixed policies for subtasks below h in the hierarchy, then a policy $\pi_h^* \in \Pi_h$ is said to be RECURSIVELY OPTIMAL if it achieves the most reward amongst all policies $\pi_h \in \Pi_h$. A set of subtask policies is recursively optimal if all policies are recursively optimal with respect to their children.

These definitions are commonly used in the literature to describe and compare hierarchical MDP planning algorithms. For example, the well-known MAXQ achieves recursive optimality (Dietterich, 2000a), whereas others such as HAM (Parr & Russell, 1998), Options (Sutton et al., 1999) and ALisp (Andre & Russell, 2001) achieve hierarchical optimality. The main difference between the two cases can be attributed to *context*. A recursively optimal solution is obtained when subtask policies are optimized without regards to the context in which they are called. In contrast, hierarchical optimality is achieved by keeping track of all possible contexts for calling subtasks, which is key when subtasks have multiple

goal states. As a result, there is a trade-off between solution quality and representation: hierarchical optimality is a stronger optimality result, but a recursively-optimal algorithm often allows more state abstraction.

Theorem 1: Recursive optimality for PolCA. *Let $M = \{S, A, T, R\}$ be an MDP and let $H = \{h_0, \ldots, h_m\}$ be a subtask graph with well-defined terminal states and pseudo-reward functions. Then the planning algorithm of Table 1 computes π^*_H, a recursively optimal policy for M that is consistent with H.*

Proof: We start by proving that Theorem 1 holds for the case where the planning algorithm is applied without state abstraction (i.e. Step 3 in Table 1). To do this, we use structural induction, which requires that we first show that the policy of any lowest-level subtask is recursively optimal, and then that assuming this, the policy of any higher-level subtask is also recursively optimal.

We first consider h, a low-level subtask containing only primitive actions $A_h = \{a_k, a_l, \ldots\}$ and no abstract actions. Applying Steps 2 and 4 1 yields a local policy π^*_h. By convergence of value iteration (which we apply in Step 4), this policy must be optimal with respect to its action set A_h. Furthermore, because it is strictly optimal, it must also be recursively optimal.

Now consider h, a higher-level subtask containing action set $A_h = \{a_k, a_l, \ldots, \bar{a}_p, \bar{a}_q, \ldots\}$, where abstract actions $\{\bar{a}_p, \bar{a}_q, \ldots\}$ are associated with corresponding subtasks $\{h_p, h_q, \ldots\}$. Assume that these subtasks have respective policies $\{\pi^*_{h_p}, \pi^*_{h_q}, \ldots\}$, all of which have been shown to be recursively optimal. Then applying Steps 2 and 4 yields a policy π_h. We now use a proof by contradiction to show that π^*_h is also recursively optimal.

Assume there exists a policy π^+_h that differs from the π^*_h by at most ϵ, such that $\exists s \in S$ where:

$$V^{\pi^+_h}(s) \;=\; V^{\pi^*_h}(s) + \epsilon \tag{23}$$

and consequently:

$$\max_{a \in A}\left[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^{\pi^+_h}(s')\right] \;=\; \max_{a \in A}\left[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^{\pi^*_h}(s')\right] + \epsilon. \tag{24}$$

Now substituting Equation 23 into Equation 24 and simplifying:

$$\max_{a \in A}\left[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')(V^{\pi^*_h}(s') + \epsilon)\right] \;\geq\; \max_{a \in A}\left[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^{\pi^*_h}(s')\right] + \epsilon \tag{25}$$

$$\max_{a \in A}\left[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^{\pi^*_h}(s')\right] + \gamma\epsilon \;\geq\; \max_{a \in A}\left[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^{\pi^*_h}(s')\right] + \epsilon \tag{26}$$

$$\gamma\epsilon \;\geq\; \epsilon \tag{27}$$

For the general case $0 \leq \gamma < 1$ this can only hold if $\epsilon = 0$, and so we can say that $V^{\pi^+_h}(s) \;= V^{\pi^*_h}(s)$, and similarly $\pi^+_h(s) = \pi^*_h(s), \forall s \in S$. Thus, we conclude that π^*_h must be recursively optimal.

The extension of this proof to the case with state abstraction depends on the proof included in Dean and Givan (1997), which shows that the model minimization algorithm preserves policy quality. **Q.E.D.**

We terminate this section by pointing out that PolCA achieves recursive optimality, rather than the stronger hierarchical optimality, specifically because it fixes low-level subtask policies prior to solving higher-level subtasks. Nonetheless we show in the experimental section that by restricting PolCA to this weaker form of optimality we achieve much greater state abstraction than would be possible otherwise.

5. Hierarchical POMDPs

Any attempt at generalizing hierarchical MDP algorithms to address partially observable MDPs must overcome two obstacles. First, both planning and execution must be re-expressed with respect to *beliefs*, instead of *states*. Second, it is now unreasonable to assume that terminal states (subgoals or otherwise) can be *detected*.[4] This section, which constitutes the cornerstone of this paper, presents the full PolCA algorithm—a POMDP-generalization of the version introduced in Section 4. Much of the algorithm remains unchanged from its MDP formulation, however some important modifications are necessary to accommodate partial observability, and to include the observation space and observation probabilities into the algorithm.

5.1 Structural Assumptions

The structural assumptions necessary for hierarchical POMDP planning are for the most part very similar to the hierarchical MDP assumptions. We add a set of observations relevant for each subtask. Formally stated, given a task hierarchy H, each internal node represents a separate POMDP subtask h defined by:

- $A_h = \{a_k, a_l, \ldots, \bar{a}_p, \bar{a}_q, \ldots\}$, the set of actions allowed in subtask h.
- $S_h = \{z_h(s_0), z_h(s_1), \ldots\}$, the (clustered) set of states relevant to subtask h.
- $\Omega_h^a = \{o_m, o_n, \ldots\}$, the (reduced) set of observations relevant to subtask h, conditioned on each action $a \in A_h$. The observation abstraction excludes those observations which cannot be observed in a given action. The abstraction preserves sufficient resolution to solve h. All observation abstraction subsets are found automatically.
- $G_h = \{s_i, s_j, \ldots\}$, the set of terminal states for subtask h, where $\{s_i, s_j, \ldots\} \in S$. The definition of goal states is only required to help specify the pseudo-reward function, $\bar{R}_h()$ The goal states need not be observed during planning or execution, and furthermore in some POMDP domains it is perfectly acceptable to define $G_h = \emptyset$.
- $\bar{R}_h(s)$, the pseudo-reward function specifying the relative desirability of each terminal state. Typically, in subtasks with uniform reward (e.g. if $R(s,a) = constant, \forall s \in S_h, \forall a \in A_h$), the pseudo-reward is set to zero for goal-satisfying terminal states and a negative constant for non-goal terminal states. In subtasks with a non-uniform reward, the pseudo-reward is often set to the real reward.

4. This is not to say that they cannot be *specified*, but that in the general case, they cannot be fully *observed*. The distinction is useful because in some cases the terminal states must be specified in the process of determining the pseudo-reward function.

As in hierarchical MDPs, we also require a model of the domain, in this case a POMDP model: $M = \{S, A, \Omega, b_0, T, O, R\}$.

5.2 Planning Algorithm

The POMDP formulation of the PolCA planning algorithm remains largely unchanged from the MDP version (Table 1). Nonetheless there are a few notable differences. First, the parameter setting step is extended to include observation probabilities. Next, the state abstraction algorithm is complicated slightly by the need to consider observation probabilities when clustering states. We also introduce automatic observation abstraction. Finally, the actual subtask solving uses appropriate POMDP techniques. Table 3 presents the hierarchical POMDP planning algorithm.

PLAN-PolCA(M,H)	0
STEP 1: Re-formulate structured state space: $H \cdot S$	1
For each subtask $h \in H$, following a bottom-up ordering	2
STEP 2: Set parameters: $T(h \cdot s, a, h \cdot s')$ and $R(h \cdot s, a), \forall s \in S, a \in A_h, o \in \Omega_h$	3
STEP 3: Minimize states: $S \rightarrow S_h$	4
STEP 3b: Minimize observations: $\Omega \rightarrow \Omega_h$	5
STEP 4: Solve subtask: $M_h \rightarrow \pi_h^*$	6
end	7
end	8

Table 3: Main PolCA planning function. The function is called using the POMDP model $M = \{S, A, \Omega, b_0, T, O, R\}$ as the first argument and the hierarchical constraints $H = \{h_0, \ldots, h_m\}$ as the second argument.

5.2.1 Step 1—*Re-formulate structured state space*

The first step is identical in both MDP and POMDP formulations (Section 4.2.1.) Simply stated, a new state space $H \cdot S$ is equal to the cross product between the original state space $S = \{s_0, \ldots, s_n\}$ and hierarchy state $H = \{h_0, \ldots, h_m\}$.

5.2.2 Step 2—*Set parameters*

This step translates the POMDP parameters specified in $M = \{S, A, \Omega, b_0, T, O, R\}$ to the structured state space $H \cdot S$. The specification of the reward and transition parameters is identical to the MDP case (Section 4.2.2), and we now add the specification of the observation parameters.

Given a POMDP $M = \{S, A, \Omega, T, O, R\}$, to set parameters for subtask h we use Equations 28–33:

Case 1 - Primitive actions: $\forall a_k \in A_h, a_k \in A$, $\forall (s, s') \in S$, $\forall o \in \Omega$:

$$T(h \cdot s, a_k, h \cdot s') := T(s, a_k, s'), \tag{28}$$

$$O(h \cdot s, a_k, o) := O(s, a_k, o), \tag{29}$$

$$R(h \cdot s, a_k) := \begin{cases} \bar{R}_h(s), & \text{if } s \in G_h \\ R(s, a_k), & \text{otherwise.} \end{cases} \quad (30)$$

Case 2 - Abstract actions: $\forall \bar{a}_p \in A_h, a_p \notin A$, $\forall (s, s') \in S$, $\forall o \in \Omega$:

$$T(h \cdot s, \bar{a}_p, h \cdot s') := T(s, \pi^*_{h_p}(s), s') \quad (31)$$

$$O(h \cdot s, \bar{a}_p, o) := O(s, \pi^*_{h_p}(s), o) \quad (32)$$

$$R(h \cdot s, \bar{a}_p) := \begin{cases} \bar{R}_h(s), & \text{if } s \in G_h \\ R(s, \pi^*_{h_p}(s)), & \text{otherwise.} \end{cases} \quad (33)$$

As explained in section 4.2.2, $\bar{a}_p$ is an abstract action available in subtask h, where $\bar{a}_p$ subsumes subtask h_p and $\pi^*_{h_p}(s)$ is the policy of h_p at state s. However in this POMDP formulation of PolCA, unlike in the special-case MDP version, the parameter assignment used for Case 2 constitutes an approximation.

5.2.3 Step 3—*Minimize states*

The state clustering procedure for POMDPs extends the MDP model minimization of Dean and Givan (1997) to also consider observation probabilities when checking for stability between clusters. As in MDPs (Section 4.2.3), the automatic state abstraction algorithm starts by selecting a set of initial clusters based on reward parameters. The cluster partition is then gradually refined according to differences in transition *and observation* parameters.

To infer $z_h(s) \to c$, the function mapping states $\{h \cdot s_0, h \cdot s_1, \ldots\}$ to the (expanding) set of clusters $C_h = \{c_0, c_1, \ldots\}$:

I - Initialize state clustering: see Equation 18.

II - Check stability of each cluster: A cluster $c \in C_h$ is deemed stable iff

$$\sum_{s' \in c'} T(h \cdot s_i, a, h \cdot s') O(h \cdot s', a, o) = \sum_{s' \in c'} T(h \cdot s_j, a, h \cdot s') O(h \cdot s', a, o), \quad (34)$$
$$\forall (s_i, s_j) \in c, \forall c' \in C_h, \forall a \in A_h, \forall o \in \Omega$$

III - If a cluster is unstable, then split it: see Equation 20

5.2.4 Step 3b—*Minimize observations*

This step automatically determines observation abstraction Ω^a_h, that defines the subset of relevant observations for each action $a \in A_h$ in subtask h. Observation abstraction is used to accelerate solving; the effect on POMDP solving can be tremendous, since the complexity of even one-step of planning is exponential in the number of observations (Equation 13). Automatic observation abstraction is done on a subtask-per-subtask basis, and within each subtask on an action-per-action basis.

