TRACTABLE PLANNING UNDER UNCERTAINTY: EXPLOITING STRUCTURE

Joelle Pineau

CMU-RI-TR-04-32

Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

August 2004

Submitted in partial fulfilment of
the requirements for the degree of
Doctor of Philosophy

Thesis Committee:
Geoffrey Gordon, Co-Chair
Sebastian Thrun, Co-Chair
Matthew Mason
Andrew Moore
Craig Boutilier, University of Toronto
Michael Littman, Rutgers University

ABSTRACT

THE problem of planning under uncertainty has received significant attention in the scientific community over the past few years. It is now well-recognized that considering uncertainty during planning and decision-making is imperative to the design of robust computer systems. This is particularly crucial in robotics, where the ability to interact effectively with real-world environments is a prerequisite for success.

The Partially Observable Markov Decision Process (POMDP) provides a rich framework for planning under uncertainty. The POMDP model can optimize sequences of actions which are robust to sensor noise, missing information, occlusion, as well as imprecise actuators. While the model is sufficiently rich to address most robotic planning problems, exact solutions are generally intractable for all but the smallest problems.

This thesis argues that large POMDP problems can be solved by exploiting natural structural constraints. In support of this, we propose two distinct but complementary algorithms which overcome tractability issues in POMDP planning. PBVI is a sample-based approach which approximates a value function solution by planning over a small number of salient information states. PolCA+ is a hierarchical approach which leverages structural properties of a problem to decompose it into a set of smaller, easy-to-solve, problems. These techniques improve the tractability of POMDP planning to the point where POMDP-based robot controllers are a reality. This is demonstrated through the successful deployment of a nursing assistant robot.

ACKNOWLEDGMENTS

This thesis is the product of many years of enjoyable and productive collaboration with my advisors, Geoff Gordon and Sebastian Thrun. I thank them for generously sharing their talents, energy, and good advice.

I am grateful to all members of the Robot Learning Lab with whom I shared a steady regimen of weekly meetings and memorable annual retreats. I was especially lucky to have the collaboration and friendship of Michael Montemerlo and Nicholas Roy. It is a testimony to their good will and hard work that this thesis features any robots at all.

My thanks to Craig Boutilier, Michael Littman, Matthew Mason, Andrew Moore and Martha Pollack for many insightful exchanges and discussions. Their technical and professional support has been invaluable.

Many thanks to Jean Harpley, Suzanne Lyons Muth and Sharon Woodside for their amazing dedication and resourcefulness.

I thank Tony Cassandra for making available his POMDP tutorial, problem repository, and code, which were a tremendous help throughout my research efforts.

My thanks to the wonderful friends and colleagues which enriched my years at CMU: Drew Bagnell, Curt Bererton, Bernardine Dias, Rosemary Emery-Montermerlo, Ashley Stroupe, Vandi Verma, Carl Wellington and Jay Wylie.

Finally, I thank my family, especially Aaron and Sophie, for their constant support and affection.

TABLE OF CONTENTS

Bibliography . . . 128

LIST OF FIGURES

LIST OF TABLES

NOTATION

- A : the action set
- a_t : the action at time t
- S : the state set
- s_t : the state at time t
- Z : the observation set
- z_t : the observation at time t

- $O()$: the observation emission probability function
- $T()$: the state-to-state transition probability function
- $R()$: the reward function
- r_t : the reward at time t

- γ : the discount factor
- $V()$: the value function
- $V_t()$: the value function at time t
- $Q()$: the MDP value for applying action a in state s
- π : the policy

- Δ : the belief simplex
- $\bar{\Delta}$: the set of all reachable beliefs
- B : a set of belief points
- b_t : the belief at time t
- $\tau()$: the belief update function

- α : an S-dimensional value function hyper-plane
- Γ : the set of α hyper-planes
- Γ_t : the set of hyper-planes sufficient to represent the value function V_t
- $\oplus$: the cross-sum operator, e. g. $\{a, b, \dots\} \oplus \{p, q, \dots\} = \{a{+}p, a{+}q, b{+}p, b{+}q, \dots\}$

- H : the task hierarchy
- h : a subtask
- f_h : a function mapping states to clusters of states
- S_h : a set of state clusters specific to subtask h
- c : a cluster of states
- g_h^a : a function mapping observations to clusters of observations
- Z_h^a : a set of observation clusters specific to subtask h and action a
- k : a cluster of observations

CHAPTER 1

Introduction

THE concept of planning is at the core of many AI and robotics problems. Planning requires a person, a system, or a robot to select a sequence of actions with the goal of satisfying a task. Automatic planning is generally viewed as an essential part of an autonomous system, be it a software agent, expert system, or mobile robot.

In the early days of AI, planning was restricted to simple tasks in static environments; actions had few and predictable effects, and could be combined sequentially to satisfy the desired goal. This gave rise to a rich and successful set of approaches that could handle planning problems of increasing complexity, including the ability to satisfy multiple goals, handle time constraints, quickly re-plan, and so on. However, these methods generally relied on the assumption that the true state of the world (or a sufficient statistic thereof) could be sensed exactly and reliably.

While this assumption is reasonable in some highly-structured domains, this is clearly not the case in many real-world problems. For example, significant research on natural language dialogue systems has sought to devise techniques for recovering state information through conversing with a person. Similarly, in robotics, sensor limitations are pervasive and the seemingly simple problem of recovering the state from sensor measurements is the key subject of research in entire research programs.

Furthermore, as robots move into human-centered living and working environments, they will face increasingly diverse and changing environments. These environments, because they are meant first and foremost for human occupants, cannot and should not be constrained and modified to accommodate robots which need to know everything about the state of the world at all times. Rather, it is the robots that need to adapt and develop the ability to handle the uncertain and the dynamic nature of their environments.

But it is not sufficient for robots to only detect and track uncertainty. Consider the case of a personal assistant robot, which interacts with a user through natural speech. Given the state of speech recognition technology, the robot should expect a certain amount of noise in its detection of user utterances. While there are clear benefits for the robot to model and reason about the uncertainty in the speech signal, what is crucial is for the robot to *act* on this uncertainty, namely to decide when to answer a query, and when to seek clarification, or solicit feedback.

Robots require the ability to formulate plans with appropriate contingencies for the frequent uncertain situations that are bound to arise. It is those problems, where planning takes into account the fact that the state of the world is only partially measurable, which motivate the research described in this thesis.

The importance of planning in uncertain environments cannot be overstated: the impact of intelligent agents in real-world applications depends directly on their ability to satisfy complex tasks, without unnecessary modification of their environment. This is the measure by which the success of autonomous agents—in particular robots—will be measured, thus the strong impetus for pursuing research on planning under uncertainty.

1.1. Planning under uncertainty

The concept of planning has a long tradition in the AI literature (Russell & Norvig, 2002; Weld, 1999). Classical planning is generally concerned with agents which operate in environments that are fully observable, deterministic, finite, static, and discrete. States and actions are described using propositional (first-order) representations. The STRIPS language (Fikes & Nilsson, 1971) is an early instance of a classical planner. It assumes a known start state and goal state, and actions are described in terms of preconditions and effects. In this context, planning is implemented as a forward (or backward) search through the state space, subject to the preconditions and effects of actions. Scalability of such planning paradigm has been achieved through the appropriate use of partial plan ordering (Chapman, 1987; McAllester & Roseblitt, 1991; Penberthy & Weld, 1992), planning graphs (Blum & Furst, 1997), constraint satisfiability (Kautz & Selman, 1992), and heuristics (Bonet & Geffner, 2001). While these techniques are able to solve increasingly large state-space problems, the basic assumptions of classical planning—full observability, static environment, deterministic actions—make these unsuitable for most robotic applications.

Planning under uncertainty aims to improve robustness by explicitly reasoning about the type of uncertainty that can arise. Conformant planning (Goldman & Boddy, 1996; Akella, Huang, Lynch, & Mason, 1997; Smith & Weld, 1998; Bertoli, Cimatti, & Roveri,

2001) deals with the special case of sensorless environments, where the plan selects actions which coerce the agent into a known state, thus overcoming state uncertainty. Conditional planning uses similar propositional representation as in classical planning, but is able to address some form of uncertainty. Such techniques generate plans where action choices are conditioned on the outcome of sensing actions (Peot & Smith, 1992; Pryor & Collins, 1996). Stochastic action outcomes can also be represented through disjunctive effects and conditional effects (Warren, 1976; Olawsky & Gini, 1990), or through probabilistic effects (Goldman & Boddy, 1994; Draper, Hanks, & Weld, 1994; Blythe, 1998).

The Partially Observable Markov Decision Process (POMDP) (Ästrom, 1965; Sondik, 1971; Monahan, 1982; White, 1991; Lovejoy, 1991b; Kaelbling, Littman, & Cassandra, 1998; Boutilier, Dean, & Hanks, 1999) has emerged as possibly the most general representation for planning under uncertainty. The POMDP supersedes other frameworks in terms of representational power simply because it combines the most essential features for planning under uncertainty.

First, POMDPs handle uncertainty in both *action effects* and *state observability*, whereas many other frameworks handle neither of these, and some handle only stochastic action effects. To handle partial state observability, plans are expressed 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.

Second, many POMDP algorithms form plans by optimizing a *value function*. This is a powerful approach to plan optimization, since it allows one to numerically trade-off between alternative ways to satisfy a goal, compare actions with different costs / rewards, as well as plan for multiple interacting goals. While value function optimization is used in other planning approaches—for example Markov Decision Processes (MDPs) (Bellman, 1957)—POMDPs are unique in expressing the value function over information states, rather than world states.

Finally, whereas classical and conditional planners produce a sequence of actions, POMDPs produce a full *policy* for action selection, which prescribes the choice of action for any possible information state. By producing a universal plan, POMDPs alleviate the need for re-planning, and allow fast execution.

Unfortunately, the fact that POMDPs produce a universal plan, combined with the fact that the space of all information states is much larger than the state space itself, means that

POMDPs are computationally much harder than other approaches. In fact, POMDP planning is PSPACE-complete, whereas propositional planning is only NP-complete. This computational intractability is arguably the most important obstacle toward applying POMDPs successfully in practice.

The main contribution of this thesis is to propose two related approaches—*Point-based value iteration* (PBVI) and *Policy-contingent abstraction* (PolCA+)—which directly tackle complexity issues in POMDP planning, and to demonstrate the impact of these approaches when applied to real-world robot problems.

This thesis exclusively addresses the computational complexity involved in policy generation (planning). We assume that the state spaces at hand are small enough (e. g. 10^4) that the information state can be calculated exactly. We also target domains for which a model of the world's dynamics, sensors, and costs / rewards is available.

1.2. Point-Based Value Iteration

As described above, POMDPs handle uncertainty by expressing plans over information states, also called *beliefs*, instead of world states. Exact planning approaches for POMDPs are designed to optimize the value function over all possible beliefs. In most domains only a subset of beliefs can be reached (assuming a known initial belief). However even the set of reachable beliefs can grow exponentially with the planning horizon. This means that the time / space requirements for computing the exact value function also grow exponentially with the planning horizon. This can quickly become intractable even for problems with only a few states, actions, and sensor observations.

Point-based value iteration (PBVI) is a new algorithm that was designed to address this problem. Instead of learning a value function for *all* belief points, it selects a small set of representative belief points, and iteratively applies value updates to those points only. The point-based update is significantly more efficient than an exact update (quadratic vs. exponential). And because PBVI updates both the value and value gradient, it can generalize fairly well to unexplored beliefs, especially those close to the selected points.

This thesis presents a theoretical analysis of PBVI, which shows that it is guaranteed to have bounded error with respect to the exact value function. While an error bound is generally in and of itself a useful assessment of performance, in the case of PBVI it also provides us with additional insight. In particular, the bound can be used to determine how to best select the number and placement of belief points necessary to find a good solution.

The complete PBVI algorithm is designed as an *anytime* algorithm, interleaving steps of value iteration and steps of belief set expansion. It starts with an initial set of belief

points for which it applies a first series of backup operations. Based on this preliminary solution, it selects new belief points to be added to the set, and finds a better value function based on the expanded set. By interleaving value backup iterations with expansions of the belief set, PBVI offers a range of solutions, gradually trading off computation time and solution quality.

Chapter 3 describes the PBVI algorithm in full detail. It derives and explains the error bound on the algorithm, including showing how it is useful for selecting belief points. Finally, it presents empirical results demonstrating the successful performance of the algorithm on a large (870 states) robot domain called *Tag*, inspired by the game of lasertag. This problem is an order of magnitude larger than other problems previously used to test scalable POMDP algorithms.

PBVI is a promising approximation algorithm for scaling to larger POMDP problems, likely effective to solve problems up to $10^3 - 10^4$ states. However while this may be considered "large" in terms of POMDP problems, it is still a long way from being useful for most real-world robot domains, where planning problems described with a few multi-valued state features can require upward of 10^6 states. This highlights the need to take greater advantage of structural properties when addressing very large planning domains.