Let $o \in \Omega^a_h$ iff:

$$\sum_{s \in c} O(h \cdot s, a, o_i) = 0, \quad \forall c \in C_h. \quad (35)$$

For each subtask h and action $a \in A_h$, only those observations which maybe be observed need be considered. The effect is to exclude those observations that have zero probability over all state clusters.

5.2.5 STEP 4—*Solve subtask*

This step focuses on learning the POMDP value function and policy for subtask h. In the case of POMDPs, unlike in MDPs, the solving is delayed until after the compact state and observation sets, S_h and Ω_h^a, have been found.

The state and observation abstraction functions are first used to re-define the POMDP parameters in terms of clusters:

$$b_0(c) = \sum_{s \in c} b_0(s), \quad \forall c \in C_h \tag{36}$$

$$T(c, a, c') = \sum_{s' \in c} T(h \cdot s, a, h \cdot s'), \quad s \in c, \quad \forall (c, c') \in C_h, \quad \forall a \in A_h \tag{37}$$

$$O(c, a, o) = O(h \cdot s, a, o), \quad s \in c, \quad \forall c \in C_h, \quad \forall o \in \Omega_h^a, \quad \forall a \in A_h \tag{38}$$

$$R(c, a) = R(h \cdot s, a), \quad s \in c, \quad \forall c \in C_h, \quad \forall a \in A_h \tag{39}$$

Planning over clusters of states and observations is realized by using an exact POMDP solver. We typically apply the incremental pruning algorithm described in Cassandra, Littman, and Zhang (1997), using code obtained from Cassandra (1999). For very large real-world domains, we have also used the Augmented-MDP algorithm described in Roy and Thrun (2000).

5.3 POMDP Policy Execution with Task Hierarchies

The only significant change in hierarchical POMDP execution, compared to the MDP case, is the fact that POMDPs require belief updating at every time step, prior to consulting the policy. Given that each subtask h uses a different state clustering S_h, it follows that its local policy π_h is expressed over a local belief.

Definition 4: Given a subtask h, we say that $b_h(c)$, the belief defined over clusters $c \in C_h$, is a LOCAL BELIEF.

Rather than update the local belief for each subtask separately, using the latest pair (a_{t-1}, o_t), we instead update the global belief $b_t()$ according to Equation 6. As the policy lookup traverses the tree, the local belief for each subtask, $b_t^h()$, is extracted from the global belief:

$$b_t^h(c) = \sum_{s \in c} b_t(s), \quad \forall c \in C_h \tag{40}$$

resulting in a simple projection according to each subtask's state clustering function.

Table 4 describes the complete hierarchical POMDP execution algorithm. This completes our exposition of the general PolCA algorithm.

5.4 Theoretical implications

Unlike in MDPs, where we can guarantee finding a recursively-optimal solution, we are unable to make theoretical claims regarding the quality of our hierarchical POMDP solution. In fact, we can easily demonstrate that the final solution will generally be sub-optimal, simply by considering Equations 31–33. This parameterization of the abstract actions constitutes an approximation, for the simple reason that the subtask policy π_h is only considered

EXECUTE-PolCA(h, b_t)	0
Let $b_t^h(c) = \sum_{s \in c} b_t(s), \ \ \forall c \in C_h$	1
Let $a_t = \pi_h(b_t^h)$	2
If a_t is a primitive action	3
Return a_t	4
Else if a_t is an abstract action (i.e. $\bar{a}_t$)	5
Let h_{child} be the subtask spanned by $\bar{a}_t$	6
EXECUTE(h_{child}, b_t)	7
end	8
end	9

Table 4: PolCA execution function. The function is initially called using the root subtask h_0 as the first argument and the current global belief b_t as the second argument.

at the corners of its belief state (i.e. when the belief is restricted to a single state—$\pi_h(s)$), thereby ignoring any policy action that may be called in high-uncertainty beliefs. This approximation is necessary if subtask h is to be modeled as a POMDP, where by definition parameters are assumed to be *linear in the belief* (e.g. $R(b,a) = \sum_{s \in S} b(s)R(s,a)$, and so on for $T(b,a,b')$, $O(b,a,o)$). Despite this approximation, the empirical results presented in the next section demonstrate the usefulness of the approach for a wide range of POMDP problems.

Embedded in our hierarchical POMDP planning algorithm are two important contributions to the area of model minimization. First, we presented a POMDP extension to the MDP state minimization algorithm by Dean and Givan (1997). Second, we proposed a separate algorithm to perform observation abstraction. We include two theorems stating that these algorithmic contributions are sound with respect to POMDP solving, independent of any hierarchical context. Proofs for both theorems are included in Appendix A.

Theorem 2: Optimality for state abstraction in POMDPs. *Let $M = \{S, A, \Omega, b_0, T, O, R\}$ be a POMDP. Then, the state minimization algorithm of Section 5.2.3 preserves sufficient information to learn π^*, the optimal policy for M.*

Theorem 3: Optimality for observation abstraction in POMDPs. *Let $M = \{S, A, \Omega, b_0, T, O, R\}$ be a POMDP. Then, the observation minimization algorithm of Section 5.2.4 preserves sufficient information to learn π^*, the optimal policy for M.*

6. Experimental Evaluation of our Hierarchical MDP Approach

Although there are few other hierarchical POMDP algorithms with which to compare PolCA, there exists a number of established hierarchical MDP approaches. For this reason, we begin our experimental evaluation by comparing the MDP version of PolCA to three other well-known hierarchical MDP algorithms: HSMQ (Dietterich, 2000b), MAXQ (Dietterich, 2000a) and ALisp (Andre & Russell, 2001). All three assume an action-based subtask decomposition very much like our own, and also leverage state abstraction to accelerate MDP problem solving. However state abstraction is derived by hand in HSMQ,

MAXQ, and ALisp, whereas the clustering is automatic in PolCA. There are other important differences in the type of abstraction exploited in MAXQ and ALisp, compared to PolCA.

We first recall that both MAXQ and ALisp use a decomposed representation of the value function:

$$Q(h \cdot s, a) = Q_r(h \cdot s, a) + Q_c(h \cdot s, a) + Q_e(h \cdot s, a) \tag{41}$$

where Q_r is the expected reward of taking action a, Q_c is the expected reward of completing subtask h after having taken a, and Q_e is the expected reward of finishing the entire task after having completed h. A 2-part $Q_r + Q_c$ decomposition was first introduced in MAXQ.[5] The full 3-part decomposition was later defined in ALisp (Andre & Russell, 2001). The advantage of decomposing the value function in this way resides in the ability to perform state abstraction separately for each component $Q_r() \rightarrow z_r()$, $Q_c() \rightarrow z_c()$, and $Q_e() \rightarrow z_e()$, which in some problems yields much greater abstraction, compared to clustering on the monolithic Q-function.[6] Specifically, it typically allows significant *funnel abstraction*. Funnel abstraction is achieved when a subtask has few terminal states (for example moving a robot to a doorway) by decoupling costs *before* and *after* the terminal states.

Nonetheless, PolCA—which does state abstraction on the non-decomposed $Q()$—has a different advantage when it comes to state abstraction. Specifically, because abstraction in higher-level subtasks takes place *after* lower-level subtasks have been solved, the abstraction in these high-level subtasks only needs to be consistent with *one* policy for each lower-level subtask (as well as all possible policies for the current subtask). In contrast, state abstraction in MAXQ and ALisp must be consistent with *any* policy per lower-level subtask. In addition, because state abstraction is automatic in PolCA, it can take advantage of symmetry in the domain (e.g. reflections, rotations, etc.; see Zinkevich and Balch (2001)) to further reduce the required parameter set.

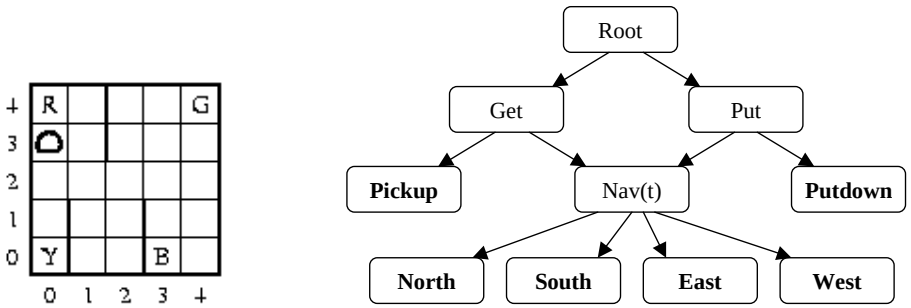
The purpose of the first experiment is to investigate the effects of these differences. Thus, we begin with an experiment that specifically looks at the amount of state abstraction allowed by each method. It is worth emphasizing that the quantity of state abstraction found is of interest because it is a direct indicator of policy computation times.

6.1 Hierarchical MDP Results - Taxi domain

The taxi domain is a well-known hierarchical MDP problem (Dietterich, 2000a). The overall task (Figure 3a) is to control a taxi agent with the goal of picking up a passenger from an initial location, and then dropping him/her off at a desired destination. The initial state is selected randomly, however it is fully observable and transitions are deterministic. Figure 3b represents the MAXQ control hierarchy used for this domain. The structured state space for this domain—called *Taxi1*—can be described by features: {X,Y,passenger,destination,H}, where $H = \{h_{Root}, h_{Get}, h_{Put}, h_{Nav(Y)}, h_{Nav(B)}, h_{Nav(R)}, h_{Nav(G)}\}$. In addition, we consider a second domain—*Taxi2*—in which the passenger can start from any location on the grid,

5. To obtain an even more concise representation, $Q_r(s, a)$ is stored only for primitive actions, and in the case of abstract actions is recursively calculated online when necessary (see Dietterich (2000a) for details).

6. In practice, automatically finding $z_c()$ can be non-trivial. The main difficulty is in estimating $T(h \cdot s_i, \bar{a}_n, h \cdot s')$ and $Q_r(h \cdot s', \bar{a}_n)$ for abstract action.



(a) (b)

Figure 3: The taxi domain is represented using four features: {X,Y,Passenger,Destination}. The X,Y represent a 5x5 grid world; the passenger can be at any of: {Y,B,R,G,taxi}; the destination is one of: {Y,B,R,G}. The taxi agent can select from six actions: {N,S,E,W,Pickup,Putdown}. Actions have a uniform -1 reward. Reward for the Pickup action is -1 when the agent is at the passenger location, and -10 otherwise. Reward for the Putdown action is $+20$ when the agent is at the destination with the passenger in taxi, and -10 otherwise.

compared to only {Y,B,R,G} in Taxi1. This task is harder for MAXQ and ALisp because of the need to represent many more completion costs in $Nav(t)$.

Without any structural assumption, we would require respectively 3000 Q-values (500 states x 6 primitive actions) and 15600 Q-values (2600 states x 6 primitive actions) to represent the solution of the Taxi1 and Taxi2 tasks. Figures 4 and 5 compare state abstraction results for each task, in terms of the number of parameters necessary to learn the solution. In both cases, all four approaches (HSMQ, MAXQ, ALisp and PolCA) yield considerable savings, compared to the full set of parameters required for an optimal MDP solution. Also in both tasks, HSMQ requires many more parameters than MAXQ, ALisp or PolCA, in large part because it only abstracts away full state features (e.g. destination is irrelevant in h_{Get}.)

Taking a closer look at the differences between MAXQ ALisp, and PolCA, we see that in the Taxi1 task, the number of parameters required are very comparable (632 values for MAXQ, 744 for ALisp and 621 for PolCA). ALisp requires a few additional parameters than the other because it represents external completion costs Q_e; and PolCA gets further abstraction in low-level subtasks (e.g. $h_{Nav()}$) because it automatically exploits symmetry in the domain, something the other approaches fail to do.

In the case of the Taxi2 task, the results illustrate the advantage of PolCA in problems where subtasks have multiple termination conditions. In this domain, both MAXQ and ALisp require many more parameters to capture the completion costs (Q_c) of subtasks $h_{Nav()}$ and h_{Get}, since the subtask can terminate in a large number of states (i.e. the passenger can be in any of the 25 cells). PolCA on the other hand uses both symmetry in the domain and constrained subtask ordering to achieve significantly more state abstraction.

Finally, we point out that the abstraction results for Q-learning, HSMQ, MAXQ and ALisp in Taxi1 are published results (Dietterich, 2000a; Andre & Russell, 2002); Taxi2 results are hand-crafted following a careful reading of each algorithm. All algorithms learn the same hierarchically-optimal policy on both of these tasks.