1.3. Hierarchical POMDPs

Some of the most successful robot control architectures rely on structural assumptions to tackle large-scale control problems (Brooks, 1986; Arkin, 1998). The Subsumption architecture, for example, uses a combination of hierarchical task partitioning and task-specific state abstraction to produce scalable control systems. However it, and other similar approaches, are not designed to handle state uncertainty, which can have dramatic effects in situations where state estimation is particularly noisy or ambiguous. Furthermore, these approaches typically rely on human designers to specify all structural constraints (hierarchy, abstraction) and in some cases even the policies.

The second algorithm presented in this thesis, named PolCA+ (for **Pol**icy-**C**ontingent **A**bstraction) is a hierarchical decomposition approach specifically designed to handle large structured POMDP problems. PolCA+ uses a human-designed task hierarchy which it traverses from the bottom up, learning a state abstraction function and action-selection policy for each subtask along the way. Though very much in the tradition of earlier structured robot architectures, PolCA+ also leverages techniques from the MDP literature to formalize the hierarchical decomposition, extending these to the partially observable case.

Chapter 4 of this thesis presents two versions of the algorithm. The first, from here on referred to as PolCA, is specifically for MDP-type problems (i. e. assuming full state observability). It is closest to the earlier hierarchical MDP approaches, and is included to allow a thorough comparison with these other algorithms. The second, referred to as PolCA+, is the POMDP version, with full ability to handle partial state observation, which is of utmost importance for real-world problems. Both PolCA and PolCA+ share many similarities with well-known MDP hierarchical algorithms (Dietterich, 2000; Andre & Russell, 2002) in terms of defining subtasks and learning policies. However there are two notable differences, which are essential for addressing robotic problems.

First, to define subtasks, PolCA / PolCA+ uses a human-specified action hierarchy, in combination with subtask-specific automatic state abstraction functions. This requires less information from the human designer than earlier approaches: s / he must specify the action hierarchy, but not the subtask-specific abstraction functions. In many cases, human experts are faster and more accurate at providing hierarchies than they are at providing state abstractions, so PolCA / PolCA+ benefits from faster controller design and deployment.

Second, PolCA / PolCA+ performs *policy-contingent* abstraction: the abstract states at higher levels of the hierarchy are left unspecified until policies at lower levels of the hierarchy are fixed. By contrast, human-designed abstraction functions are usually *policy-agnostic* (correct for all possible policies) and therefore cannot obtain as much abstraction. Humans may sometimes (accidentally or on purpose) incorporate assumptions about policies into their state abstraction functions, but because these are difficult to identify and verify, they can easily cause problems in the final plan.

PolCA+ is the full-featured POMDP version of our hierarchical algorithm. It differs from PolCA in a number of ways necessary to accommodate partial observability, including how subtasks are defined, how they are solved, and how dependencies between them are handled. First, when defining subtasks, the state abstraction must take partial observability into account, and therefore in some cases it is necessary to preserve additional state variables which are subject to ambiguity. This further highlights the importance of automatic state abstraction, since reasoning about which states may or not be confused could be particularly difficult for a human designer.

Second, when defining subtasks, PolCA+ also applies automatic *observation* abstraction. To the best of our knowledge this is new to the POMDP literature (regardless of any hierarchical context), and has important implications for POMDP solving in general since the number of observations is an important factor in the exponential growth of reachable

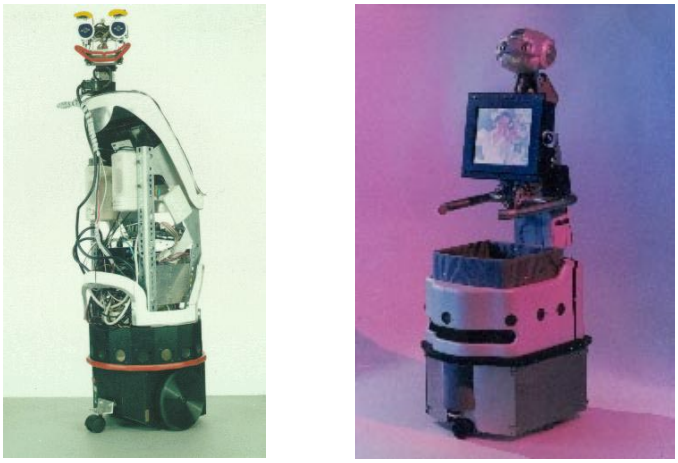
beliefs (as described in Section 1.1). In the context of PolCA+, automatic observation abstraction is useful to discard observations that are irrelevant to some specific tasks. For example, when controlling an interactive robot, a subtask specialized to robot navigation can safely ignore most speech input, since it is unlikely to contribute any useful information to localization and path planning.

When solving subtasks PolCA+ can use any existing (non-structured) POMDP solver. The choice of solver can vary between subtasks, based on their properties (e. g. size, performance requirement, etc.). Ideally, each subtask would be sufficiently small to be solved exactly, but in practice this rarely happens. The PBVI solver described in Chapter 3, which has the ability to handle tasks on the order of 10^3 states, can easily be applied to most subtasks.

Finally, when combining local policies from each subtask in PolCA+ to form a global policy, we must once again take into account partial observability. In this case, the important consideration comes from the fact that we cannot even assume that subtask *completion* is fully observable. This may seem like a small detail, but in practice, it has a profound effect on our execution model. Most hierarchical approaches for fully observable domains proceed with a subtask until it is completed, then return control to the higher-level subtask that invoked it. In the case of PolCA+, the decision to proceed (or not) with a subtask must be re-evaluated periodically, since there are no guarantees that subtask completion will be observed. To accommodate this, we use top-down control at every step (also known as *polling* execution). This means that at *every time step* we first query the policy of the top subtask; if it returns an abstract action we query the policy of that subtask, and so on down the hierarchy until a primitive action is returned. Since policy polling occurs at every time step, a subtask may be interrupted before its subgoal is reached, namely when the parent subtask suddenly selects another action.

Chapter 4 first presents PolCA (the fully observable version), including a full description of the automatic state abstraction, subtask solution, and execution model. We also present results describing the performance of PolCA on standard structured MDP problems, and compare it with that of other hierarchical MDP approaches.

Chapter 4 then presents PolCA+ (the partially observable version), and describes in detail the algorithmic components that perform state abstraction, observation abstraction, subtask solution, and polling execution. The chapter also contains empirical results obtained from applying the algorithm to a series of simulated POMDP problems.



(a) Flo (b) Pearl

Figure 1.1. Nursebot platforms

1.4. Application Domain

The overall motivation behind the work described in this thesis is the desire to provide high-quality robust planning for real-world autonomous systems, and in particular for robots. On a more practical scale, our search for a robust robot controller has been in large part guided by the Nursebot project. The goal of this project is to develop a mobile robot assistant for elderly institutionalized people. Flo (left) and Pearl (right), shown in Figure 1.1, are the main robotic platforms used throughout this project.

The long-term vision is to have a robot permanently based in a human-living environment (personal home or nursing home), where it interacts with one or many elderly individuals suffering from mild cognitive and physical impairments, to help them preserve their autonomy. Key tasks of the robot could include delivering information (reminders of appointments, medications, activities) and guiding people through their environment while interacting in socially appropriate ways. Designing a good robot controller for this domain is critical since the cost of executing the wrong command can be high. Poor action choices can cause the robot to wander off to another location in the middle of a conversation, or cause is to continue issuing reminders even once a medication has been taken. The design of the controller is complicated by the fact that much of the human-robot interaction is speech-driven. While today's recognizers yield high recognition rates for articulate speakers, elderly people often lack clear articulation or the cognitive awareness to place

themselves in an appropriate position for optimal reception. Thus the controller must be robust to high noise levels when inferring, and responding to, users' requests.

Given these characteristics, this task is a prime candidate for robust POMDP planning. However until recently, the computational intractability of POMDP planning would have made it a poor choice of framework to address this new problem. By combining the algorithms described in this thesis, PolCA+ to perform high-level structural decomposition and PBVI to solve subtasks, we are able to address complex dialogue management and robot control problems.

Chapter 5 describes the Nursebot planning domain in terms of the POMDP framework. It discusses the structural properties and assumptions that make it suitable for PolCA+, and show how we have solved the problem using our joint approach. It also presents a sequence of simulation results analyzing the performance of our algorithms on this large-scale planning domain.

As part of the Nursebot project, a POMDP-based high-level robot controller using PolCA+ as its main robot control architecture has been deployed for testing in a nursing home facility. Chapter 5 describes the design and deployment of this system. The results show that the PolCA+ approach produces a planning algorithm capable of performing high-level control of a mobile interactive robot, and as such was a key element for the successful performance of the robot in the experiments with elderly users.

1.5. Thesis Contributions

The contributions of this thesis include both significant algorithmic development in the area of POMDP planning, as well as novel approaches improving robustness to state uncertainty for high-level robot control architectures.

The first algorithmic contribution is the PBVI algorithm, an approximation algorithm for POMDP planning, which can handle problems on the order of 10^3 states. This is an order of magnitude larger than problems typically used to test scalability of POMDP algorithms. The algorithm is widely applicable, since it makes few assumptions about the nature of the domain. Furthermore, because it is an anytime algorithm, it allows an effective trade-off between planning time and solution quality. Finally, a theoretical analysis of the algorithm shows that it has bounded error with respect to the exact value function solution.

The second algorithmic contribution of this thesis is the PolCA+ algorithm, a hierarchical decomposition approach for structured POMDPs. This algorithm extends earlier

hierarchical approaches (MDP and others) to domains with partial state observability, and thus can be expected to have wide impact on large-scale robot control problems.

This thesis goes beyond these algorithmic contributions, and includes an important experimental component, where the algorithms are deployed and evaluated in the context of real-world robot systems. In addition to thoroughly demonstrating the effectiveness of the proposed algorithms on realistic tasks, this is also meaningful in terms of state-of-the-art robot control architectures. Our application of the PolCA+ controller in the context of the Nursebot project provides a first instance of a robot using POMDP techniques at the highest level of robot control to perform a task in a real-world environment.

CHAPTER 2

Partially Observable Markov Decision Processes

PARTIALLY Observable Markov Decision Processes provide a general planning and decision-making framework for acting optimally in partially observable domains. They are well-suited to a great number of real-world problems where decision-making is required despite prevalent uncertainty. They generally assume a complete and correct world model, with stochastic state transitions, imperfect state tracking, and a reward structure, and from that find an optimal way to operate in the world. This chapter first establishes the basic terminology and essential concepts pertaining to POMDPs, and then reviews numerous algorithms—both exact and approximate—that have been proposed to do POMDP planning.

2.1. Review of POMDPs

Formally, a POMDP is characterized by seven distinct quantities, denoted S, A, Z, b_0, T, O, R. The first three of these are:

- *States.* The state of the world is denoted s, with the finite set of all states denoted by $S = \{s_0, s_1, \dots\}$. 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 $Z = \{z_0, z_1, \dots\}$. The observation at time t is denoted z_t. Observation z_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 finite set of actions, denoted $A = \{a_0, a_1, \dots a_k\}$. Actions stochastically 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 renders independent the future from the past (Pearl, 1988). It is commonly assumed that actions and observations are alternated over time. This assumption does not restrict the general expressiveness of the approach, but is adopted throughout for notational convenience.

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

- The *initial state probability distribution*,

$$b_0(s) \quad := \quad Pr(s_0 = s), \tag{2.1}$$

  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*,

$$T(s, a, s') \quad := \quad Pr(s_t = s' \mid s_{t-1} = s, a_{t-1} = a) \ \ \forall t, \tag{2.2}$$

  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*,

$$O(s, a, z) \quad := \quad Pr(z_t = z \mid s_{t-1} = s, a_{t-1} = a) \ \ \forall t, \tag{2.3}$$

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

Finally, the objective of POMDP planning is to optimize action selection, so the agent is given a reward function describing its performance:

- The *reward function*. $R(s, a) : S \times A \longrightarrow \mathbb{R}$, assigns a numerical value quantifying the utility of performing action a when in state s. The goal of the agent is to maximize the sum of its reward over time. Mathematically, this is commonly defined by a sum of the form:

$$E[\sum_{t=t_0}^{T} \gamma^{t-t_0} r_t], \tag{2.4}$$

  where r_t is the reward at time t, $E[\,]$ is the mathematical expectation, and γ where $0 \leq \gamma < 1$ is a *discount factor*, which ensures that the sum in Equation 2.4 is finite.

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

2.1.1. Belief computation

The key characteristic that sets POMDPs apart 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 $\{z_1, \ldots, z_t\}$, which convey incomplete information about the world's state.

Given that the state is not directly observable, the agent can instead maintain a complete trace of all observations and all actions it ever executed, and use this to select its actions. The action / observation trace is known as a *history*. We formally define

$$h_t \quad := \quad \{a_0, z_1, \ldots, z_{t-1}, a_{t-1}, z_t\} \tag{2.5}$$

to be the history at time t.