7. Experimental Evaluation of Hierarchical POMDP Approach

In this section, we move on to POMDP domains and present experimental results comparing the PolCA algorithm with other well-established POMDP solving algorithms, both exact and approximate, on three contrasting POMDP problems. The first domain can be solved exactly, and has little structure to speak of. The second domain is a partially-observable variant of the taxi domain (Section 6.1). Finally, the third domain is an information-contingent POMDP which requires a policy with multi-step information-gathering actions, and is therefore unsolvable by many approximate POMDP algorithms.

There exists a large variety of POMDP solution algorithms, both exact and approximate. For the three target problems, we compare the performance of PolCA with that of one exact POMDP solver and six approximate algorithms. These algorithms are all described in Hauskrecht (2000), which offers a thorough comparative analysis of these, as well as other approximate-value POMDP approaches. The approaches we consider include:

1. The *POMDP* solution is obtained using the incremental pruning algorithm (Cassandra et al., 1997; Cassandra, 1999). This is an exact POMDP solution, with exponential time complexity.

2. The $MDP_{lookahead}$ solution is obtained by solving the problem as fully observable using the MDP value iteration algorithm, and thus disregarding any state uncertainty. The control policy is obtained by combining the MDP value function with a single step of exact POMDP solution. The result is a policy that assumes full observability for every step but the current one. This solution is polynomial in the size of the state and action sets.

3. The *QMDP* solution is obtained by solving the problem as fully observable, but keeping track of all $Q(s, a)$ values rather than only $V(s)$ values. The POMDP policy is extracted by linearly extrapolating Q-values to cover the entire belief space and taking the action with maximum expected value. The result is a policy that assumes uncertainty in the immediate time step, but full observability thereafter. This solution is also polynomial in the size of the state and action sets.

4. The *FIB* (*fast-informed bound*) solution is obtained by solving the problem much like the QMDP solution, with the difference that the Q-values are now weighted according to observation probabilities. The POMDP policy is extracted by linearizing the modified Q-values in the belief state and taking the action with maximum expected value. This algorithm is polynomial in the size of the state, action, and observation sets.

5. The $QMDP_{lookahead}$ solution is obtained by first optimizing the QMDP value function, and then using the QMDP solution set to seed a single step of exact POMDP solving.

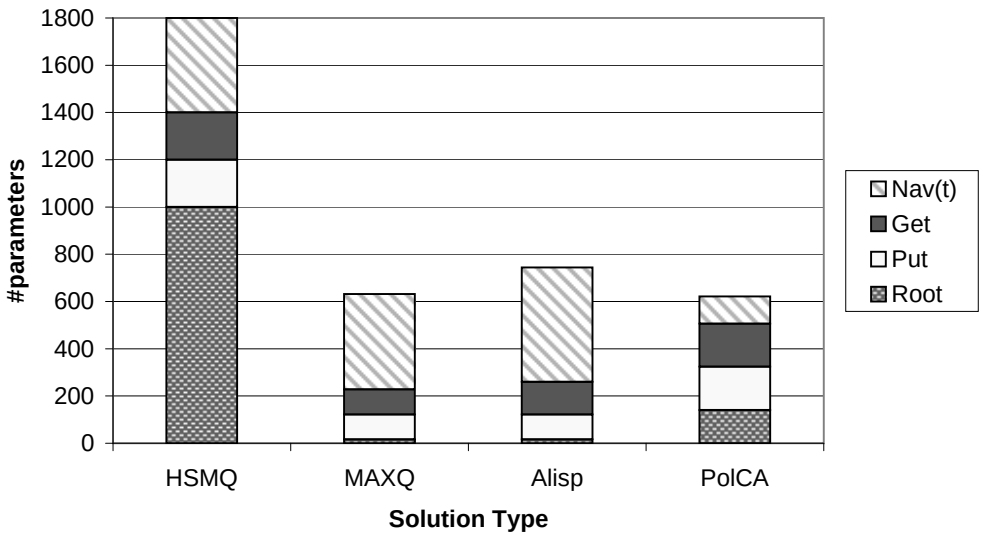


Figure 4: Number of parameters required to find a solution for Taxi1 task. HSMQ=hierarchical semi-Markov Q-learning (Dietterich, 2000b), MAXQ (Dietterich, 2000a), ALisp (Andre & Russell, 2002), PolCA-Q=our algorithm with clustering on Q-values (see end of Section 4.2.3).

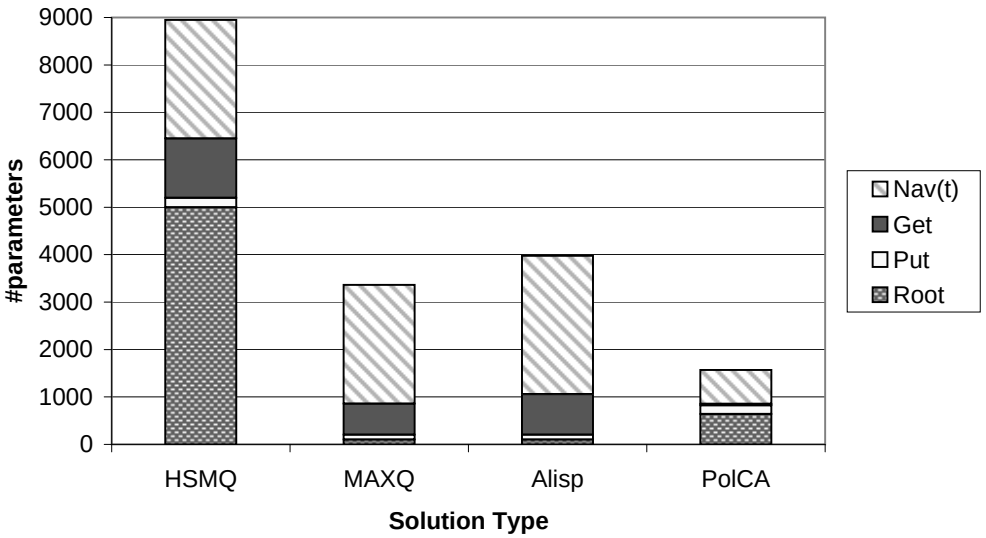


Figure 5: Number of parameters required to find a solution for Taxi2 task.

This solution has exponential time complexity due to the one-step of full POMDP solution.

6. The $FIB_{lookahead}$ solution is obtained by first optimizing the FIB value function, and then using the FIB solution set to seed a single step of exact POMDP solving. This solution has exponential time complexity due to the one-step of full POMDP solution.

7. The *UMDP* planner assumes a completely unobservable version of the given POMDP problem, and applies a full POMDP solution to this worst-case problem characterization. The advantage is that the size of the POMDP solution—namely the number of α-vectors—only grows by a multiplicative factor, $|A|$, with each iteration. Consequently the complexity, though still exponential in the horizon length, is of a lower-degree than for exact POMDP solving. The final policy is extracted from the set of α-vectors, as in any POMDP.

8. The *PolCA* planner uses the algorithm introduced in this paper (more specifically, Table 3.) Each subtask is solved using the exact POMDP algorithm, thus the complexity is exponential in the size of the largest subtask, which is typically substantially less than for exact planning, due to the reduced action, state and observation sets.

The following set of equations, directly adapted from Hauskrecht (2000), illustrates the value-iteration update formula for the various approaches. Each value function expression is mathematically bounded above by the expression listed above it in the list.

$$
\begin{aligned}
V_{t+1}^{MDP}(b) &= \sum_{s\in S} b(s) \max_{a\in A} \left[R(s,a) + \gamma \sum_{s'\in S} Pr(s' \mid a, s) V_t^{MDP}(s') \right] \\
&\geq \\
V_{t+1}^{QMDP}(b) &= \max_{a\in A} \left[\sum_{s\in S} b(s) \left[R(s,a) + \gamma \sum_{s'\in S} Pr(s' \mid a, s) V_t^{QMDP}(s') \right] \right] \\
&\geq \\
V_{t+1}^{FIB}(b) &= \max_{a\in A} \left[\sum_{s\in S} b(s) \left[R(s,a) + \gamma \sum_{o\in\Omega} \sum_{s'\in S} Pr(s', o \mid a, s) V_t^{FIB}(s') \right] \right] \\
&\geq \\
V_{t+1}^{POMDP}(b) &= \max_{a\in A} \left[\sum_{s\in S} b(s) R(s,a) + \gamma \sum_{o\in\Omega} Pr(o \mid a, b) V_t^{POMDP}(b'_o) \right] \\
&\geq \\
V_{t+1}^{UMDP}(b) &= \max_{a\in A} \left[\sum_{s\in S} b(s) R(s,a) + \gamma \max_{\alpha_j\in\Gamma_t} \left[\sum_{s\in S} \sum_{s'\in S} Pr(s' \mid a, s) b(s) \alpha_j(s') \right] \right]
\end{aligned}
$$

7.1 Simulation Domain 1: Part-Painting Problem

The first task considered is based on the part-painting problem described in Kushmerick, Hanks, and Weld's (1995) paper. It was selected because it is sufficiently small to be solved

exactly. It also contains very little structure, and is therefore a valuable sanity test for a structured algorithm such as PolCA.

The task consists of processing a part which may or may not be flawed. If the part is flawed, it must be rejected, and alternately if the part is not flawed it must be painted and then shipped. The POMDP state is described by a Boolean assignment of three state features: *flawed*={*0,1*}, *blemished*={*0,1*}, *painted*={*0,1*}. Not all assignments are included, and thus the state set includes only four possible states: {*unflawed-unblemished-unpainted, unflawed-unblemished-painted, flawed-unblemished-painted, flawed-blemished-unpainted*}. In addition, the domain contains four possible actions: A={*inspect, paint, ship, reject*} and two observations: Ω={*blemished, unblemished*}.

Shipping an *unflawed-unblemished-painted* part yields a $+1$ reward; otherwise shipping yields a -1 reward. Similarly, rejecting a *flawed-blemished-unpainted* piece yields a $+1$ reward, and otherwise rejecting yields a -1 reward. Inspecting the part yields a noisy observation. Finally, painting the part generally has the expected effect:

$$Pr(s_{painted} = 1 \mid a = paint, s_{painted} = 0) = 0.9 \tag{42}$$

$$Pr(s_{painted} = 0 \mid a = paint, s_{painted} = 0) = 0.1 \tag{43}$$

and in the case of a blemished part, generally hides the blemish:

$$Pr(s_{blemished} = 0 \mid a = paint, s_{blemished} = 1) = 0.9 \tag{44}$$

$$Pr(s_{blemished} = 1 \mid a = paint, s_{blemished} = 1) = 0.1 \tag{45}$$

Figure 6 shows the action hierarchy used for this task. Though there are many possible hierarchies, this intuitively seemed like a good hierarchy given minimum knowledge of the problem.

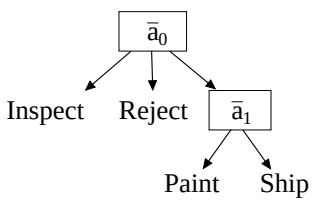


Figure 6: Action hierarchy for part-painting task.

Table 5 contains the results of the experiment with the set of different POMDP planners. For this very small domain, rather than evaluating the performance of each policy using multiple simulation trials, we can look directly at the policy yielded by each different planning method. We observe that the different algorithms each learned one of three policies (as indicated in the *Policy* column.) Figure 7 illustrates those three policies. Policy π^- is clearly very poor: by rejecting every part, it achieves the goal only 50% of the time. On the other hand, optimal policy π^* and near-optimal policy π^+ both achieve the goal 75% of the time (failing whenever action *inspect* returns an incorrect observation). In fact, π^* and π^+ are nearly identical (within a discount factor, $\gamma = 0.95$) since the reward for a paint action is always zero. Nonetheless, the optimal policy π^* yields a higher reward by virtue

of its faster reset rate. The effect of the approximation introduced when modelling abstract action $\bar{a}_1$ (in Figure 6) is seen in policy π^+.

Problem Solution $\lvert S\rvert$=4, $\lvert A\rvert$=4, $\lvert\Omega\rvert$=2	CPU time (secs)	Policy
POMDP	39.28	π^*
$\text{MDP}_{lookahead}$	< 0.01	π^-
QMDP	< 0.01	π^*
FIB	< 0.01	π^+
$\text{QMDP}_{lookahead}$	0.07	π^+
$\text{FIB}_{lookahead}$	0.06	π^*
UMDP	0.33	π^-
PolCA	5.84	π^+

Table 5: Performance results for part-painting task

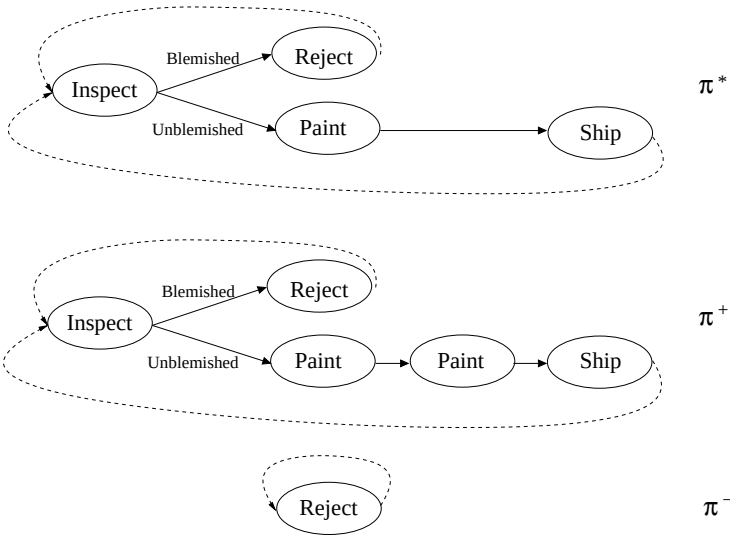


Figure 7: Policies for part-painting task. (Nodes show actions; arrows indicate observations when appropriate; dotted lines indicate a task reset, which occurs after a part has been rejected or shipped).

The planning time for most of the approximate algorithms was extremely short—$< 0.01s$ for $\text{MDP}_{lookahead}$, QMDP and FIB, $0.07s$ for $\text{QMDP}_{lookahead}$, $0.06s$ for $\text{FIB}_{lookahead}$ and $0.33s$ for UMDP—compared to the exact solution. Planning with PolCA also resulted in significant time savings over the exact solution ($5.84s$ vs $39.28s$). We conclude that even for small problems with little structure to speak of, PolCA is able to find reasonable policies.

Nonetheless, we observe that the QMDP approximate algorithm was able to find an optimal policy in a fraction of the time required by PolCA. We attribute the optimal performance of the QMDP to the fact that this problem contains a single information-gathering action. The QMDP algorithm is able to select an information-gathering action when it represents the lowest-risk option in high-uncertainty beliefs.

7.2 Simulation Domain 2: Cheese-Taxi Problem

This section addresses a robot navigation task that is a cross between Dietterich's (2000a) taxi problem (with added state uncertainty) and McCallum's (1993) cheese maze. This domain is interesting because it combines the cheese maze's state uncertainty and the taxi task's hierarchical structure. We assume that a taxi agent is operating in a world that has the configuration of the cheese maze (Figure 8), where the agent must pickup a passenger located at state *s10* and then proceed to deliver the passenger to his/her desired destination (either *s0* or *s4*). The state space is represented using 33 discrete states, formed by taking the cross-product of two state variables: taxi locations {*s0, s1, . . . , s10*} and destinations {*s0, s4, s10*}. The agent has access to seven actions: {*North, South, East, West, Query, Pickup, Putdown*}, and can perceive ten distinct observations: {*o1, o2, o3, o4, o5, o6, o7, destinationS0, destinationS4, null*}.

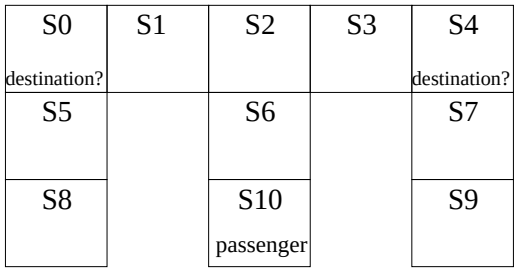


Figure 8: State space for the cheese-taxi task.

One of the first seven observations is received whenever a motion action is applied, partially disambiguating the taxi's current location. As defined in McCallum (1993), this observation is a localization signature indicating wall placement in all four directions immediately adjacent to the location. According to this convention, states {*s5, s6, s7*} look identical, as do respectively {*s1,s3*} and {*s8, s9*}; finally states *s0*, *s2* and *s4* have a unique identifier. The two observations {*destinationS0, destinationS4*} are provided (without noise) in response to the *Query* action, fully disambiguating the taxi destination state variable, but only when the passenger is onboard. The null observation is received after the *Pickup* and *Putdown* actions.

The state transition model is deterministic, for example motion actions have the expected transition effects:

$$Pr(s' = s2 \mid a = North, s = s6) = 1 \tag{46}$$

and so on for the other actions. In addition, the agent incurs a -1 reward for any action, as well as a final reward of $+20$ for delivering the passenger at the correct location. A -10 reward is incurred for applying the *Pickup* and *Putdown* actions incorrectly.

There are three sources of uncertainty in this problem. First, as in McCallum's original cheese maze task, the initial location of the taxi is randomly distributed over maze cells {*s0, s1, ..., s9*} and can only be disambiguated by taking a sequence of motion actions. Secondly, the passenger's destination (*s0* or *s4*) is randomly selected when the passenger is picked-up and can only be observed by using the *Query* action. And thirdly, whenever the taxi has the passenger onboard and is in cells *s2* or *s6*, there is a small possibility that the passenger will change his/her mind and suddenly select the other destination. The new destination can only be observed, once again, by using the *Query* action.

The transition and reward parameters used here are consistent with the original taxi task; the observations parameters (with the exception of the *Query* action) are borrowed directly from the original cheese maze. Finally, we also adopt the taxi task's usual hierarchical action decomposition, as shown in Figure 3b.

This problem, unlike the previously considered part-painting problem, requires the use of a pseudo-reward function in subtasks with a uniform reward (e.g. $h_{Nav()}$ has a uniform reward function $R_{Nav()}(s,a) = -1, \forall(s,a)$). Thus, we artificially reward achievement of partial goals in the $h_{Nav()}$ subtask by using pseudo-reward function:

$$\bar{R}_{Nav(S0)}(s = S0, a) \;=\; 0, \;\; \forall a \in A_{Nav(S0)}$$

and similarly for $S4$ and $S10$. This is identical to the pseudo-reward function used in Dietterich (2000a).

Figure 9 and 10 present results for the cheese-taxi domain, for each of the POMDP solving algorithms. Figure 9 illustrates the sum of rewards to accomplish the full task, averaged over 1000 trials, whereas Figure 10 illustrates the computation time necessary to reach the solution. These figures include results for two different hierarchical POMDP solutions (PolCA and PolCA-NoAbs). PolCA is the full algorithm as described in Section 5. PolCA-NoAbs uses the same algorithm, but without any state or observation abstraction, which leads to a longer solution time. Both use the decomposition of Figure 3b.

The UMDP solution stands apart by its extremely poor performance; its policy is such that the goal of dropping off the passenger is occasionally reached, but not from all starting positions (we forcefully terminated any trial lasting 100 time steps.) Furthermore, the policy never uses the *Query* action and therefore attempts to putdown the passenger at the wrong destination in half of the trials.

Under all other policies, the agent starts with a sequence of motion actions that serves to make progress towards the passenger's location (*S10*) while also disambiguating location. Upon reaching *S10* with full certainty, the agent selects the *Pickup* action. The agent then takes a sequence of motion actions, interspersed with the *Query* action, which leads it to the correct passenger destination.

The main difference between the top four policies (PolCA, PolCA-NoAbs, QMDP$_{lookahead}$, and FIB$_{lookahead}$) and the next four best ones (POMDP, MDP, QMDP and FIB) is in their policy for the first sequence of motion action, going from the random start location to the passenger's station at $S10$. The better policies exhibit optimized action sequences, such that a minimum number of moves is required to simultaneously disambiguate position and

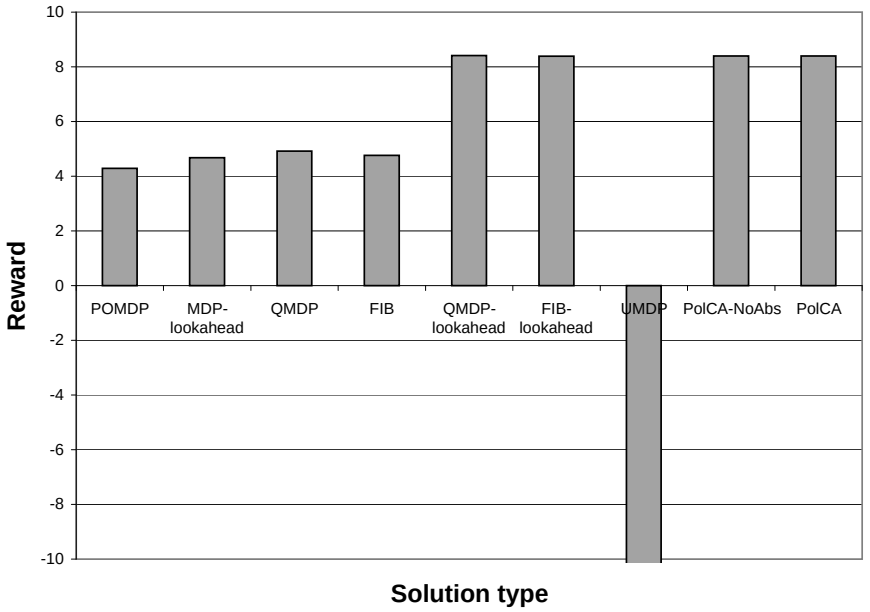


Figure 9: Reward profile for solving the cheese-taxi task.

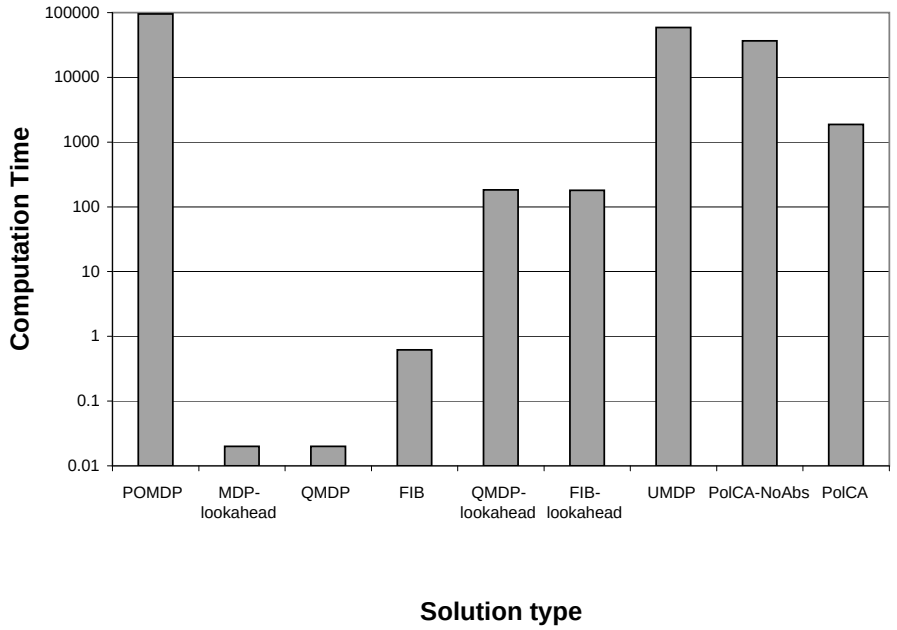


Figure 10: Computation time for solving the cheese-taxi task. Both the UMDP and POMDP algorithms were terminated after many hours of computation, before either had converged to a solution. The *Root* and *Put* subtasks in both PolCA solutions (PolCA and PolCA-NoAbs) were also terminated before convergence. In all cases, the intermediate solution from the last completed iteration was used to evaluate the algorithm and generate the results of Figure 9.

reach $S10$. The other three approximate algorithms ($\text{MDP}_{lookahead}$, QMDP and FIB) are unable to correctly trade-off progress towards goal versus localization information, and consequently require extra steps to reach $S10$. Finally the exact POMDP algorithm, though theoretically able to find the shortest action sequence, would require longer computation to do so. It was terminated after over 24hrs of computation, having completed only 5 iterations of exact POMDP solving.