This history trace can get very long as time goes on. A well-known fact is that this history does not need to be represented explicitly, but can instead be summarized via a *belief distribution* (Ästrom, 1965), which is the following posterior probability distribution:

$$b_t(s) \quad := \quad Pr(s_t = s \mid z_t, a_{t-1}, z_{t-1}, \ldots, a_0). \tag{2.6}$$

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}, along with the most recent action a_{t-1} and observation z_t:

$$\begin{aligned} b_t(s) &= \tau(b_{t-1}, a_{t-1}, z_t) \\ &= \frac{\sum_{s'} O(s', a_{t-1}, z_t)\ T(s, a_{t-1}, s')\ b_{t-1}(s')}{\sum_s \sum_{s'} O(s', a_{t-1}, z_t)\ T(s, a_{t-1}, s')\ b_{t-1}(s')} \end{aligned} \tag{2.7}$$

where the denominator 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), where it is known as the forward algorithm. 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 thesis, the belief is a continuous quantity. It is defined over a simplex describing the space of all distributions over the state space S. For

very large state spaces, calculating the belief update (Eqn 2.7) can be computationally challenging. Recent research has led to efficient techniques for belief state computation that exploit structure of the domain (Dean & Kanazawa, 1988; Boyen & Koller, 1998; Poupart & Boutilier, 2000; Thrun, Fox, Burgard, & Dellaert, 2000). However, by far the most complex aspect of POMDP planning is the generation of a policy for action selection, which is described next. For example in robotics, calculating beliefs over state spaces with 10^6 states is easily done in real-time (Burgard, Cremers, Fox, Hahnel, Lakemeyer, Schulz, Steiner, & Thrun, 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.

2.1.2. Computing an Optimal Policy

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

$$\pi(b) \quad \longrightarrow \quad a, \tag{2.8}$$

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:

$$\pi^*(b_t) \quad = \quad \underset{\pi}{\operatorname{argmax}}\, E_\pi \left[\sum_{t=t_0}^{\infty} \gamma^{t-t_0} r_t \mid b_t \right]. \tag{2.9}$$

There are two distinct but interdependent reasons why computing an optimal policy is challenging.

The more widely-known reason is the so-called curse of dimensionality: in a problem with n physical states, π is defined over all belief states in an $(n-1)$-dimensional continuous space.

The less-well-known reason is the curse of history: POMDP policy optimization is in many ways like breadth-first search in the space of belief states. Starting from the empty history, it grows a set of histories (each corresponding to a reachable belief) by simulating the POMDP. So, the number of distinct action-observation histories considered for policy optimization grows exponentially with the planning horizon.

The two curses—dimensionality and history—are related: the higher the dimension of a belief space, the more room it has for distinct histories. But, they often act independently: planning complexity can grow exponentially with horizon even in problems with only a

few states, and problems with a large number of physical states may still only have a small number of relevant histories.

The most straightforward approach to finding optimal policies remains the value iteration approach (Sondik, 1971), 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 $\mathbb{R}$. Beginning with the initial value function:

$$V_0(b) \quad = \quad \max_a \sum_{s \in S} b(s) R(s,a), \tag{2.10}$$

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

$$V_t(b) \quad = \quad \max_a \left[\sum_{s \in S} b(s) R(s,a) + \gamma \sum_{z \in Z} Pr(z \mid a, b) V_{t-1}(\tau(b,a,z)) \right], \tag{2.11}$$

where the function $\tau(b,a,z)$ is the belief updating function defined in Equation 2.7. This value function update maximizes the expected sum of all (possibly discounted) future payoffs the agent receives in the next t time steps, for any belief state b. Thus, it produces a policy that is optimal under the planning horizon t. The optimal policy can also be directly extracted from the previous-step value function:

$$\pi_t^*(b) \quad = \quad \operatorname*{argmax}_a \left[\sum_{s \in S} b(s) R(s,a) + \gamma \sum_{z \in Z} Pr(z \mid a, b) V_{t-1}(\tau(b,a,z)) \right]. \tag{2.12}$$

Sondik showed that the value function at any finite horizon t can be expressed by a set of vectors: $\Gamma_t = \{\alpha_0, \alpha_1, \ldots, \alpha_m\}$. Each α-vector represents an $|S|$-dimensional hyperplane, and defines the value function over a bounded region of the belief:

$$V_t(b) \quad = \quad \max_{\alpha \in \Gamma_t} \sum_{s \in S} \alpha(s) b(s). \tag{2.13}$$

In addition, each α-vector is associated with an action, defining the best immediate policy assuming optimal behavior for the following $(t-1)$ steps (as defined respectively by the sets $\{V_{t-1}, ..., V_0\}$).

The t-horizon solution set, Γ_t, can be computed as follows. First, we rewrite Equation 2.11

$$V_t(b) \quad = \quad \max_{a \in A} \left[\sum_{s \in S} R(s,a) b(s) + \gamma \sum_{z \in Z} \max_{\alpha \in \Gamma_{t-1}} \sum_{s \in S} \sum_{s' \in S} T(s,a,s') O(s',a,z) \alpha(s') b(s) \right]. \tag{2.14}$$

The value $V_t(b)$ cannot be computed directly for each belief $b \in B$ (since there are infinitely many), but the corresponding set Γ_t can be generated through a sequence of operations on the set Γ_{t-1}.

The first operation is to generate intermediate sets $\Gamma_t^{a,*}$ and $\Gamma_t^{a,z}, \forall a \in A, \forall z \in Z$ *(Step 1)*:

$$\begin{aligned} \Gamma_t^{a,*} &\leftarrow \alpha^{a,*}(s) = R(s,a) \\ \Gamma_t^{a,z} &\leftarrow \alpha_i^{a,z}(s) = \gamma \sum_{s' \in S} T(s,a,s') O(s',a,z) \alpha_i'(s'), \forall \alpha_i' \in \Gamma_{t-1} \end{aligned} \tag{2.15}$$

where each $\alpha^{a,*}$ and $\alpha_i^{a,z}$ is once again an $|S|$-dimensional hyper-plane.

Next we create Γ_t^a ($\forall a \in A$), the cross-sum [1] over observations, which includes one $\alpha^{a,z}$ from each $\Gamma_t^{a,z}$ *(Step 2)*:

$$\Gamma_t^a = \Gamma_t^{a,*} \oplus \Gamma_t^{a,z_1} \oplus \Gamma_t^{a,z_2} \oplus \ldots \tag{2.16}$$

Finally we take the union of Γ_t^a sets *(Step 3)*:

$$\Gamma_t = \cup_{a \in A} \Gamma_t^a. \tag{2.17}$$

The actual value function V_t is extracted from the set Γ_t as described in Equation 2.13.

Using this approach, bounded-time POMDP problems with finite state, action, and observation spaces can be solved exactly given a choice of the horizon t. If the environment is such that the agent might not be able to bound the planning horizon in advance, the policy $\pi_t^*(b)$ is an approximation to the optimal one whose quality improves with the planning horizon t (assuming $0 \leq \gamma < 1$).

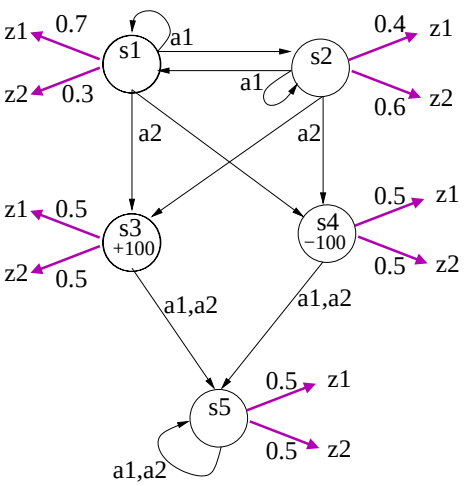
As mentioned above, the value function V_t can be extracted directly from the set Γ_t. An important result shows that for a finite planning horizon, this value function is a piecewise linear, convex, and continuous function of the belief (Sondik, 1971). The piecewise-linearity and continuous properties are a direct result of the fact that V_t is composed of finitely many linear α-vectors. The convexity property can be attributed to the $\max$ in Equation 2.13. It is worth pointing out that the intermediate sets $\Gamma_t^{a,z}$, $\Gamma_t^{a,*}$ and Γ_t^a are also composed entirely of segments that are linear in the belief. This property holds for the intermediate representations because they incorporate the expectation over observation probabilities (Eqn 2.15).

Before proceeding any further, it is useful to consider a simple POMDP example first proposed by Thrun, Fox and Burgard (In preparation) and go through the steps of constructing a value function solution.

EXAMPLE 2.1.1. *Consider the 5-state problem illustrated in Figure 2.1. The agent starts in state $s1$ or $s2$ with equal probability. When in those states, the observation function provides (noisy)*

[1] The symbol $\oplus$ denotes the cross-sum operator. A cross-sum operation is defined over two sets, $A = \{a_1, a_2, \ldots, a_m\}$ and $B = \{b_1, b_2, \ldots, b_n\}$, and produces a third set, $C = \{a_1 + b_1, a_1 + b_2, \ldots, a_1 + b_n, a_2 + b_1, a_2 + b_2, \ldots, \ldots, a_m + b_n\}$.

evidence of the current state. By taking action $a1$, the agent stochastically moves between $s1$ and $s2$, whereas by taking action $a2$ the agent moves to $s3$ or $s4$. State $s5$ is an absorbing state. The reward function is such that it is good (+100) to go through state s3 and bad (-100) to go through state s4. The reward elsewhere is zero. A discount factor $\gamma = 1$ is assumed.



$$T(s, a1, s') = \begin{matrix} \\ s \end{matrix} \begin{matrix} s' \\ \end{matrix} \begin{bmatrix} 0.1 & 0.9 & 0 & 0 & 0 \\ 0.8 & 0.2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.0 \\ 0 & 0 & 0 & 0 & 1.0 \\ 0 & 0 & 0 & 0 & 1.0 \end{bmatrix}$$

$$T(s, a2, s') = \begin{matrix} \\ s \end{matrix} \begin{matrix} s' \\ \end{matrix} \begin{bmatrix} 0 & 0 & 0.9 & 0.1 & 0 \\ 0 & 0 & 0.1 & 0.9 & 0 \\ 0 & 0 & 0 & 0 & 1.0 \\ 0 & 0 & 0 & 0 & 1.0 \\ 0 & 0 & 0 & 0 & 1.0 \end{bmatrix}$$

$$O(s, z) = \begin{matrix} \\ s \end{matrix} \begin{matrix} z \\ \end{matrix} \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \\ 0.5 & 0.5 \\ 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix} \qquad R(s) = \begin{matrix} \\ s \end{matrix} \begin{bmatrix} 0 \\ 0 \\ 100 \\ -100 \\ 0 \end{bmatrix}$$

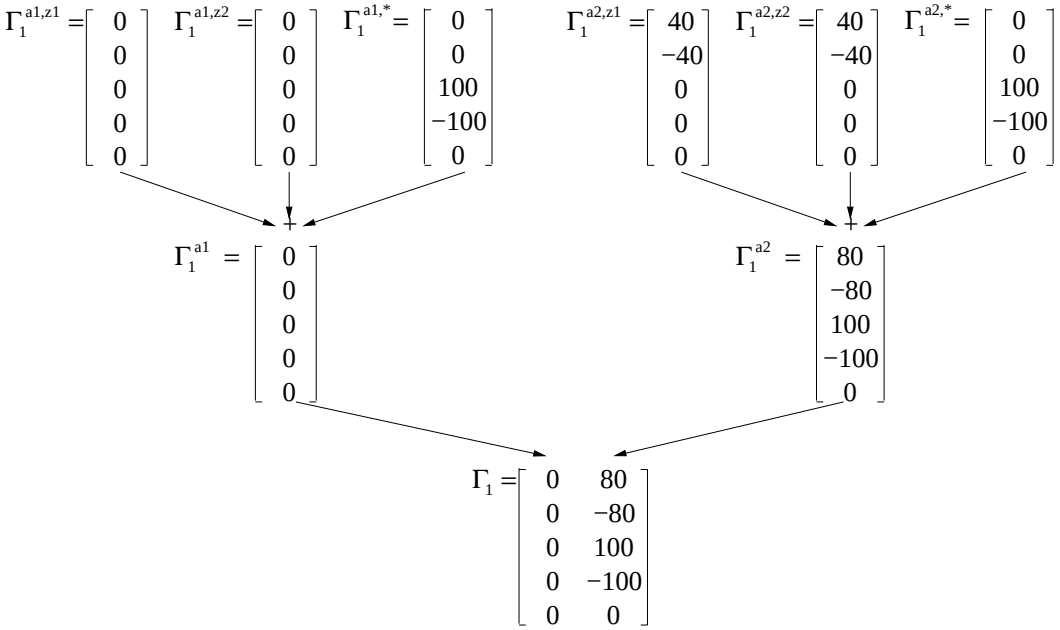
Figure 2.1. Simple POMDP example

To begin solving this problem, an initial solution set Γ_0 is extracted directly from the reward function, by including one α-vector per action:

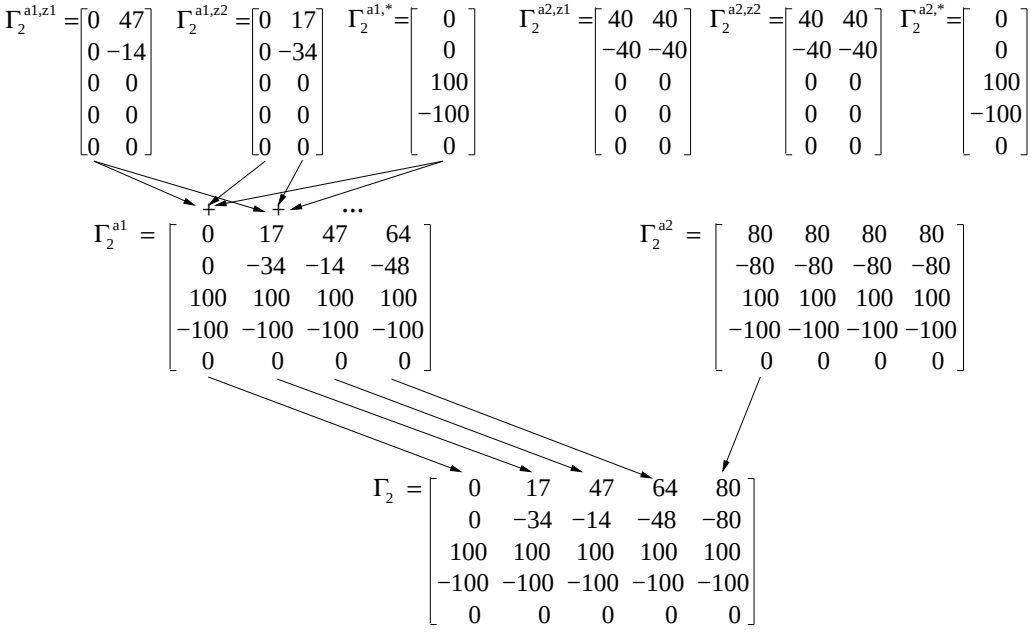
$$\Gamma_0 \leftarrow \alpha^a(s) = R(s, a), \forall a \in A.$$

Figure 2.3a shows the initial value function V_0. This figure only shows the first two dimensions (i. e. $V(b)$ for $b(s1), b(s2)$), even though the full value function is defined in five dimensions (one per state). In this problem, the value function happens to be constant (for any horizon t) in the other dimensions, therefore it is sufficient to show only the first two dimensions.

Figure 2.2a describes the steps leading to a horizon $t = 1$ solution. The first step is to project Γ_0 according to each action / observation pair, as described in Equation 2.15. The second step describes the cross-sum operation (Eqn 2.16). In this case, because each $\Gamma_1^{a,z}$ contains a single vector, the cross-sum reduces to a simple sum. The final step is to take the union of the two Γ_1^a as described in Equation 2.17. This produces the horizon $t = 1$ solution for this five-state problem. The corresponding value function is illustrated in Figure 2.3b.

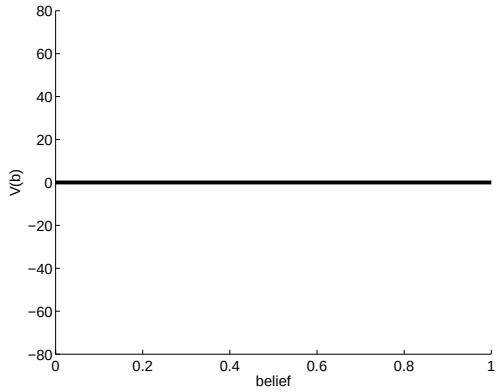


(a) t=1

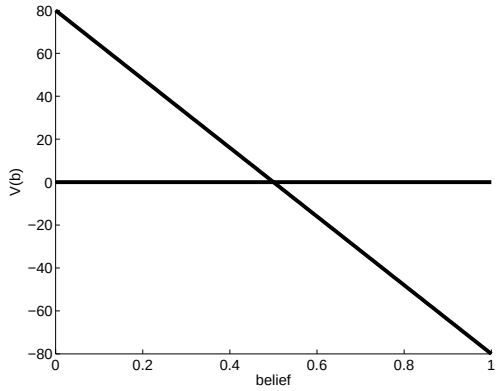


(b) t=2

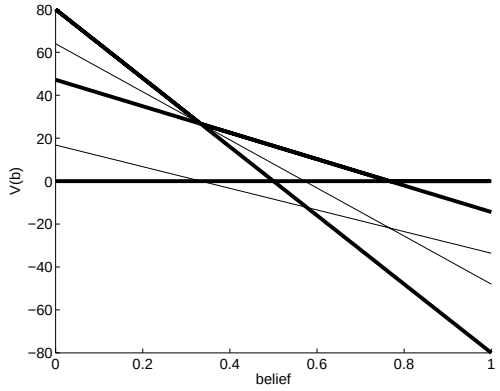
Figure 2.2. Exact value iteration



(a) t=0



(b) t=1



(c) t=2

Figure 2.3. Value function for first three iterations

Figure 2.2b describes the construction of the horizon $t = 2$ value function. It begins by projecting the Γ_1 vectors according to each action / observation pair. In this case, there are two vectors in Γ_1, therefore there will be two vectors in each set $\Gamma_2^{a,z}$. Next, the cross-sum operation takes all possible combinations of vectors from $\Gamma_2^{a,z1}$ and $\Gamma_2^{a,z2}$ and sums them (in combination with $\Gamma_2^{a,*}$). In the case of Γ_2^{a2}, this leads to four identical vectors, since each set $\Gamma_2^{a,z}$ contains two copies of the same vector. The final step is to take the union of Γ_2^{a1} and Γ_2^{a2}; in this case it is safe to include only one copy of the vectors from Γ_2^{a2}. The set Γ_2 then contains five vectors, as illustrated in Figure 2.3c. Additional iterations can be performed in this manner to plan over a longer horizon. This concludes the discussion of this example. □

In the worst case, the exact value update procedure described here requires time doubly exponential in the planning horizon t (Kaelbling et al., 1998). To better understand the complexity of the exact update, let $|S|$ be the number of states, $|A|$ the number of actions, $|Z|$ the number of observations, and $|\Gamma_{t-1}|$ the number of α-vectors in the previous solution set. Then Step 1 creates $|A|\,|Z|\,|\Gamma_{t-1}|$ projections and Step 2 generates $|A|\,|\Gamma_{t-1}|^{|Z|}$ cross-sums. So, in the worst case, the new solution requires:

$$|\Gamma_t| = O(|A||\Gamma_{t-1}|^{|Z|}) \tag{2.18}$$

α-vectors to represent the value function at horizon $t-1$; these can be computed in time $O(|S|^2|A|\,|\Gamma_{t-1}|^{|Z|})$.

It is often the case that a vector in Γ_t will be completely dominated by another vector:

$$\alpha_i \cdot b < \alpha_j \cdot b, \quad \forall b. \tag{2.19}$$

Similarly, a vector may be fully dominated by a set of other vectors. This vector can then be pruned away without affecting the solution. A quick look at the graphical representation of Γ_2 in the example above (Fig. 2.3c) shows that two of its vectors can be eliminated since they are dominated by other vectors.

Finding dominated vectors can be expensive. Checking whether a single vector is dominated requires solving a linear program with $|S|$ variables and $|\Gamma_t|$ constraints. But, it can be time-effective to apply pruning after each iteration to prevent an explosion of the solution size. In practice, $|\Gamma_t|$ often appears to grow singly exponentially in t, even given clever mechanisms for pruning unnecessary linear functions. This enormous computational complexity has long been a key impediment toward applying POMDPs to practical problems.

2.2. Existing POMDP Approaches

A number of approaches have been proposed to overcome the computational hurdle posed by exact POMDP planning. The rest of this section reviews the rich literature of POMDP algorithms—both exact and approximate—which are available. Unless stated otherwise, all methods assume a fully known model of the problem domain.

2.2.1. Exact Value Iteration Algorithms

The exact value iteration (VI) algorithm described in Section 2.1 is generally known as the Enumeration algorithm (Sondik, 1971; Monahan, 1982). It was not the first exact POMDP algorithm, but is probably the simplest to explain. Many early exact VI algorithms propose variations on the same basic ideas.

Sondik's (1971) One-Pass algorithm selects arbitrary belief points, constructs an α-vector for that point, and a belief region over which the α-vector is dominant. The definition of regions is generally conservative, and thus the algorithm may re-define the same α-vector for multiple adjacent regions. Cheng's (1988) linear support algorithm works along similar lines, but uses less constraining conditions to define the belief region. As a result, it may define fewer belief regions, but checking the constraints on the region can be more expensive.

Littman's (1996) Witness algorithm uses an even more sophisticated approach: given a belief point, it constructs the corresponding α-vector for a specific action and observation. It then considers the region over which this vector is dominant, and looks for evidence (i. e. a *Witness* point) where the vector is suboptimal. When it finds such a point, it can generate the best vector at that point and so on until no new witnesses are found. This produces an optimal solution.

The Incremental Pruning algorithm (Zhang & Liu, 1996; Cassandra, Littman, & Zhang, 1997) is a direct extension of the enumeration algorithm we described above. The principal insight is that the pruning of dominated α-vectors (Eqn 2.19) can be interleaved directly with the cross-sum operator (Eqn 2.16). The resulting value function is the same, but the algorithm is more efficient because it discards unnecessary vectors earlier on.

The most recent (and most effective) exact VI algorithm for POMDPs interleaves point-based value updates (much like Cheng's algorithm), with full exact value backups (Zhang & Zhang, 2001). Unlike in Cheng's algorithm, the belief points for the point-based updates

are selected heuristically and therefore are many fewer. The use of point-based value updates mean that many fewer exact updates are needed, while the interleaved exact updates guarantee that the algorithm converges to the optimal solution.

Despite the increasing degrees of sophistication exhibited by exact value iteration algorithms, they are still completely impractical for problems with more than a few dozen states and even fewer actions and observations. The main hurdle remains the (potentially exponential) number of α-vectors generated with each value backup.

2.2.2. Grid-Based Value Function Approximations

There exists many approaches that approximate the value function using a finite set of belief points along with their values. These points are often distributed according to a grid pattern over the belief space, thus the name *grid-based approximation*. An interpolation-extrapolation rule specifies the value at non-grid points as a function of the value of neighboring grid-points.

Performing value backups over grid-points is relatively straightforward: dynamic programming updates as specified in Equation 2.11 can be adapted to grid-points for a simple polynomial-time algorithm. Given a set of grid points G, the value at each $b^G \in G$ is defined by:

$$V(b^G) = \max_a \left[\sum_{s \in S} b^G(s) R(s,a) + \gamma \sum_{z \in Z} Pr(z \mid a, b) V(\tau(b,a,z)) \right]. \quad (2.20)$$

If $\tau(b,a,z)$ is part of the grid, then $V(\tau(b,a,z))$ is defined by the value backups. Otherwise, $V(\tau(b,a,z))$ is approximated using an interpolation rule such as:

$$V(\tau(b,a,z) = \sum_{i=1}^{|G|} \lambda(i) V(b_i^G), \quad (2.21)$$

where $\lambda(i) \geq 0$ and $\sum_{i=1}^{|G|} \lambda(i) = 1$. This produces a convex combination over grid-points. The two more interesting questions with respect to grid-based approximations are (1) how to calculate the interpolation function; and (2) how to select grid points.

In general, to find the interpolation that leads to the best value function approximation at a point b requires solving the following linear program:

$$\text{Minimize} \quad \sum_{i=1}^{|G|} \lambda(i) V(b_i^G) \tag{2.22}$$

$$\text{Subject to} \quad b = \sum_{i=1}^{|G|} \lambda(i) b_i^G \tag{2.23}$$

$$\sum_{i=1}^{|G|} \lambda(i) = 1 \tag{2.24}$$

$$0 \leq \lambda(i) \leq 1, 1 \leq i \leq |G|. \tag{2.25}$$

Different approaches have been proposed to select grid points. Lovejoy (1991a) constructs a fixed-resolution regular grid over the entire belief space. A benefit is that value interpolations can be calculated quickly by considering only neighboring grid-points. The disadvantage is that the number of grid points grows exponentially with the dimensionality of the belief (i. e. with the number of states). A simpler approach would be to select random points over the belief space (Hauskrecht, 1997). But this requires slower interpolation for estimating the value of the new points. Both of these methods are less than ideal when the beliefs encountered are not uniformly distributed. In particular, many problems are characterized by dense beliefs at the edges of the simplex (i. e. probability mass focused on a few states, and most other states have zero probability), and low belief density in the middle of the simplex. A distribution of grid-points that better reflects the actual distribution over belief points is therefore preferable.

Alternately, Hauskrecht (1997) also proposes using the corner points of the belief simplex (e. g. [1 0 0 ...], [0 1 0 ...], ... , [0 0 0 ... 1]), and generating additional successor belief points through one-step stochastic simulations (Eqn 2.7) from the corner points. He also proposes an approximate interpolation algorithm that uses the values at $|S| - 1$ critical points plus one non-critical point in the grid. An alternative approach is that of Brafman (1997), which builds a grid by also starting with the critical points of the belief simplex, but then uses a heuristic to estimate the usefulness of gradually adding intermediate points (e. g. $b_k = 0.5b_i + 0.5b_j$, for any pair of points). Both Hauskrecht's and Brafman's methods—generally referred to as *non-regular grid approximations*—require fewer points than Lovejoy's regular grid approach. However the interpolation rule used to calculate the value at non-grid points is typically more expensive to compute, since it involves searching over all grid points, rather than just the neighboring sub-simplex.

Zhou and Hansen (2001) propose a grid-based approximation that combines advantages from both regular and non-regular grids. The idea is to sub-sample the regular fixed-resolution grid proposed by Lovejoy. This gives a variable resolution grid since some parts of the beliefs can be more densely sampled than others and by restricting grid points to lie on the fixed-resolution grid the approach can guarantee fast value interpolation for non-grid points. Nonetheless, the algorithm often requires a large number of grid points to achieve good performance.

Finally, Bonet (2002) proposes the first grid-based algorithm for POMDPs with ϵ-optimality (for any $\epsilon > 0$). This approach requires thorough coverage of the belief space such that every point is within δ of a grid-point. The value update for each grid point is fast to implement, since the interpolation rule depends only on the nearest neighbor of the one-step successor belief for each grid point (which can be pre-computed). The main limitation is the fact that ϵ-coverage of the belief space can only be attained by using exponentially many grid points.

2.2.3. General Value Function Approximations

Another class of proposed POMDP planning algorithms focuses on directly approximating the value function. In the work of Parr and Russell (1995), discrete-state POMDPs are solved by approximating the piecewise-linear continuous POMDP value function using a smooth and differentiable function that is optimized by gradient descent. Thrun (2000) describes how continuous state POMDPs can be solved by using particle filtering to do approximate tracking of the belief state and using a nearest-neighbor function approximation for the value function. While value function approximation is a promising avenue for addressing large-scale POMDPs, it generally offers few theoretical guarantees on performance.

2.2.4. MDP-Type Heuristics

MDP planning is a special case of POMDP planning, which assumes that the state is fully observable at each time step. While it is clear that the optimal MDP solution will be sub-optimal for partially observable domains, it can nonetheless lead to reasonably good control in some situations. Many heuristic POMDP approaches use the exact MDP policy, in combination with full exact belief tracking, to extract a control policy.

These heuristics generally optimize an MDP solution by performing dynamic programming updates on the Q-function:

$$Q_{MDP}(s,a) \quad = \quad R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') \max_{a' \in A} Q_{MDP}(s',a') \tag{2.26}$$

where $Q_{MDP}(s,a)$ represents the expected discounted sum of rewards for taking action a in state s and is defined over all states $s \in S$ and actions $a \in A$. All other terms are defined as in Section 2.1. Whenever the state is fully observable, the exact MDP value function and policy can extracted by maximizing the action selection:

$$V_{MDP}(s) \quad = \quad \max_{a \in A} Q_{MDP}(s,a) \tag{2.27}$$

$$\pi_{MDP}(s) \quad = \quad \operatorname*{argmax}_{a \in A} Q_{MDP}(s,a). \tag{2.28}$$

When the state is only partially observable, the heuristic methods described below use the Q-function to extract a belief-conditioned policy $\pi(b)$. The belief is typically tracked according to Equation 2.7.

The simplest MDP-type heuristic for POMDP control is the Most-Likely State (*MLS*) heuristic (Nourbakhsh, Powers, & Birchfield, 1995):

$$\pi_{MLS}(b) \quad = \quad \operatorname*{argmax}_{a \in A} Q_{MDP}(\operatorname*{argmax}_{s \in S} b(s), a). \tag{2.29}$$

It has been used extensively in robot navigation applications. In fact, it is usually the common assumption underlying any approach that uses exact MDP planning for real-world domains. As its name implies, it typically performs well when the uncertainty is localized around a single most-likely state, but performs poorly when there is clear ambiguity since it is unable to reason about actions that would explicitly resolve the uncertainty.

A similar approach is the *voting* heuristic (Simmons & Koenig, 1995):

$$\pi_{VOTING}(b) \quad = \quad \operatorname*{argmax}_{a \in A} \sum_{s \in S} b(s) \delta(\operatorname*{argmax}_{a' \in A} Q_{MDP}(s,a'), a) \tag{2.30}$$

$$\text{where } \delta(a',a) = \{1, \text{ if } a = a'; 0, \text{ else } \}, \tag{2.31}$$

which weighs the action choice by the probability of each state. In the case of uni-modal belief distributions, the policy is the same as with the *MLS* heuristic. However some cases with competing hypotheses may be better handled by the *voting* heuristic where consistent action choices by many low-probability states could outweigh the choice of the most-likely state.

The *QMDP* heuristic (Littman, Cassandra, & Kaelbling, 1995a) takes into account partial observability at the current step, but assumes full observability on subsequent steps:

$$\pi_{QMDP}(b) \quad = \quad \operatorname*{argmax}_{a \in A} \sum_{s \in S} b(s) Q_{MDP}(s,a). \tag{2.32}$$

The resulting policy has some ability to resolve uncertainty, but cannot benefit from long-term information gathering, or compare actions with different information potential. Despite this, it often outperforms the *MLS* heuristic by virtue of its ability to reason about at least one step of uncertainty.

The Fast-Informed Bound *FIB* heuristic (Hauskrecht, 2000) uses a similar approach, but incorporates observation weights into the calculation of the Q-function:

$$Q_{FIB}(s,a) = R(s,a) + \gamma \sum_{z \in Z} \max_{a' \in A} \sum_{s' \in S} O(s',a,z) T(s,a,s') Q_{FIB}(s',a') \quad (2.33)$$

$$V_{FIB}(b) = \max_{a \in A} \sum_{s \in S} b(s) Q_{FIB}(s,a) \quad (2.34)$$

$$\pi_{FIB}(b) = \operatorname*{argmax}_{a \in A} \sum_{s \in S} b(s) Q_{FIB}(s,a). \quad (2.35)$$

The goal is to choose the best action conditioned on the expected observation probabilities, in addition to the next state. The *FIB* value function, V_{FIB}, has the nice property that it is guaranteed to lie between the MDP value function (Eqn 2.27) and the exact POMDP solution (Eqn 2.11). Hauskrecht (2000) shows promising experimental results obtained by using this approach on a simulated 20-state maze domain. In many domains, it performs on par with the *QMDP* heuristic.

2.2.5. Belief Space Compression

While the grid-based methods of Section 2.2.2 reduce computation by sparsely sampling the belief state, there exists a related class of algorithms that explicitly looks at finding lower dimensional representations of the belief space. These approaches tackle the POMDP planning problem by first finding an appropriate sub-dimensional manifold of the belief space, and then by learning a value function over that sub-dimensional space.

One such approach is called value-directed compression (Poupart & Boutilier, 2003). It considers a sequence of linear projections to find the smallest linear sub-dimensional manifold that is both consistent with the reward function, and invariant with respect to transition and observation parameters. Since the algorithm finds a linear projection of the belief space, exact POMDP planning can be done directly in the projected space, and the full value function recovered through inverse projection. In practice, few domains have low-dimensional linear sub-manifolds. In such cases, an approximate version of the algorithm is also possible.

An alternative approach is the E-PCA algorithm (Roy & Gordon, 2003), which uses Exponential-family Principal Component Analysis to project high-dimensional beliefs onto

a low-dimensional, non-linear, manifold. By considering non-linear manifolds, this approach generally achieves much greater compression than linear compression techniques. However, planning over a non-linear manifold is more complicated. Grid-based-type approaches can be adapted to produce a computationally-feasible solution (Roy, 2003), but it does not offer any theoretical guarantees with respect to optimal performance.

Overall, these algorithms offer promising ways of overcoming the curse of dimensionality, and in particular the E-PCA has shown impressive success in planning over large-scale domains. However planning over sub-dimensional manifolds is still subject to the curse of history, and therefore may best be used in conjunction with history-reducing approaches, such as the ones proposed in this thesis, to offer maximum scalability.

2.2.6. History-Based Approaches

The main idea behind history-based methods is to move away from the concept of a belief state, and instead express policies conditioned on sequences of recent observations. The advantage of these methods is that they do not require model parameterization, but rely strictly on observable quantities (actions, observations, rewards) to express and optimize a control policy.

The UTree algorithm (McCallum, 1996) offers an approach in which the observation histories are represented using a suffix tree with variable depth leaves, and where branches are grown whenever a new observation sequence is not Markovian with respect to the reward.

The more recent Predictive State Representation (PSR) (Littman, Sutton, & Singh, 2002; Singh, Littman, Jong, Pardoe, & Stone, 2003) is based on a similar premise, but instead of using *history* to condition action-choices, the policy is conditioned on *test predictions*, where a test is a sequence of future observations. In this context, states are expressed strictly in terms of probabilities of observation sequences. The set of core tests can be learned directly from exploration data (Singh et al., 2003; Rosencrantz, Gordon, & Thrun, 2004).

A key advantage of these approaches is that they do not require a model of the domain. Instead, training data is required to learn the policy. However this can be problematic for planning problems where exploration costs are high.

2.2.7. Structured Approaches

Many real-world domains have structure that can be exploited to find good policies for complex problems. This is a common idea in planning, and has been richly exploited by the MDP community. Leveraging of structure for POMDP planning is also found in

a number of hierarchical POMDP algorithms, where structure takes the form of multi-resolution temporally abstract actions (which are in fact policies over primitive actions). In this framework, the goal is to plan over subtasks by learning policies at different levels of action abstraction.

Preliminary attempts at hierarchical partitioning of POMDP problems into subtasks typically make strict assumptions about prior knowledge of low-level tasks and ordering, which are substantially restrictive. The HQ-learning algorithm (Wiering & Schmidhuber, 1997) learns a sequence of subgoals, assuming that each subgoal is satisfied through a reactive policy, and subgoal completion is fully observable. Castañon (1997) addresses a specific sensor management problem, for which he decomposes a multi-object detection problem into many single-object detection problems. He assumes a hierarchy of depth=2, where each single-object problem (i. e. low-level subtask) is solved using a standard POMDP algorithm, and these solutions are used to guide high-level coordination and resource allocation such that the multi-object problem is satisfied. This does not obviously extend to significantly different problems, such as those encountered in robot control.

Most recently, interesting hierarchical approaches to POMDPs have been proposed, which rely heavily on exploration and training to learn policies for large POMDP problems. One approach proposes a hierarchical extension of McCallum's (1996) Utile Suffix Memory algorithm, which builds observation histories at variable time resolutions (Hernandez-Gardiol & Mahadevan, 2001). Another approach extends the Hierarchical Hidden Markov Model (Fine, Singer, & Tishby, 1998) to include actions, and thus accommodate POMDP problems, thereby allowing various levels of spatial abstraction (Theocharous, Rohanimanesh, & Mahadevan, 2001). Both of these approaches assume that termination conditions are defined for subtasks, and can be detected during execution, which limits their applicability to many POMDP problems. Furthermore, they are best suited to problems where exploration and data collection are inexpensive.

Other structured POMDP approaches do not rely on hierarchical decomposition, but instead derive their computational power from assuming a highly-independent factored state representation (Boutilier & Poole, 1996; Boutilier et al., 1999). In this case, a set of orthogonal multi-valued features is used to represent state and/or action sets. One can then use two-stage temporal Bayes nets with associated tree-structured conditional probability tables (CPTs) to describe the dynamics and rewards of a factored state POMDP. The CPTs can be manipulated directly to perform exact value iteration or policy iteration. While this approach successfully reduces the POMDP state space representation, it does not directly

reduce the size of the value function representation, which is the main obstacle to the efficient optimization of POMDP solutions.

2.2.8. Policy Search Algorithms

Most methods described so far in this chapter focus on estimating a POMDP value function, from which a policy can then be extracted. An alternate perspective is to directly optimize the policy, and this is explored in a number of algorithms. There are three main considerations when designing a policy search algorithm. First, there is the question of how the policy should be represented. The most often-used representations are the finite-state machine and the parameterized policy class. Second, there is the question of how candidate policies can be evaluated. And finally, there is the question of the actual optimization step, describing which new candidate policies should be considered.