In this domain, the computation time for PolCA, even with abstraction, is significantly longer than with any of the other approximate algorithms (except UMDP), and the benefits in terms of policy performance over two of the approximations ($\text{QMDP}_{lookahead}$ and $\text{FIB}_{lookahead}$) are only marginal. We conclude that the PolCA algorithm is able to perform on par with other approximate algorithms in this domain, however the longer computation time required does not necessarily make it an appropriate choice for such a domain. As in the part-painting problem, $\text{QMDP}_{lookahead}$ and $\text{FIB}_{lookahead}$ perform quite well in large part because this problem requires only a single information-gathering action. We now consider a domain where this level of approximation is insufficient because a good policy can only be achieved through multi-step information-gathering actions.

7.3 Simulation Domain 3: Twenty-questions game

This new POMDP domain is based on an interactive game called *Twenty-questions* (Burgener, 2002), also know as *"Animal, Vegetable, or Mineral?"*. In this two-player game, the first player selects a specific object in his/her mind, and the second player must then guess what that object is. The second player is allowed to ask a series of {*yes/no*} questions, which the other person must answer truthfully (e.g. *Is it an animal? Is it green? Is it a turtle?*). The second player wins a round if s/he correctly identifies the object within twenty questions (thus the name of the game.)

When modeling the game as a POMDP, the goal is to learn a POMDP policy that correctly guesses the object selected by the user. We represent each possible object as a state. The action space involves two types of actions: *guesses* and *questions*. There is one *guess* per object in the state space (e.g. *Is is a turtle?*). The list of *questions* should be sufficient to disambiguate between state-objects (e.g. *Is is green? Is is it a fruit? Is it a mineral?*). The observation space contains only three items: {*yes, no, noise*}, corresponding to possible verbal responses from the non-POMDP player having picked the object. This POMDP domain can easily be scaled by adding more objects: each new object automatically adds one state and one action, and information-eliciting questions can also be added as necessary. This example is a prototypical information-contingent POMDP, characterized by a large action space (relative to the state space) which includes a variety of information-gathering actions.

With respect to model parameterization, following the natural rules of the game the state transitions should all be self-transitions. However, we add a small probability of randomly transitioning from the current state-object to another one, thus allowing the first player to change his/her mind about the target object. Though not traditionally part of this game, adding stochasticity in the state transitions makes this a much more interesting POMDP. We assume that after each question, the state stays the same with probability 0.9, and randomly changes to any of the other states with cumulative probability 0.1. The

reward is consistently -1 for all *question*-actions, whereas *guess*-actions yield a $+5$ reward if the guess is correct and a -20 reward otherwise. The task is reset every time the policy selects a *guess*-action. Finally, the observation probabilities for each *question*-action noisily reflects the state, for example:

$$\begin{aligned} P(o = yes \mid s = turtle, a = green) &:= 0.85 \\ P(o = no \mid s = turtle, a = green) &:= 0.1 \\ P(o = noise \mid s = turtle, a = green) &:= 0.05 \end{aligned}$$

We implemented a 12-object version of this domain. The POMDP representation contains 12 states (one per object), 20 actions (12 guesses + 8 questions), and 3 observations (*yes, no, noise*). We considered two alternate hierarchical decompositions for this domain. Figure 11 illustrates the first decomposition (referred to as D1). In this case, the domain is decomposed into four subtasks, with some action redundancy between subtasks. Preliminary experiments with this decomposition quickly showed that most of the computation necessary to apply hierarchical planning was spent in solving subtask $h_{vegetable}$.[7] We therefore proposed the second decomposition (referred to as D2), which is illustrated in Figure 12. This decomposition further partitions the action space of the $h_{vegetable}$ subtask, to produce two new lower-level subtasks: $h_{real\text{-}vegetable}$ and h_{fruit}.

We applied our hierarchical planning algorithm twice, once for each decomposition, and also generated policies using the full set of algorithms. For this domain, the performance of each policy was evaluated in simulation using 1000 independent trials. Trials failing to make a *guess* after 100 time steps were terminated.

Figure 13 shows the sum of rewards for each run, averaged over the 1000 trials and plotted as a function of the number of value iteration updates completed (in the case of the hierarchical planners, the full number of iterations was completed for each subtask.) These results clearly illustrate the failures of the QMDP and FIB algorithms when faced with an information-contingent POMDP domain. Looking closely at the policies generated by the QMDP and FIB algorithms, we noted that they are unable to differentiate between the various *question*-actions, and therefore randomly select questions until the belief is sufficiently certain to make a guess. This certainty threshold is slightly lower for the FIB algorithm, thus explaining its slightly less dismal performance. The QMDP algorithm on the other hand was never able to take a correct guess, and in each trial spent 100 time steps asking random questions without any useful effect. As expected, the performance of the exact POMDP solver (in terms of accumulated reward) exceeds that of the approximate methods. For the hierarchical approach, both D1 and D2 converge within approximately 20 iterations, but converge to slightly sub-optimal policies. Furthermore, we note that the additional structural assumptions in D2 cause a greater loss of performance, compared to D1.

Figure 14 presents the same results as in Figure 13, but now plotting the reward performance as a function of computation time. This graph clearly shows the computational savings—note the *log(time)* x-axis—obtained through the use of hierarchical structural assumptions. By comparing D1 and D2 we can also see the trade-off resulting from different

7. It is a convention of this game to let all plant-related objects be identified as "vegetables".

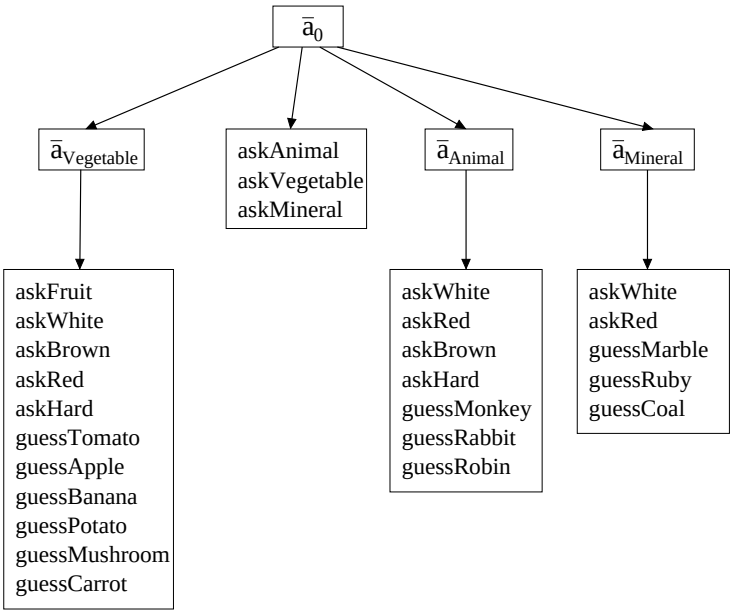


Figure 11: Action hierarchy D1 for twenty-questions domain.

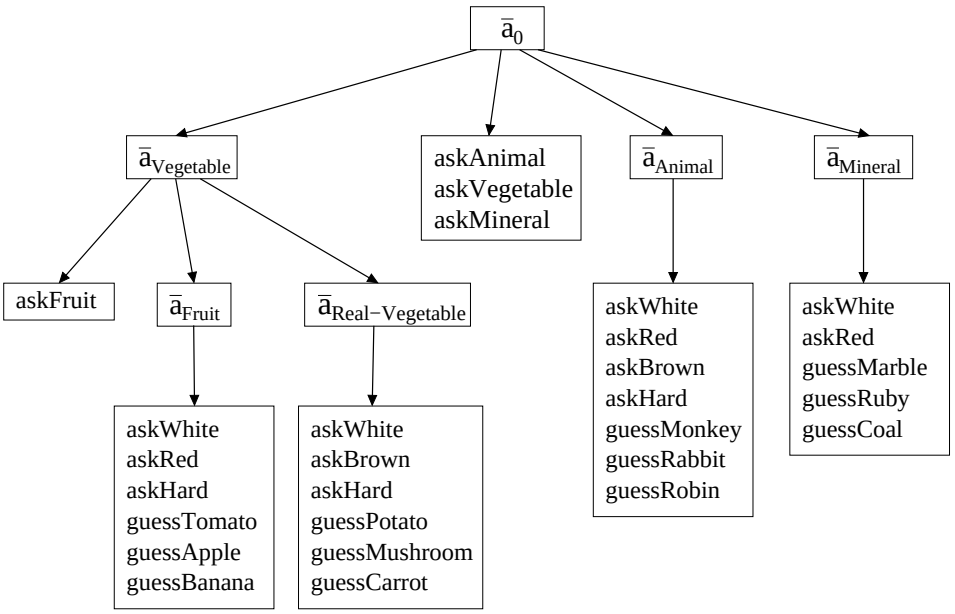


Figure 12: Action hierarchy D2 for twenty-questions domain.

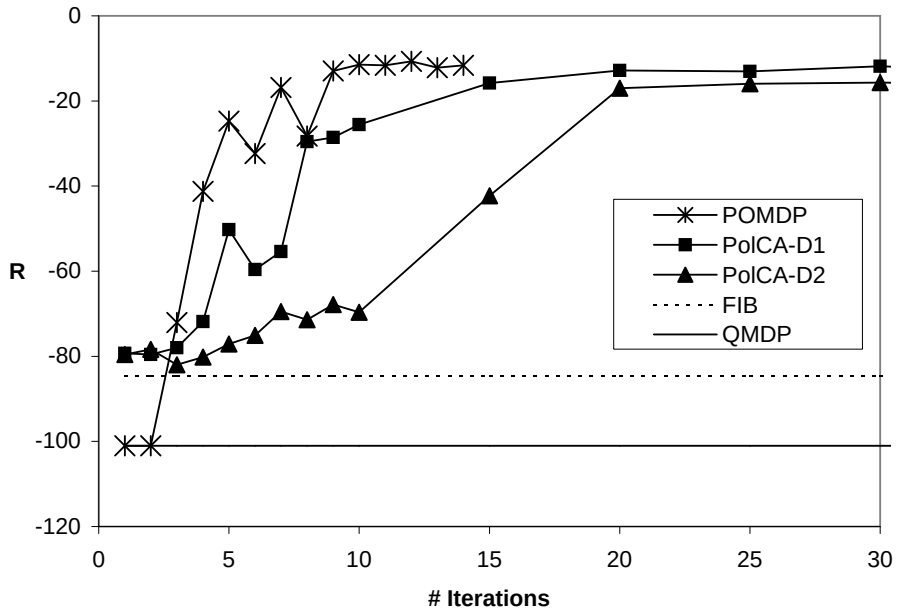


Figure 13: Simulation results for the twenty-questions domain. The QMDP and FIB results are plotted as constants, representing optimized performance. The $\text{MDP}_{lookahead}$, $\text{QMDP}_{lookahead}$, and $\text{FIB}_{lookahead}$ are not illustrated on this figure; all three generated results equivalent to the QMDP performance.

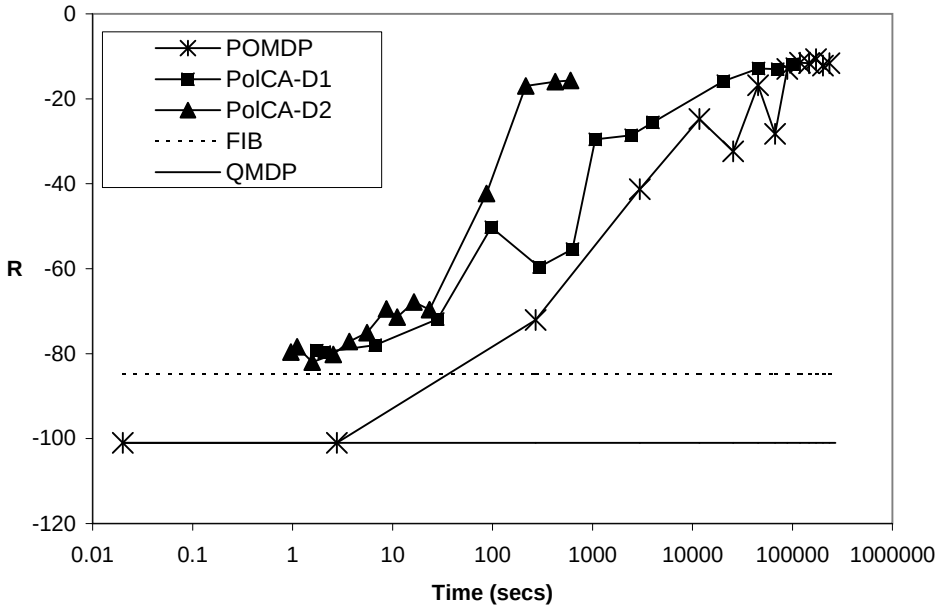


Figure 14: Simulation results as a function of computation time for the twenty-questions domain. All POMDP computations, including for hierarchical subtasks, assumed a pruning criterion of $\epsilon = 0.5$.

structural assumptions. We conclude that PolCA's hierarchical decomposition preserves sufficient richness in representation to successfully address information-contingent POMDPs. Furthermore, through the design of the hierarchy, one can effectively control the trade-off between performance and computation. Other possible approaches to solve this problem which we have not investigated include the even-odd POMDP (Bayer Zubek & Dietterich, 2000) and decision trees (Quinlan, 1986). However the stochasticity in state transitions make decision trees a poor choice for this specific formulation of the 20-questions domain.

We are particularly interested in this domain because it shares many commonalities with POMDP-based dialogue modeling (Roy, Pineau, & Thrun, 2000). When modeling a robot interaction manager as a POMDP, as we do in the next section, the inclusion of information-gathering actions is crucial to a good policy, since human-robot interactions are typically marred by ambiguities, errors and noise. This game in essence provides us with a stylized (and naturally scalable) version of an interaction task, thereby allowing us to perform comparative analyses in simulation, before moving to a real-world implementation. This is an extremely valuable tool given the difficulty of staging real human-robot dialogue experiments. For these reasons, we believe that this domain can be to information-contingent POMDPs what the often-used maze navigation task has been to goal-driven POMDPs, in particular the robot navigation domain.

8. A Real-World Application Domain

In this section, we follow up on our simulation experiments with a real-world implementation of the PolCA approach in the context of a robot behavioral manager. High-level robot control has been a popular topic in AI, and decades of research have led to a reputable collection of architectures (Arkin, 1998; Brooks, 1985; Gat, 1996). However, existing architectures rarely take uncertainty into account during planning. We propose a robot control architecture where a POMDP incorporates high-level uncertainty obtained through both navigation sensors (e.g. laser range-finder) and interaction sensors (e.g. speech recognition and touchscreen) to performs high-level control. The POMDP is also able to arbitrate between information gathering and performance-related actions, as well as negotiate over goals from different specialized modules. Unfortunately, POMDPs of the size necessary for good robot control are an order of magnitude larger than exact POMDP algorithms can tackle. This domain, however, offers significant structure in the form of multiple alternate goals as well as decomposable goals, thus making it a good candidate for PolCA.

In this domain, the POMDP policy is used to control the high-level behavior of a robot in the context of an interactive task. This work was conducted as part of a larger project dedicated to the development of a prototype nursing assistant robot (Montemerlo et al., 2002; Pollack et al., 2002; Pineau & Thrun, 2002). The overall goal of the project is to develop personalized robotic technology that can play an active role in providing improved care and services to non-institutionalized elderly people. The target user is an elderly individual with mild cognitive and/or physical impairment.

The robot Pearl (shown in Figure 15 on the right) is the primary test-bed for the POMDP-based behavior management system. It is a wheeled robot with an on-board speaker system and microphone for speech input and output. It uses the Sphinx II speech recognition system (Ravishankar, 1996) and the Festival speech synthesis system (Black,

Talor, & Caley, 1999). It also has two on-board PCs connected to the Internet via wireless Ethernet.

Figure 15: Pearl, the robotic nursing assistant, interacting with elderly people at a nursing facility.

In this domain, PolCA controls high-level decision-making over a large set of robot activities, both verbal and non-verbal. Typical activities include sending the robot to pre-selected locations, accompanying a person between locations, engaging the person in a conversation, and offering both general information and specific cognitive reminders. This task also involves the integration of multiple robot-based sensors into the POMDP belief state. Current sensors include laser readings, speech recognition, and touch-screen input. These can exhibit significant uncertainty, attributed in large part to poor speech recognition, but also to noisy navigation sensors and erroneous human input.

Table 6 shows a sample dialogue between the robot and a test subject, using a POMDP policy. Because of the uncertainty management in POMDPs, the robot chooses to ask a clarification question at three occasions. The number of such questions depends on the clarity of a person's speech, as detected by the Sphinx speech recognition system. An MDP policy would not exhibit similar adaptability to poor recognition performance.

8.1 POMDP Modeling

To formally test the performance of the PolCA algorithm in this domain, we developed an experimental scenario that tested the effectiveness of the POMDP interaction manager onboard the robot, in the context of a guidance task. This scenario required Pearl to interact with elderly residents, with the primary goal of reminding them of, and taking them to, a scheduled physiotherapy appointment, and with the secondary goal of providing them with interesting information. In the course of this scenario, Pearl had to navigate to

Observation	True State	Action	Reward
pearl hello	request_begun	say_hello	100
pearl what is like	start_meds	**ask_repeat**	−100
pearl what time is it for will the	want_time	say_time	100
pearl was on abc	want_tv	ask_which_station	−1
pearl was on abc	want_abc	say_abc	100
pearl what is on nbc	want_nbc	**confirm_channel_nbc**	−1
pearl yes	want_nbc	say_nbc	100
pearl go to the that pretty good what	send_robot	ask_robot_where	−1
pearl that that hello be	send_robot_bedroom	**confirm_robot_place**	−1
pearl the bedroom any i	send_robot_bedroom	go_to_bedroom	100
pearl go it eight a hello	send_robot	ask_robot_where	−1
pearl the kitchen hello	send_robot_kitchen	go_to_kitchen	100

Table 6: A sample dialogue with a test subject. Actions in bold font are clarification actions, generated by the POMDP because of high uncertainty in the speech signal.

a resident's room, establish contact, possibly accompany the person to the physiotherapy center, and eventually return to a recharging station. The task typically also required the robot to answer simple information requests by the test subject, for example providing the time or the weather forecast. Pearl's high-level behavior (including both speech and motion commands) was completely governed by the POMDP interaction manager.

In this scenario the robot interface domain was modeled using 576 states, which are described using a collection of multi-valued state features. Those states were not directly observable by the robot interface manager; however, the robot was able to perceive 18 distinct observations. The state and observation features are listed in Table 7.

Observations were perceived through 4 different modalities; in many cases the listed observations constitute a summary of more complex sensor information. For example, in the case of the laser range-finder, the raw laser data was processed and correlated to a map to determine when the robot had reached a known landmark (e.g. $\rightarrow$ *Laser=robotAtHome*). Similarly, in the case of a user-emitted speech signal, a keyword filter was applied to the output of the speech recognizer (e.g. *"Give me the weather forecast for tomorrow."* $\rightarrow$ *SpeechKeyword=weather*). In general, the speech recognition and touchscreen input were used as redundant sensors to each other, generating very much the same information, but assumed to have a greater degree of reliability when coming from the touchscreen. The *Reminder* observations were received from a high-level intelligent scheduling module. This software component, developed by McCarthy and Pollack (2002) in the context of the Pearl project, reasons temporally about the user and his/her activity, with the goal of issuing appropriately timed cognitive reminders to warn the person of upcoming scheduled events (e.g. need to take medication, doctor's appointment, social activities, etc.).

In response to the observations, the robot could select from 19 distinct actions, falling into three broad categories:

- COMMUNICATE={RemindPhysioAppt, RemindVitamin, UpdateChecklist, CheckPersonPresent, TerminateGuidance, TellTime, TellWeather, ConfirmGuideToPhysio, Ver-

State Features	**Feature values**
RobotLocation	home, room, physio
PersonLocation	room, physio
PersonPresent	yes, no
ReminderGoal	none, physio, vitamin, checklist
MotionGoal	none, toPhysio
InfoGoal	none, wantTime, wantWeather
Observation Features	**Feature values**
Speech	yes, no, time, weather, go, unknown
Touchscreen	t_yes, t_no, t_time, t_weather, t_go
Laser	atRoom, atPhysio, atHome
Reminder	g_none, g_physio, g_vitamin, g_checklist

Table 7: Component description for human-robot interaction scenario

ifyInfoRequest, ConfirmWantTime, ConfirmWantWeather, ConfirmGoHome, ConfirmDone}

- MOVE={GotoPatientRoom, GuideToPhysio, GoHome}
- OTHER={DoNothing, RingDoorBell, RechargeBattery}

Each discrete action enumerated above invoked a well-defined sequence of operations on the part of the robot (E.g. *Give Weather* requires the robot to first look up the forecast using its wireless Ethernet, and then emit *SpeechSynthesis="Tomorrow's weather should be sunny, with a high of 80."*). The actions in the *Communicate* category involved a combination of redundant speech synthesis and touchscreen display, such that the selected information or question was presented to the test subject through both modalities simultaneously. Given the sensory limitations common in our target population, we found the use of redundant audio-visual communication important for both input to, and output from, the robot. The actions in the *Move* category were translated into a sequence of motor commands by a motion planner, which uses dynamic programming to plan a path from the robot's current position to its destination (Roy & Thrun, 2002).

The POMDP model parameters were selected by a designer. The reward structure, also hand-crafted, reflects the relative costs of applying actions in terms of robot resources (e.g. robot motions actions are typically costlier than spoken verification questions), as well as reflecting the appropriateness of the action with respect to the state. For example, we use:

- positive rewards for correctly satisfying a goal, e.g.
  R($a = TerminateGuidance$)=+50
  if $s(MotionGoal = \{toPhysio\}, RobotLocation = \{physio\}, PersonLocation = \{physio\})$
- a large negative rewards for applying an action unnecessarily, e.g.
  R($a = GuidetoPhysio$)=−200
  if $s(MotionGoal = \{none\})$

- a small negative reward for verification questions, e.g.
  R($a = ConfirmGuidetoPhysio$)=-1
  given any state condition

The scenario was implemented and tested using only a single policy, generated by the PolCA approach (Figure 16 shows the action hierarchy used for this domain). The difficulties involved in carrying out experiments with elderly subjects made it prohibitively difficult at this stage to perform a full scale comparative evaluation with alternate POMDP solutions.

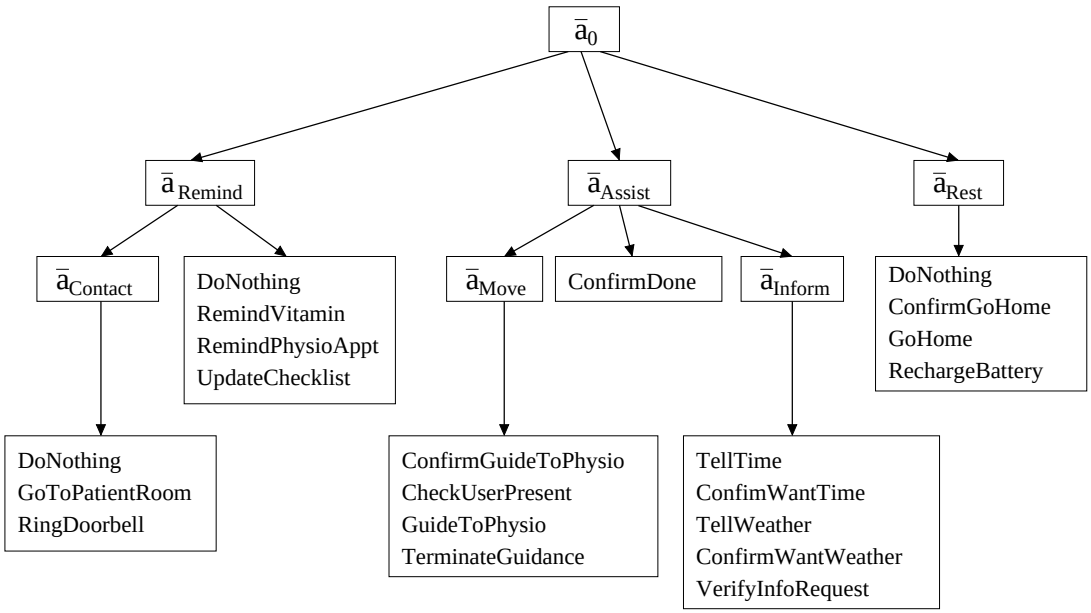


Figure 16: Action hierarchy for robot experiment.

8.2 Experimental Results

We performed two days of formal experiments during which the robot autonomously led 12 full guidances, involving 6 different elderly people. Figure 17 shows an example guidance experiment, involving an elderly person who uses a walking aid. The sequence of images illustrates the major stages of a successful delivery: from contacting the person, explaining to her the reason for the visit, walking her through the facility, and providing information after the successful delivery—in this case on the weather.