The first exact policy search algorithm for POMDPs is due to Sondik (1978). It represents policies as a mapping from polyhedral regions of the belief space to actions. However, evaluating policies using this representation is extremely complex. Hansen (1998) suggests representing the policy as a finite-state machine or policy graph instead. The policy graph contains a set of nodes, each of which is labeled by an action. Node-to-node (directed) transitions are labeled according to observation; each node has one outgoing transition for each observation. It is worth pointing out that each policy node in a finite-state machine has a corresponding distinct α-vector in the equivalent value function representation (e. g. Fig. 2.3). Policy evaluation is much easier using this representation: it is sufficient to construct the value function from the finite-state machine, which can be done by solving a set of linear equations. Finally, policy improvement is carried-out by operating directly on the policy graph (adding, removing, or re-labeling nodes). Empirical results show that this approach converges faster than exact value iteration, in large part because it often requires fewer iterations. In general, this approach is still overwhelmed by most problems beyond a dozen states; there are exceptions, in particular some infinite-horizon problems which can be controlled by a very simple policy graph (Peshkin, Meuleau, & Kaelbling, 1999).

In an attempt to improve scalability, approximate algorithms have been proposed. Some of these restrict computation by applying policy search over a restricted class of policies. One such approach used a generative model of the POMDP to alternately build and evaluate trajectory trees (Kearns, Mansour, & Ng, 2000). This approach was extended to reduce the number of trees required to guarantee a bound on the error of the policy's value (Ng & Jordan, 2000). In cases where the policy class is a differentiable function (e. g. assuming a stochastic policy, or a continuous action space), gradient ascent can also be

used to optimize the policy (Williams, 1992; Baird & Moore, 1999; Baxter & Bartlett, 2000; Ng, Parr, & Koller, 2000; Kakade, 2002).

Finally, a recent approach named Bounded Policy Iteration (Poupart & Boutilier, 2004) combines insights from both exact policy search and gradient search. This algorithm performs a search over finite-state machines as described by Hansen (1998), but only over controllers of a fixed size. Meanwhile whenever the search is stopped by a local minima, the controller size is increased slightly and the search continues. This approach has demonstrated good empirical performance on relatively large POMDP problems.

There are many reasons for preferring policy search approaches over value function methods. They generalize easily to continuous state/action problems; stochastic policies tend to perform better than (sub-optimal) deterministic ones; value function approximation often does not converge to a stable policy. Nonetheless, they suffer from some limitations: selecting a good policy class can be difficult, and gradient-methods can get trapped in local minima. Despite this, policy search techniques have been successfully applied in real-world domains (Bagnell & Schneider, 2001).

2.3. Summary

This chapter describes the basic concepts in POMDP planning. It discusses the reasons for the computational intractability of exact POMDP solutions, and presents a number of existing algorithms that can overcome these difficulties with varying levels of success.

Despite the rich set of approaches available, we still lack solutions that simultaneously offer performance guarantee, and scalability. Most of the approaches that have been successfully used in real-world domains lack performance guarantees, whereas those algorithms that offer performance bounds typically have not scaled beyond small simulation problems.

The next chapter presents a new algorithm, *Point-Based Value Iteration* (PBVI) which offers both reasonable scalability (in the form of polynomial-time value updates) and an error bound on its performance with respect to the optimal solution. PBVI draws inspiration from many of the approaches discussed in this chapter, in particular grid-based approximations.

CHAPTER 3

Point-Based Value Iteration

POMDPs offer a rich framework to optimize a control strategy. However, computational considerations limit the usefulness of POMDPs in large domains. These considerations include the well-known curse of dimensionality (where the dimensionality of planning problem is directly related to the number of states) and the lesser known curse of history (where the number of plan contingencies increases exponentially with the planning horizon).

In this chapter, we present the *Point-Based Value Iteration* (PBVI) algorithm, which specifically targets the curse of history. From a high-level stand-point, PBVI shares many similarities with earlier grid-based methods (see Section 2.2.2). As with grid-methods, PBVI first selects a small set of representative belief points, and then iteratively applies value updates to those points. When performing value backups however, PBVI updates both the *value* and *value gradient*; this choice provides better generalization to unexplored beliefs, compared to interpolation-type grid-based approximations, which only update the value. Another important aspect is the strategy employed to select belief points. Rather than picking points randomly, or according to a fixed grid, PBVI uses exploratory stochastic trajectories to sample belief points. This approach allows it to restrict belief points to reachable regions of the belief, thus reducing the number of belief points necessary to find a good solution compared to earlier approaches.

This chapter expands on these ideas. Sections 3.1 and 3.2 present the basic PBVI algorithm. Section 3.3 then presents a theoretical analysis of PBVI, which shows that it is guaranteed to have bounded error, with respect to the optimal solution. Section 3.4 discusses the appropriate selection of belief points. Section 3.5 presents an empirical evaluation of the algorithm. Finally, Section 3.6 presents an extension of PBVI that partitions belief points in a metric-tree structure to further accelerate planning.

3.1. Point-Based Value Backup

PBVI relies on one very important assumption, namely that it is often sufficient to plan for a small set of belief points, even when the goal is to get a good solution over a large number of beliefs. Given this premise, it is crucial to understand what it means to plan for a small set of points.

As explained in Section 2.1.2, a value function update can be implemented as a sequence of operations on a set of α-vectors. If we assume that we are only interested in updating the value function at a fixed set of belief points, $B = \{b_0, b_1, ..., b_q\}$, then it follows that the value function will contain at most one α-vector for each belief point. The point-based value function is therefore represented by the corresponding set $\{\alpha_0, \alpha_1, \ldots, \alpha_q\}$.

Figure 3.1 shows two versions of a POMDP value function representation, one that uses a point-based value function (on the left) and one that uses a grid (on the right). As shown on the left, by maintaining a full α-vector for each belief point, PBVI can preserve the piecewise linearity and convexity of the value function, and define a value function over the entire belief simplex. This is in contrast to interpolation-type grid-based approaches which update only the value at each belief grid point.

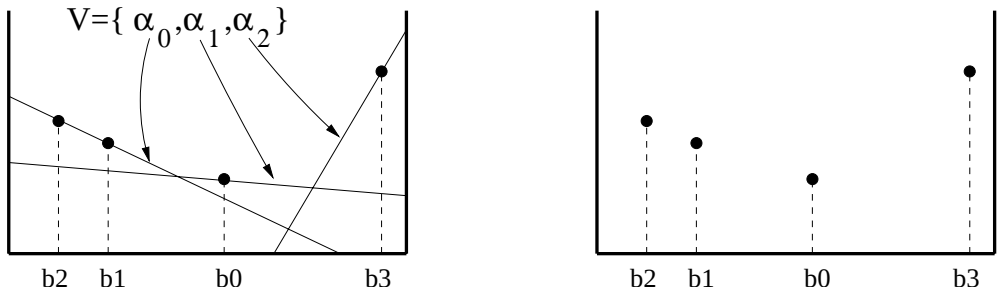


Figure 3.1. Comparing POMDP value function representations

Given a $t-1$-horizon plan, it is relatively straightforward to generate the t-horizon α-vector for a given belief b (Sondik, 1971; Cheng, 1988). In PBVI, we apply this procedure to the entire set of points B such that we generate a full t-horizon value function.

Given a solution set Γ_{t-1}, we begin by modifying the exact backup operator (Eqn 2.14) such that only one α-vector per belief point is maintained. The first operation is to generate intermediate sets $\Gamma^{a,*}$ and $\Gamma^{a,z}, \forall a \in A, \forall z \in Z$ (exactly as in Eqn 2.15) *(Step 1)*:

$$\begin{aligned} \Gamma^{a,*} &\leftarrow \alpha^{a,*}(s) = R(s,a) \\ \Gamma^{a,z} &\leftarrow \alpha_i^{a,z}(s) = \gamma \sum_{s' \in S} T(s,a,s') O(s',a,z) \alpha_i'(s'), \forall \alpha_i' \in \Gamma_{t-1}. \end{aligned} \tag{3.1}$$

Next, whereas performing an exact value update requires a cross-sum operation (Eqn 2.16), by operating over a finite set of points, we can instead use a simple summation. We construct $\Gamma_b^a, \forall b \in B, \forall a \in A$ *(Step 2)*:

$$\Gamma_b^a \quad = \quad \Gamma^{a,*} + \sum_{z \in Z} \operatorname*{argmax}_{\alpha \in \Gamma^{a,z}} \left(\sum_{s \in S} \alpha(s) b(s) \right). \tag{3.2}$$

Finally, we find the best action for each belief point *(Step 3)*:

$$\Gamma_t \quad \leftarrow \quad \operatorname*{argmax}_{\Gamma_b^a, \forall a \in A} \left(\sum_{s \in S} \Gamma_b^a(s) b(s) \right), \quad \forall b \in B. \tag{3.3}$$

While these operations preserve only the best α-vector at each belief point $b \in B$, the value function at *any* belief in the simplex (including $b \notin B$) can be extracted from the set Γ_t:

$$V_t(b) \quad = \quad \max_{\alpha \in \Gamma_t} \sum_{s \in S} \alpha(s) b(s). \tag{3.4}$$

To better understand the complexity of a single point-based update, let $|S|$ be the number of states, $|A|$ the number of actions, $|Z|$ the number of observations, and $|\Gamma_{t-1}|$ the number of α-vectors in the previous solution set. As with an exact update, Step 1 creates $|A||Z||\Gamma_{t-1}|$ projections (in time $|S|^2|A||Z||\Gamma_{t-1}|$). Steps 2 and 3 then reduce this set to at most $|B|$ components (in time $|S||A||\Gamma_{t-1}||Z||B|$). Thus, a full point-based value update takes only polynomial time, and even more crucially, the size of the solution set Γ_t remains constant at every iteration. The point-based value backup algorithm is summarized in Table 3.1.

Γ_t=BACKUP(B, Γ_{t-1})	1
For each action $a \in A$	2
For each observation $z \in Z$	3
For each solution vector $\alpha' \in \Gamma_{t-1}$	4
$\alpha^{a,z}(s) = \gamma \sum_{s' \in S} T(s,a,s') O(s',a,z) \alpha'(s'), \forall s \in S$	5
$\Gamma^{a,z} \leftarrow \alpha^{a,z}$	6
End	7
End	8
End	9
For each belief point $b \in B, \forall s \in S$	10
$\alpha_b \leftarrow \operatorname{argmax}_{a \in A} \left[\sum_{s \in S} R(s,a) b(s) + \sum_{z \in Z} \max_{\alpha \in \Gamma^{a,z}} \left[\sum_{s \in S} \alpha(s) b(s) \right] \right]$	11
If($\alpha_b \notin \Gamma_t$)	12
$\Gamma_t \leftarrow \alpha_b$	13
End	14
Return Γ_t	15

Table 3.1. Point-based value backup

It is worth pointing out that this algorithm includes a trivial pruning step (lines 12-13), whereby PBVI refrains from adding to Γ_t any vector already included in it. As a result, it is

often the case that $|\Gamma_t| \leq |B|$. This situation arises whenever multiple nearby belief points support the same vector (e. g. b_1, b_2 in Fig. 3.1). This pruning step can be computed rapidly and is clearly advantageous in terms of reducing the set Γ_t.

3.2. The Anytime PBVI Algorithm

The complete PBVI algorithm is designed as an *anytime* algorithm, which interleaves two main components: the value update step described in Table 3.1, and steps of belief set expansion. We assume for the moment that belief points are chosen at random, uniformly distributed over the belief simplex. More sophisticated approaches to selecting belief points are presented in Section 3.4 (with a description of the EXPAND subroutine).

PBVI starts with a (small) initial set of belief points to which it applies a first series of backup operations. The set of belief points is then grown, a new series of backup operations are applied to all belief points (old and new), and so on until a satisfactory solution is obtained. By interleaving value backup iterations with expansions of the belief set, PBVI offers a range of solutions, gradually trading off computation time and solution quality.

The full algorithm is presented in Table 3.2. The algorithm accepts as input an initial belief point set ($B_I nit$), an initial value (Γ_0), the number of desired expansions (N), and the planning horizon (T). For problems with a finite horizon T, we run T value backups between each expansion of the belief set. In infinite-horizon problems, we select the horizon so that

$$\gamma^T [R_{\max} - R_{\min}] < \epsilon,$$

where $R_{\max} = \max_{s,a} R(s,a)$ and $R_{\min} = \min_{s,a} R(s,a)$.

The complete algorithm terminates once a fixed number of expansions (N) have been completed. Alternately, the algorithm could terminate once the value function approximation reaches a given performance criterion. This is discussed further in Section 3.3.

Γ=PBVI-MAIN(B_{Init}, Γ_0, N, T)	1
B=B_{Init}	2
$\Gamma = \Gamma_0$	3
For N expansions	4
For T iterations	5
Γ =BACKUP(B,Γ)	6
End	7
B_{new} =EXPAND(B,Γ)	8
$B = B \cup B_{new}$	9
End	10
Return Γ	11

Table 3.2. Algorithm for Point-Based Value Iteration (PBVI)

3.3. Convergence and Error Bounds

For any belief set B and horizon t, PBVI produces an estimate V_t^B. We now show that the error between V_t^B and the optimal value function V^* is bounded.

The bound depends on how densely B samples the belief simplex Δ; with denser sampling, V_t^B converges to V^*. Cutting off the PBVI iterations at any sufficiently large horizon, we know that the difference between V_t^B and the optimal infinite-horizon V^* is not too large. The overall error in PBVI, $||V_t^B - V^*||_\infty$, is bounded by:

$$\|V_t^B - V_t^*\|_\infty + \|V_t^* - V^*\|_\infty.$$

The second term is bounded by $\gamma^t \|V_0^* - V^*\|$ (Bertsekas & Tsitsiklis, 1996). The remainder of this section states and proves a bound on the first term.

We begin by defining the density δ_B of a set of belief points B to be the maximum distance from any legal belief to B. More precisely:

$$\delta_B = \max_{b' \in \Delta} \min_{b \in B} \|b - b'\|_1.$$

Then, we can prove the following lemma:

LEMMA 3.3.1. *The error introduced in PBVI when performing one iteration of value backup over B, instead of over Δ, is bounded by*

$$\epsilon \leq \frac{(R_{\max} - R_{\min})\delta_B}{1-\gamma}$$

Proof: Let $b' \in \Delta$ be the point where PBVI makes its worst error in value update, and $b \in B$ be the closest (1-norm) sampled belief to b'. Let α be the vector that is maximal at b, and α' be the vector that would be maximal at b'. By failing to include α' in its solution set, PBVI makes an error of at most $\alpha' \cdot b' - \alpha \cdot b'$. On the other hand, since α is maximal at b, then $\alpha' \cdot b \leq \alpha \cdot b$. So,

$$\begin{array}{rcll}
\epsilon & \leq & \alpha' \cdot b' - \alpha \cdot b' & \\
& = & \alpha' \cdot b' - \alpha \cdot b' + (\alpha' \cdot b - \alpha' \cdot b) & \text{add zero} \\
& \leq & \alpha' \cdot b' - \alpha \cdot b' + \alpha \cdot b - \alpha' \cdot b & \alpha \text{ optimal at } b \\
& = & (\alpha' - \alpha) \cdot (b' - b) & \text{collect terms} \\
& \leq & \|\alpha' - \alpha\|_\infty \|b' - b\|_1 & \text{Hölder inequality} \\
& \leq & \|\alpha' - \alpha\|_\infty \delta_B & \text{definition of } \delta_B \\
& \leq & \frac{(R_{\max} - R_{\min})\delta_B}{1-\gamma} & \text{see text}
\end{array}$$

The last inequality holds because each α-vector represents the reward achievable starting from some state and following some sequence of actions and observations. The sum of rewards must fall between $\frac{R_{\min}}{1-\gamma}$ and $\frac{R_{\max}}{1-\gamma}$. □

THEOREM 3.3.1. *For any belief set B and any horizon t, the error of the PBVI algorithm $\epsilon_t = \|V_t^B - V_t^*\|_\infty$ is bounded by*

$$\epsilon_t \leq \frac{(R_{\max} - R_{\min})\delta_B}{(1-\gamma)^2}$$

Proof:

$$\begin{array}{rcll}
\epsilon_t & = & ||V_t^B - V_t^*||_\infty & \text{definition of } \epsilon_t \\
& = & ||\tilde{H}V_{t-1}^B - HV_{t-1}^*||_\infty & \tilde{H} \text{ denotes PBVI backup,} \\
& & & \text{and } H \text{ denotes exact backup} \\
& \leq & ||\tilde{H}V_{t-1}^B - HV_{t-1}^B||_\infty + ||HV_{t-1}^B - HV_{t-1}^*||_\infty & \text{triangle inequality} \\
& \leq & \epsilon + ||HV_{t-1}^B - HV_{t-1}^*||_\infty & \text{definition of } \epsilon \\
& \leq & \epsilon + \gamma||V_{t-1}^B - V_{t-1}^*||_\infty & \text{contraction} \\
& = & \epsilon + \gamma\epsilon_{t-1} & \text{definition of } \epsilon_{t-1} \\
& \leq & \frac{(R_{\max}-R_{\min})\delta_B}{1-\gamma} + \gamma\epsilon_{t-1} & \text{lemma 3.3.1} \\
& \leq & \frac{(R_{\max}-R_{\min})\delta_B}{(1-\gamma)^2} & \text{series sum } \square
\end{array}$$

The bound described in this section depends on how densely B samples the belief simplex Δ. In the case where not all beliefs are reachable, we don't need to sample all of Δ densely, but can replace Δ by the set of reachable beliefs $\bar{\Delta}$ (Fig. 3.2). The error bounds and convergence results hold on $\bar{\Delta}$. Nevertheless, it may be difficult to achieve a sufficiently dense sampling of $\bar{\Delta}$ to obtain a reasonable bound. We speculate that it may be possible to devise a more useful bound by incorporating forward-simulation on the selected points. This would tighten the bound over those belief points that are in low-density areas but which, with high probability, lead to high-density areas.

As a side note, it is worth pointing out that because PBVI makes no assumption regarding the initial value function V_0^B, the point-based solution V^B is not guaranteed to improve with the addition of belief points. Nonetheless, the theorem presented in this section shows that the *bound on the error* between V_t^B (the point-based solution) and V^* (the optimal solution) is guaranteed to decrease (or stay the same) with the addition of belief points. In cases where V_t^B is initialized pessimistically (e. g. $V_0^B(s) = \frac{R_{min}}{1-\gamma}, \forall s \in S$), then V_t^B will improve (or stay the same) with each value backup and addition of belief points.

This chapter has thus far skirted the issue of belief point selection, however the bound presented in this section clearly argues in favor of dense sampling over the belief simplex. While randomly selecting points according to a uniform distribution may eventually accomplish this, it is generally inefficient, in particular for high dimensional cases. Furthermore, it does not take advantage of the fact that the error bound holds for dense sampling over *reachable* beliefs. Thus we seek more efficient ways to generate belief points than at random over the entire simplex. This is the issue explored in the next section.

3.4. Belief Point Set Expansion

There is a clear trade-off between including fewer beliefs (which would favor fast planning over good performance), versus including many beliefs (which would slow down planning, but ensure better performance). This brings up the question of *how many* belief points should be included. However the number of points is not the only consideration. It is likely that some collections of belief points (e. g. those frequently encountered) are more likely to produce a good value function than others. This brings up the question of *which* beliefs should be included.

The error bound in Section 3.3 suggests that PBVI performs best when its belief set is uniformly dense in the set of reachable beliefs. As shown in Figure 3.2, we can build a tree of reachable beliefs. In this representation, each path through the tree corresponds to a sequence in belief space, and increasing depth corresponds to an increasing plan horizon.

As shown in this figure, the set of reachable beliefs, $\bar{\Delta}$, grows exponentially with the planning horizon. Including all reachable beliefs would guarantee optimal performance, but at the expense of computational tractability. Therefore, we must select a subset $B \subset \bar{\Delta}$, which is sufficiently small for computational tractability, but sufficiently large for good value function approximation.

The approach we propose consists of initializing the set B to contain the initial belief b_0, and then gradually expanding B by greedily choosing new reachable beliefs that improve the worst-case density as rapidly as possible.

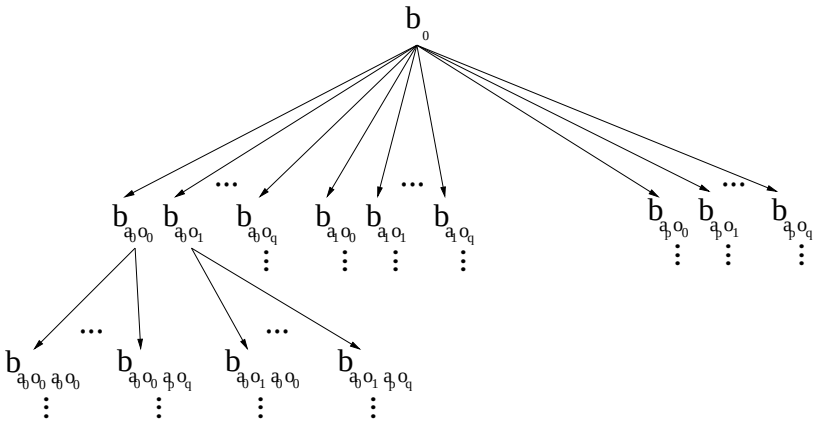


Figure 3.2. The set of reachable beliefs

To choose new reachable beliefs, PBVI stochastically simulates single-step forward trajectories from those points already in B. Simulating a single-step forward trajectory for

a given $b \in B$ requires selecting an action and observation pair (a, z), and then computing the new belief $\tau(b, a, z)$ using the Bayesian update rule (Eqn 2.7).

Rather than selecting a single action to simulate the forward trajectory for a given $b \in B$, PBVI does a one step forward simulation with *each action*, thus producing new beliefs $\{b_{a_0}, b_{a_1}, ...\}$. Rather than accept all new beliefs $\{b_{a_0}, b_{a_1}, ...\}$, PBVI calculates the L_1 distance between each b_a and its closest neighbor in B. It then keeps only that point b_a that is farthest away from any point already in B.

We use the L_1 distance to be consistent with the error bound in Theorem 3.3.1. However the actual choice of norm doesn't appear to matter in practice; we have used both L_1 and L_2 in experiments and the results were practically identical.

Table 3.3 summarizes the belief expansion algorithm. As noted, the single-step forward simulation procedure is repeated for each point $b \in B$, thereby generating one new belief from each previous belief. This means that B at most doubles in size on each belief expansion. Alternately, we could use the same forward simulation procedure to add a fixed number of new beliefs, but since value iteration is much more expensive than belief computation, it seems appropriate to double the size of B at each expansion.

B_{new}=EXPAND(B, V)	1
B_{new}= B	2
Foreach $b \in B$	3
Foreach $a \in A$	4
s=rand$_{multinomial}$(b)	5
s'=rand$_{multinomial}$(T(s,a,$\cdot$))	6
z=rand$_{multinomial}$(O(s',a,$\cdot$))	7
b_a=BELIEF-UPDATE(b, a, z) (Eqn 2.7)	8
End	9
$B_{new} \leftarrow max_{a \in A}\ min_{b' \in B}\ sum_{s \in S}\ \lvert b_a(s) - b'(s) \rvert$	10
End	11
Return B_{new}	12

Table 3.3. Algorithm for belief expansion

3.5. Experimental Evaluation

This section looks at four POMDP simulation domains to evaluate the empirical performance of PBVI. The first three domains—Tiger-grid, Hallway, Hallway2—are extracted from the established POMDP literature. The fourth—Tag—is introduced as a new challenge for POMDP algorithms.

3.5.1. Maze Problems

There exists a set of benchmark problems commonly used to evaluate POMDP planning algorithms (Cassandra, 1999). This section presents results demonstrating the performance of PBVI on some of those problems. It also includes a comparison between PBVI's performance and that of alternate value approximation approaches such as the QMDP heuristic (Littman et al., 1995a), a grid-based method (Brafman, 1997), and another point-based approach (Poon, 2001). While these benchmark problems are relatively small (at most 92 states, 5 actions, and 17 observations) compared to most robotics planning domains, they are useful from an analysis point of view and for comparison to previous work.

The initial performance analysis for PBVI focuses on three well-known problems from the POMDP literature: Tiger-grid (also known as Maze33), Hallway, and Hallway2. All three are maze navigation problems of various sizes. The problems are fully described by Littman, Cassandra, and Kaelbling (1995b); parameterization is available from Cassandra (1999).

Figure 3.3a presents results for the Tiger-grid domain. Replicating earlier experiments (Brafman, 1997), test runs terminate after 500 steps (there's an automatic reset every time the goal is reached) and results are averaged over 151 runs.

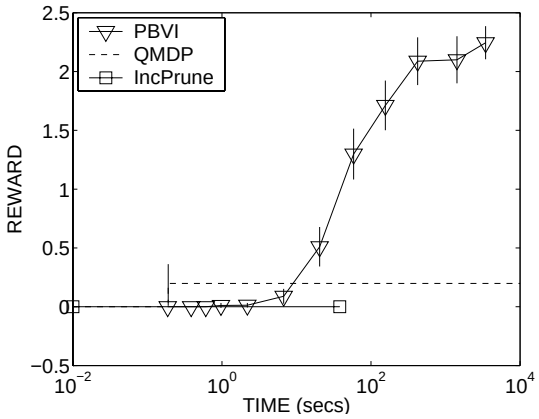
Figures 3.3b and 3.3c present results for the Hallway and Hallway2 domains, respectively. In this case, test runs are terminated when the goal is reached or after 251 steps (whichever occurs first), and the results are averaged over 251 runs. This is consistent with earlier experiments (Littman et al., 1995a).