Overall, the policy generated using PolCA successfully controlled the robot without any human intervention, in all guidance experiments. The six subjects completed their scenario without difficulty and were very pleased with the experience.[8] Throughout the experiment, speech recognition performance was particularly poor due to the significant amount

8. For video footage see: *http://www.cs.cmu.edu/~thrun/movies/pearl-assist.mpg*

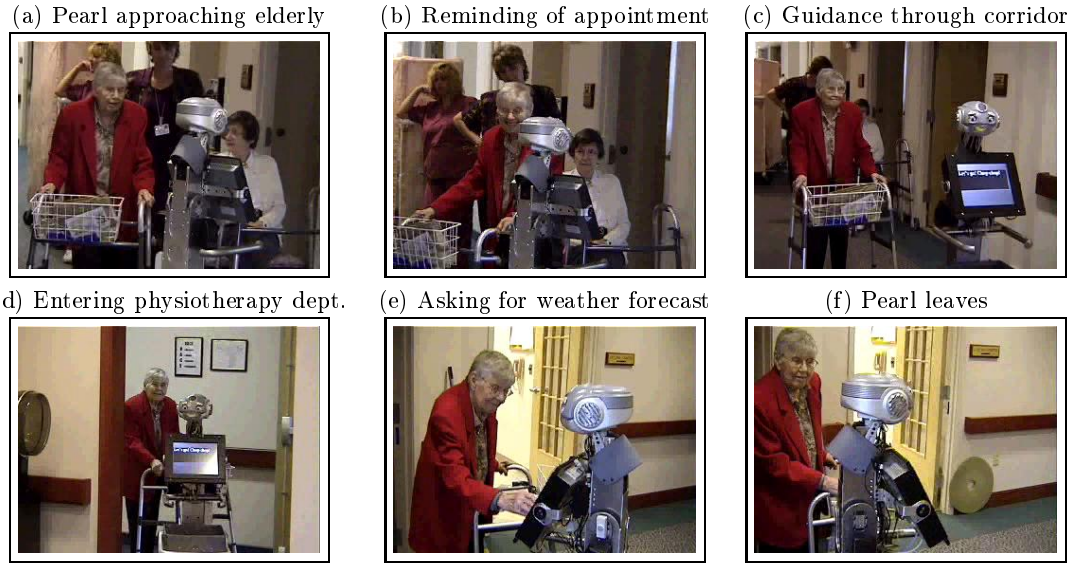


Figure 17: Example of a successful guidance experiment. Pearl picks up the patient outside her room, reminds her of a physiotherapy appointment, walks the person to the department, and responds to a request of the weather report. Throughout this interaction, communication took place through speech and the touch-sensitive display.

of ambient noise, however the redundancy offered by the touch-screen allowed users to communicate with the dialogue manager without difficulty. In addition, in early experiments, the robot lacked the ability to adapt its speed to that of the person, during guidance to the physiotherapy center, causing it to run-away from slow-moving test subjects. This was corrected by the addition of a second laser in the back of the robot, allowing it to adapt its speed appropriately. This experiment constitutes encouraging evidence that, with appropriate approximations, POMDP control can be feasible and useful in real-world robotic applications.

9. Concluding Discussion

This paper has presented a novel approach to solve complex POMDP problems. PolCA appears especially well-suited to information-contingent problems, in which the ability to select from alternate information-gathering actions is paramount to successful planning, and where existing algorithms often fail. The approach has also been successfully used in a robot system for the control of high-level robot behavior, in the context of an interactive service task. Furthermore, in the special case of MDP planning, this approach appears to compare favorably to existing MDP approaches: in some problems it automatically discovered more state abstraction than was possible with previous algorithms. We conclude this paper by discussing manners in which this work can be extended, as well as relations to other work on hierarchical planning.

First, the hierarchical planning algorithm presented in this paper requires having non-trivial local reward functions in each subtask, which in some cases requires the use of a pseudo-reward function. It is possible to loosely divide hierarchical POMDPs into two types of problems: single goal and multi-goal domains. The division into these two types is interesting because it offers some insight into which problems may require pseudo-reward functions. The first type of problems is characterized by having a single large goal which is achieved by sequentially completing relevant subgoals. Each subgoal is generally associated with a subtask. The taxi task (Section 6.1) is an example of such a problem. It is often the case that only the final goal completion is rewarded, and thus other subtasks need to be provided with hand-crafted pseudo-reward functions. In the future, it is possible that work on reward shaping (Ng, Harada, & Russell, 1999) will offer some insight into automatically defining appropriate pseudo-reward functions. In contrast, the second type of problems includes those in which many different alternative goals can be satisfied within a unified framework. The information-contingent POMDP for the 20-questions domain (Section 7.3) is an instance of such a problem. For this type of problems, a good hierarchy will typically partition goals amongst subtasks and thus the local reward assumption can be met without difficulty, and without the need for pseudo-rewards.

Second, the automated state clustering algorithm described in Section 4.2.3 tends to be useful in MDPs only if it can be applied without requiring full enumeration of the state space. This is necessary because otherwise the complexity of the clustering algorithm is equivalent to that of the planning algorithm, and therefore impractical given those large problems for which hierarchical planning is most needed. In general, it is often possible to obtain an ϵ-stable clustering solution without fully enumerating the state space. In the case of POMDPs, the exponential complexity of computing a solution (Equation 13) means that using a clustering algorithm that is polynomial in the size of the domain is by no means prohibitive compared to planning costs. Thus, it is always feasible to compute an $(\epsilon = 0)$-stable clustering of states. Nonetheless, a coarser and approximate clustering may be preferable since it further reduces the size of the problem, and therefore the planning time.

Third, because PolCA uses a monolithic Q-function, instead of a decomposed Q-function (Equation 41), it it unable to achieve full funnel abstraction. This is most relevant in the MDP version of PolCA, where costs can often be effectively decoupled around subtask terminal states. We believe future versions of PolCA can use the decomposed Q-function for still greater savings.

Fourth, the results presented in Figure 10 briefly allude to the fact that some subtasks were not solved to completion. In particular, the results of Figure 9 assume that subtasks *Root* and *Put* were terminated after 9 and 5 iterations respectively. This opens up the question of whether lower-level subtasks should be solved fully (namely to convergence) before higher-level subtasks are solved. For very large domains, it may be better to interleave planning between subtasks of different levels, to ensure an any-time solution. However, this would require re-parameterization of the abstract actions (Equations 31–33) every time the lower-level policy changed. In practice, this overhead will likely be very small compared to the actual POMDP solving applied to each subtask; however, it is probably not advantageous to start computation on high-level tasks too early, when the policies of low-level tasks are extremely poor and change frequently.

Finally, PolCA combines action-decomposition with automated state and observation abstraction to offer a highly-structured approach to POMDP planning. In general, the prevalence of abstraction is a direct result of imposing the hierarchy. We predict that a better understanding of the interaction between action hierarchies and state/observation abstraction may lead to better ways of exploiting structure in problem solving, as well as inspire new methods for automatically discovering action hierarchies

10. Acknowledgments

The authors would like to acknowledge invaluable contributions by the following researchers: David Andre, Tom Dietterich, Geoff Gordon, Michael Littman, and Nicholas Roy.

References

Andre, D., & Russell, S. (2001). Programmable reinforcement learning agents. In Lean, T., Dietterich, T., & Tresp, V. (Eds.), *Advances in Neural Information Processing Systems (NIPS)*, Vol. 13, pp. 1019–1025. MIT Press.

Andre, D., & Russell, S. (2002). State abstraction for programmable reinforcement learning agents. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI)*, pp. 119–125.

Arkin, R. (1998). *Behavior-Based Robotics*. MIT Press.

Asoh, H., Hayamizu, S., Isao, H., Motomura, Y., Akaho, S., & Matsui, T. (1997). Socially embedded learning of office-conversant robot jijo-2. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 880–887.

Bayer Zubek, V., & Dietterich, T. (2000). A POMDP approximation algorithm that anticipates the need to observe. In Springer-Verlag (Ed.), *Proceedings of the Pacific Rim Conference on Artificial Intelligence (PRICAI); Lecture Notes in Computer Science*, pp. 521–532, New York.

Black, A., Talor, P., & Caley, R. (1999). The Festival speech synthesis system. 1.4 edition.

Bonet, B., & Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, *129*, 5–33.

Borenstein, J., Everett, B., & Feng, L. (1996). *Navigating Mobile Robots: Systems and Techniques*. A. K. Peters, Ltd.

Boutilier, C., Brafman, R. I., & Geib, C. (1997). Prioritized goal decomposition of Markov decision processes: Toward a synthesis of classical and decision theoretic planning. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1156–1162.

Boutilier, C., Dearden, R., & Goldszmidt, M. (2000). Stochastic dynamic programming with factored representations. *Artificial Intelligence*, *121*, 49–107.

Boyen, X., & Koller, D. (1998). Tractable inference for complex stochastic processes. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 33–42.

Brafman, R. I. (1997). A heuristic variable grid solution method for pomdps. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI)*, pp. 727–733.

Brooks, R. A. (1985). A robust layered control system for a mobile robot. Tech. rep. TR AI memo 864, MIT.

Burgard, W., Cremers, A. B., Fox, D., Hahnel, D., Lakemeyer, G., Schulz, D., Steiner, W., & Thrun, S. (1999). Experiences with an interactive museum tour-guide robot. *Artificial Intelligence*, *114*, 3–55.

Burgener, R. (2002). Twenty questions: The neural-net on the internet. http://www.20q.net/index.html.

Cassandra, A. (1999). Tony's pomdp-solve page. http://www.cs.brown.edu/research/ai/pomdp/code/index.html.

Cassandra, A., Littman, M. L., & Zhang, N. L. (1997). Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 54–61.

Cox, I. J., & Wilfong, G. T. (Eds.). (1990). *Autonomous Robot Vehicles*. Springer Verlag.

Dayan, P., & Hinton, G. (1993). Feudal reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, Vol. 5, pp. 271–278, San Francisco, CA. Morgan Kaufmann.

Dean, T., & Givan, R. (1997). Model minimization in Markov decision processes. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI)*, pp. 106–111.

Dean, T., Givan, R., & Leach, S. (1997). Model reduction techniques for computing approximately optimal solutions for Markov decision processes. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 124–131.

Dean, T., & Lin, S. H. (1995). Decomposition techniques for planning in stochastic domains. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1121–1129.

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

Dietterich, T. G. (2000b). An overview of MAXQ hierarchical reinforcement learning. In Choueiry, B. Y., & Walsh, T. (Eds.), *Proceedings of the Symposium on Abstraction, Reformulation and Approximation (SARA), Lecture Notes in Artificial Intelligence*, pp. 26–44, New York. Springer Verlag.

Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *2*, 189–208.

Gat, E. (1996). Esl: A language for supporting robust plan execution in embedded autonomous agents. AAAI Fall Symposium: Issues in Plan Execution.

Hauskrecht, M. (2000). Value-function approximations for partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, *13*, 33–94.

Hengst, B. (2002). Learning hierarchical decomposition for factored MDPs. In *Machine Learning: Proceedings of the 2002 International Conference (ICML)*.

Hernandez-Gardiol, N., & Mahadevan, S. (2001). Hierarchical memory-based reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, Vol. 13, pp. 1047–1053.

Jazwinski, A. M. (1970). *Stochastic Processes and Filtering Theory*. Academic, New York.

Kaelbling, L. P. (1993). Hierarchical reinforcement learning: Preliminary results. In *Machine Learning: Proceedings of the 1993 International Conference (ICML)*, pp. 167–173.

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

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME, Journal of Basic Engineering*, *82*, 35–45.

Kushmerick, N., Hanks, S., & Weld, D. (1995). An algorithm for probabilistic planning. *Artificial Intelligence*, *76*, 239–286.

Littman, M. L., Cassandra, A. R., & Kaelbling, L. P. (1995). Learning policies for partially obsevable environments: Scaling up. In *Proceedings of Twelfth International Conference on Machine Learning*, pp. 362–370.

Lovejoy, W. S. (1991). A survey of algorithmic methods for partially observable Markov decision processes. *Annals of Operations Research*, *28*, 47–66.

McCallum, R. A. (1993). Overcoming incomplete perception with utile distinction memory. In *Machine Learning: Proceedings of the 1993 International Conference (ICML)*, pp. 190–196.

McCarthy, C. E., & Pollack, M. (2002). A plan-based personalized cognitive orthotic. In *Proceedings of the 6th International Conference on AI Planning & Scheduling (AIPS)*.

McGovern, A., & Barto, A. G. (2001). Automatic discovery of subgoals in reinforcement learning using diverse density. In *Machine Learning: Proceedings of the 2001 International Conference (ICML)*, pp. 361–368.

Meuleau, N., Hauskrecht, M., Kim, K.-E., Peshkin, L., Kaelbling, L. P., Dean, T., & Boutilier, C. (1998). Solving very large weakly coupled Markov decision processes. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI)*, pp. 165–172.

Monahan, G. E. (1982). A survey of partially observable Markov decision processes: Theory, models, and algorithms. *Management Science*, *28*(1), 1–16.

Montemerlo, M., Pineau, J., Roy, N., Thrun, S., & Verma, V. (2002). Experients with a mobile robotic guide for the elderly. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI)*, pp. 587–592.

Moore, A. W., & Atkeson, C. G. (1995). The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, *21*, 199–233.

Ng, A. Y., Harada, D., & Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Machine Learning: Proceedings of the 1999 International Conference (ICML)*, pp. 278–287.

Parr, R., & Russell, S. (1995). Approximating optimal policies for partially observable stochastic domains. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1088–1094, Montreal, Quebec. Morgan Kauffmann.

Parr, R., & Russell, S. (1998). Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems (NIPS)*, Vol. 10, pp. 1043–1049.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann.

Pickett, M., & Barto, A. G. (2002). Policyblock: An algorithm for creating useful macro-actions in reinforcement learning. In *Machine Learning: Proceedings of the 2002 International Conference (ICML)*.

Pineau, J., & Thrun, S. (2002). High-leel robot behavior control using pomdps. In *Workshop on Cognitive Robotics (CogRob), National Conference on Artificial Intelligence (AAAI)*, pp. 80–87.

Pollack, M., Engberg, S., Matthews, J. T., Thrun, S., Brown, L., Colbry, D., Orosz, C., Peintner, B., Ramakrishnan, S., Dunbar-Jacob, J., McCarthy, C., Montemerlo, M., Pineau, J., & Roy, N. (2002). Pearl: A mobile robotic assistant for the elderly. In *Workshop on Automation as Caregiver: the Role of Intelligent Technology in Elder Care, National Conference on Artificial Intelligence (AAAI)*, pp. 85–91.

Poupart, P., & Boutilier, C. (2000). Value-directed belief state approximation for POMDPs. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 409–416.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*, 81–106.

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, *77*(2), 257–285.

Ravishankar, M. (1996). *Efficient Algorithms for Speech Recognition*. Ph.D. thesis, School of Computer Science, Carnegie Mellon University.

Roy, N., Pineau, J., & Thrun, S. (2000). Spoken dialog management for robots. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Roy, N., & Thrun, S. (2000). Coastal navigation with mobile robots. In *Advances in Neural Information Processing Systems (NIPS)*, Vol. 12, pp. 1043–1049.

Roy, N., & Thrun, S. (2002). Robot navigation using policy search. In *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Russell, S., & Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Ryan, M. (2002). Using abstract models of behavious to automatically generate reinforcement learning hierarchies. In *Machine Learning: Proceedings of the 2002 International Conference (ICML)*.

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

Singh, S., & Cohn, D. (1998). How to dynamically merge Markov decision processes. In *Advances in Neural Information Processing Systems (NIPS)*, Vol. 10, pp. 1057–1063.

Sondik, E. J. (1971). *The Optimal Control of Partially Observable Markov Processes*. Ph.D. thesis, Stanford University.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, *3*, 9–44.

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

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM*, *38*(3), 58–68.

Theocharous, G., Rohanimanesh, K., & Mahadevan, S. (2001). Learning hierarchical partially observable Markov decision process models for robot navigation. In *Proceedings of the 2001 IEEE International Conference on Robotics & Automation (ICRA)*, pp. 511–516.

Thrun, S. (2000). Monte Carlo POMDPs. In *Advances in Neural Information Processing Systems (NIPS)*, Vol. 12, pp. 1064–1070.

Thrun, S., & Schwartz, A. (1995). Finding structure in reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, Vol. 7, pp. 1064–1070.

Torrance, M. C. (1994). Natural communication with robots. Master's thesis, Department of Electrical and Computer Science, MIT.

Wang, G., & Mahadevan, S. (1999). Hierarchical optimization of policy-coupled semi-Markov decision processes. In *Machine Learning: Proceedings of the 1999 International Conference (ICML)*, pp. 464–473.

White, C. C. (1991). A survey of solution techniques for the partially observed Markov decision process. *Annals of Operations Research*, *32*, 215–230.

Wiering, M., & Schmidhuber, J. (1997). HQ-learning. *Adaptive Behavior*, *6*(2), 219–246.

Zinkevich, M., & Balch, T. (2001). Symmetry in Markov decision processes and its implications for single agent and multiagent learning. In *Machine Learning: Proceedings of the 2001 International Conference (ICML)*, pp. 632–639.

Appendix A - Convergence under POMDP state/observation abstraction

This appendix contains the proofs for Theorems 2 and 3 of Section 5.4.

Theorem 2: Optimality for state abstraction in POMDPs. *Let* $M = \{S, A, \Omega, b_0, T, O, R\}$ *be a POMDP. Then, the state minimization algorithm of Section 5.2.3 preserves sufficient information to learn* π^**, the optimal policy for* M*.*

Proof: We consider two states s_i and s_j, with matching cluster assignment: $c = z(s_i) = z(s_j)$ obtained by the POMDP state clustering algorithm of Section 5.2.3. We use a proof by induction to show that any two beliefs $b = \{b_0, ..., b_i, b_j, ...\}$ and $b' = \{b_0, ..., b'_i, b'_j, ...\}$ that differ only in their probability over states s_i and s_j, have identical value $V(b) = V(b')$.

First we consider the value at time $t = 0$:

$$V_0(b) = \max_{a \in A} \left[b(s_i)R(s_i, a) + b(s_j)R(s_j, a) + \sum_{s \in S, s \neq \{s_i, s_j\}} b(s)R(s, a) \right] \quad (47)$$

$$V_0(b') = \max_{a \in A} \left[b'(s_i)R(s_i, a) + b'(s_j)R(s_j, a) + \sum_{s \in S, s \neq \{s_i, s_j\}} b(s)R(s, a) \right]. \quad (48)$$

Assuming that $z(s_i) = z(s_j)$, then by Equation 18 we can substitute $R(s_j) \leftarrow R(s_i)$ in Equation 48:

$$V_0(b') = \max_{a \in A} \left[(b'(s_i) + b'(s_j)) R(s_i, a) + \sum_{s \in S, s \neq \{s_i, s_j\}} b(s)R(s, a) \right] \quad (49)$$

And, because $\sum_{s \in S} b(s) = 1$, we can substitute $(b'(s_i) + b'(s_j)) \leftarrow (b(s_i) + b(s_j))$ in Equation 49:

$$V_0(b') = \max_{a \in A} \left[(b(s_i) + b(s_j)) R(s_i, a) + \sum_{s \in S, s \neq \{s_i, s_j\}} b(s)R(s, a) \right], \quad (50)$$

leading to the conclusion that:

$$V_0(b) = V_0(b'). \quad (51)$$

Next, we assume that the values at time $t - 1$ are equal:

$$V_{t-1}(b) = V_{t-i}(b'). \quad (52)$$

Finally, we must show that the values at time t are equal:

$$V_t(b_t) = \max_{a \in A} \left[\sum_{s \in S} b_t(s)R(s, a) + \gamma \sum_{o \in \Omega} Pr(o \mid a, b_t)V_{t-1}(b_{t-1}) \right] \quad (53)$$

$$V_t(b'_t) = \max_{a \in A} \left[\sum_{s \in S} b'_t(s)R(s, a) + \gamma \sum_{o \in \Omega} Pr(o \mid a, b'_t)V_{t-1}(b'_{t-1}) \right]. \quad (54)$$

Using Equation 51, we can substitute: $\sum_{s \in S} b'_t(s)R(s, a) \leftarrow \sum_{s \in S} b_t(s)R(s, a)$ in Equation 54:

$$V_t(b'_t) = \max_{a \in A} \left[\sum_{s \in S} b_t(s)R(s, a) + \gamma \sum_{o \in \Omega} Pr(o \mid a, b'_t)V_{t-1}(b'_{t-1}) \right]. \quad (55)$$

Next, we use the POMDP stability criterion (Equation 34) in conjunction with Equation 52 and the belief update equation (Equation 6) to infer that $b'_{t-1} = b_{t-1}$, conditioned on each observation $o \in \Omega$, and therefore:

$$V_t(b'_t) = \max_{a \in A} \left[\sum_{s \in S} b_t(s)R(s, a) + \gamma \sum_{o \in \Omega} Pr(o \mid a, b_t)V_{t-1}(b_{t-1}) \right], \quad (56)$$

leading to the conclusion that $V_t(b) = V_t(b')$. **Q.E.D.**

Theorem 3: Optimality for observation abstraction in POMDPs. *Let $M = \{S, A, \Omega, b_0, T, O, R\}$ be a POMDP. Then, the observation minimization algorithm of Section 5.2.4 preserves sufficient information to learn π^*, the optimal policy for M.*

Proof: We consider an observation $o \in \Omega$ which satisfies Equation 35 and thus is excluded from a given set Ω_h^a. We consider a second set $\Omega_h^{a+} = \{\Omega_h^a, o\}$. We consider two POMDP subtasks which are identical in every way except that the first uses Ω_h^a and the second uses Ω_h^{a+}. We show that solving both generates the same solution.[9] Let V^{h*} be the solution obtained when using Ω_h^a and V^{h+} be the solution obtained when using Ω_h^{a+}.

Using a proof by induction, we first consider:

$$\begin{aligned} V_0^{h*}(b) &= \max_{a \in A_h} \left[\sum_{s \in S} b_0(s) R(s,a) \right] \\ V_0^{h+}(b) &= \max_{a \in A_h} \left[\sum_{s \in S} b_0(s) R(s,a) \right] \end{aligned} \tag{57}$$

from which we can conclude that:

$$V_0^{h*}(b) = V_0^{h+}(b). \tag{58}$$

We now assume that:

$$V_{t-1}^{h*}(b) = V_{t-1}^{h+}(b). \tag{59}$$

Next, we consider the exact POMDP value update equation of section 7:

$$V_t^{h*}(b) = \max_{a \in A} \left[\sum_{s \in S} b(s) R(s,a) + \gamma \sum_{o \in \Omega_h^{a*}} Pr(o \mid a, b) V_{t-1}^{h*}(b'_o) \right] \tag{60}$$

$$V_t^{h+}(b) = \max_{a \in A} \left[\sum_{s \in S} b(s) R(s,a) + \gamma \sum_{o \in \Omega_h^{a+}} Pr(o \mid a, b) V_{t-1}^{h+}(b'_o) \right]. \tag{61}$$

Substituting Equation 59 into 60 and re-arranging, we get:

$$\begin{aligned} V_t^{h*}(b) = \max_{a \in A} \Bigg[& \sum_{s \in S} b(s) R(s,a) + \gamma \sum_{o \in \Omega_h^{a+}} Pr(o \mid a, b) V_{t-1}^{h+}(b'_o) \\ & + \gamma Pr(o \mid a, b) V_{t-1}^{h+}(b'_o) \Big]. \end{aligned} \tag{62}$$

9. In the interest of clarity, we assume no state abstraction (i.e. $S_h = S$), however extending the proof to the case with state abstraction is trivial.

Expanding the last term:

$$V_t^{h*}(b) = \max_{a \in A} \left[\sum_{s \in S} b(s) R(s,a) + \gamma \sum_{o \in \Omega_h^{a+}} Pr(o \mid a, b) V_{t-1}^{h+}(b'_o) + \gamma V_{t-1}^{h+}(b'_o) \sum_{s \in S} Pr(o \mid a, s) b(s) \right], \quad (63)$$

we can then eliminate the last term using the observation abstraction condition in Equation 35, such that:

$$V_t^{h*}(b) = \max_{a \in A} \left[\sum_{s \in S} b(s) R(s,a) + \gamma \sum_{o \in \Omega_h^{a+}} Pr(o \mid a, b) V_{t-1}^{h+}(b'_o) \right], \quad (64)$$

and therefore:

$$V_t^{h*}(b) = V_t^{h+}(b). \quad (65)$$

We conclude that no loss of performance results from eliminating an observation o which satisfies Equation 35. **Q.E.D.**