All three figures compare the performance of three different algorithms:

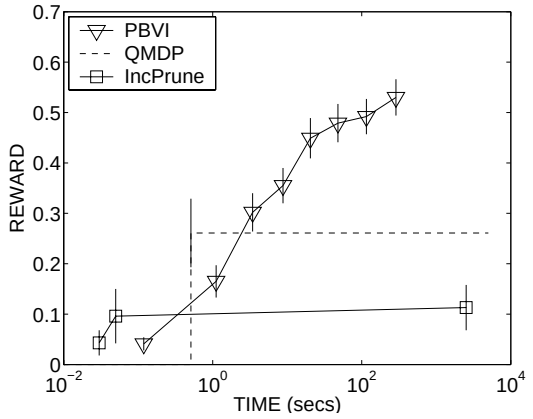
1. QMDP as described in Section 2.2.4,
2. PBVI as described in this chapter,
3. Incremental Pruning as described in Section 2.2.1.

The QMDP algorithm can be seen as providing a good performance baseline; it finds the best plan achievable without considering state uncertainty. For the three problems considered, it finds a policy extremely quickly, but the policy is clearly sub-optimal.

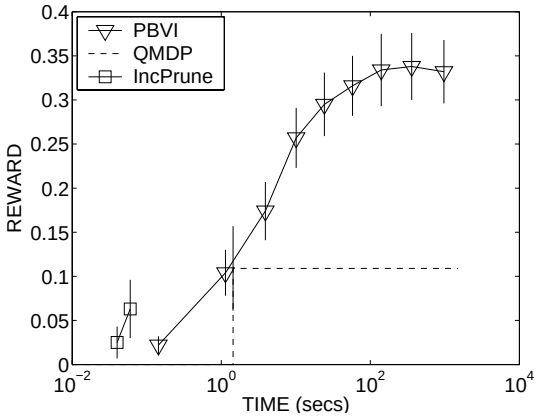
At the other end of the spectrum, the Incremental Pruning algorithm can theoretically find an optimal policy, but for the three problems illustrated, this procedure would take far too long. In fact, only a few iterations of exact value backups were completed in reasonable time. In all three cases, the resulting short-horizon policy was worse than the corresponding QMDP policy.



(a) Tiger-grid



(b) Hallway



(c) Hallway2

Figure 3.3. PBVI performance on well-known POMDP problems

As shown in Figure 3.3, PBVI provides a much better time/performance trade-off. The policies it finds are better than those obtained with QMDP at the expense of longer planning time. Nonetheless in all cases, PBVI is able to find a good policy in a matter of seconds, and does not suffer from the same paralyzing complexity as Incremental Pruning.

While these results are promising, it is not sufficient to compare PBVI only to QMDP and Incremental Pruning—the two ends of the spectrum—when there exists other approximate POMDP approaches. Table 3.4 compares PBVI's performance with previously published results for three additional algorithms: a grid method (Brafman, 1997), an (exact) value-directed compression (VDC) technique (Poupart & Boutilier, 2003), and an alternate point-based approach by (Poon, 2001). While there exist many other algorithms (see Section 2.2 for a detailed listing), these three were selected because they are representative and because related publications provided sufficient information to either replicate the experiments or re-implement the algorithm.

As shown in Table 3.4, we consider the same three problems (Tiger-grid, Hallway and Hallway2) and compare goal completion rates, sum of rewards, policy computation time, and number of required belief points, for each approach. We point out that the results marked [*] were computed by us; other results were likely computed on different platforms, and therefore time comparisons may be approximate at best. Nonetheless the number of samples (where provided) is a direct indicator of computation time. All results

Method	Goal%	Reward $\pm$ Conf.Int.	Time(s)	$\|B\|$
Tiger-Grid (Maze33)				
QMDP (Littman et al., 1995a)[*]	n.a.	0.198	0.19	n.a.
Grid (Brafman, 1997)	n.a.	0.94	n.v.	174
VDC (Poupart & Boutilier, 2003)[*]	n.a.	0.0	24hrs+	n.a.
PBUA (Poon, 2001)	n.a.	2.30	12116	660
PBVI[*]	n.a.	2.25 ± 0.14	3448	470
Hallway				
QMDP (Littman et al., 1995a)[*]	47	0.261	0.51	n.a.
VDC (Poupart & Boutilier, 2003)[*]	25	0.113	24hrs+	n.a.
PBUA (Poon, 2001)	100	0.53	450	300
PBVI[*]	96	0.53 ± 0.04	288	86
Hallway2				
QMDP (Littman et al., 1995a)[*]	22	0.109	1.44	n.a.
Grid (Brafman, 1997)	98	n.v.	n.v.	337
VDC (Poupart & Boutilier, 2003)[*]	15	0.063	24hrs+	n.a.
PBUA (Poon, 2001)	100	0.35	27898	1840
PBVI[*]	98	0.34 ± 0.04	360	95

n.a.=not applicable n.v.=not available

Table 3.4. Results of PBVI for standard POMDP domains

assume a standard (not *lookahead*) controller. In all domains we see that QMDP and the grid method achieve sub-par performance compared to PBUA and PBVI. In the case of QMDP, this is because of fundamental limitations in the algorithm. While the grid method could theoretically reach optimal performance, it would require significantly longer time to do so. Overall, PBVI achieves competitive performance, but with fewer samples than its nearest competitor, PBUA. The reward reported for PBUA seems slightly higher than with PBVI in Tiger-grid and Hallway2, but the difference is well within confidence intervals. However, the number of belief points—and consequently the planning time—is much lower for PBVI. This can be attributed to the belief expansion heuristic used by PBVI, which is described in Section 3.4. The fine details of the algorithmic differences between PBUA and PBVI are discussed in greater detail at the end of this chapter (Section 3.7).

3.5.2. Tag Problem

While the previous section establishes the good performance of PBVI on some well-known simulation problems, these are quite small and do not fully demonstrate the scalability of the algorithm. To provide a better understanding of PBVI's effectiveness for large problems, this section presents results obtained when applying PBVI to the *Tag* problem, a robot version of the popular game of lasertag. In this problem, the agent must navigate its environment with the goal of searching for, and tagging, a moving target (Rosencrantz, Gordon, & Thrun, 2003). Real-world versions of this problem can take many forms; we are particularly interested in a version where an interactive service robot must find an elderly patient roaming the corridors of a nursing home. This scenario is an order of magnitude larger (870 states) than most other POMDP problems considered thus far in the literature (Cassandra, 1999), and was recently proposed as a new challenge for fast, scalable, POMDP algorithms (Pineau, Gordon, & Thrun, 2003a; Roy, 2003).

This scenario can be formulated as a POMDP problem, where the robot learns a policy optimized to quickly find the patient. The patient is assumed to move stochastically according to a fixed policy. The spatial configuration of the environment used throughout this experiment is illustrated in Figure 3.4.

The state space is described by the cross-product of two position features, $\textit{Robot} = \{s_0, \ldots, s_{29}\}$ and $\textit{Person} = \{s_0, \ldots, s_{29}, s_{found}\}$. Both start in independently-selected random positions, and the scenario finishes when $\textit{Person} = s_{found}$. The robot can select from five actions: {*North, South, East, West, Tag*}. A reward of -1 is imposed for each motion action; the *Tag* action results in a $+10$ reward if $\textit{Robot} = \textit{Person}$, or -10 otherwise. Throughout the scenario, the Robot's position is fully observable, and a *Move* action has the predictable

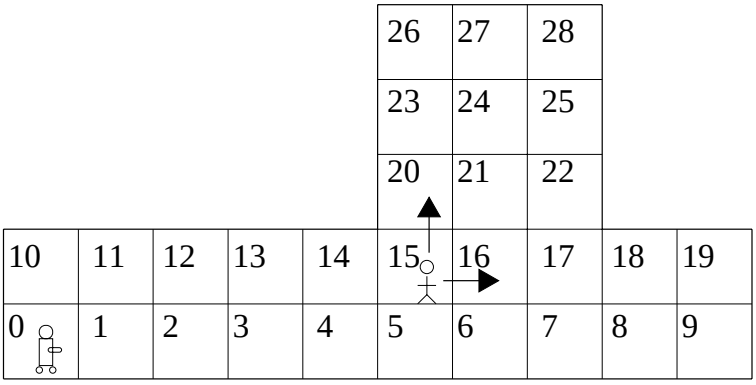


Figure 3.4. Spatial configuration of the domain

deterministic effect, e. g.:

$$Pr(Robot = s_{10} \mid Robot = s_0, North) = 1,$$

and so on for each adjacent cell and direction.

The position of the person, on the other hand, is completely unobservable unless both agents are in the same cell. Meanwhile at each step, the person (with omniscient knowledge) moves away from the robot with $Pr = 0.8$ and stays in place with $Pr = 0.2$, e. g.:

$$Pr(Person = s_{16} \mid Person = s_{15} \& Robot = s_0) = 0.4$$

$$Pr(Person = s_{20} \mid Person = s_{15} \& Robot = s_0) = 0.4$$

$$Pr(Person = s_{15} \mid Person = s_{15} \& Robot = s_0) = 0.2.$$

Figure 3.5 shows the performance of PBVI on the Tag domain. Results are averaged over 10 runs of the algorithm, times 100 different (randomly chosen) start positions for each

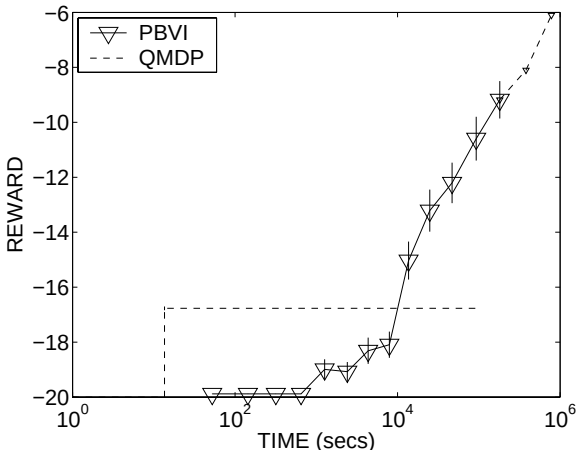


Figure 3.5. PBVI performance on Tag problem

run. It shows the gradual improvement in performance as samples are added (each shown data point represents a new expansion of the belief set with value backups). The QMDP approximation is also tested to provide a baseline comparison. PBVI requires approximately 100 belief points to overcome QMDP, and the performance keeps on improving as more points are added. These results show that PBVI can effectively tackle a problem with 870 states.

This problem is far beyond the reach of the Incremental Pruning algorithm. A single iteration of optimal value iteration on a problem of this size could produce over 10^{20} α-vectors before pruning. Therefore, it was not applied.

This section describes one version of the Tag problem. In fact, it can be re-formulated in a variety of ways to accommodate different environments, person motion models, and observation models. Chapter 5 discusses variations on this problem using more realistic robot and person models.

In addition to the empirical evidence presented here in support of PBVI, it is useful to consider its theoretical properties. The next section discusses the convergence behavior of the algorithm and derives theoretical bounds over its approximation error.

3.5.3. Validation of the Belief Set Expansion

Table 3.3 presents a very specific approach to the initial selection, and gradual expansion, of the belief set. There are many alternative heuristics one could use to generate belief points. This section explores three other possible approaches and compares their performance with the standard PBVI algorithm.

In all cases, we start by assuming that the initial belief b_0 (given as part of the model) is the sole point in the initial set. We then consider four possible expansion methods:

1. Random (**RA**)
2. Stochastic Simulation with Random Action (**SSRA**)
3. Stochastic Simulation with Greedy Action (**SSGA**)
4. Stochastic Simulation with Exploratory Action (**SSEA**)

The **RA** method consists of sampling a belief point from a uniform distribution over the entire belief simplex. **SSEA** is the standard PBVI expansion heuristic (Section 3.4). **SSRA** similarly uses single-step forward simulation, but rather than try all actions, it randomly selects an action $a \in A$ and automatically accepts the posterior belief b_a unless it is already in B. Finally, **SSGA** uses the most recent value function solution to pick the current best (i. e. greedy) action at the given belief b, and uses that action to perform a single-step

forward simulation, which yields a new belief. Tables 3.5 and 3.6 summarize the belief expansion procedure for SSRA and SSGA respectively.

B_{new}=EXPAND$_{SSRA}$(B,V)	1
B_{new}= B	2
Foreach $b \in B$	3
a=rand$_{uniform}$(A)	4
s=rand$_{multinomial}$(b)	5
s'=rand$_{multinomial}$(T(s,a,$\cdot$))	6
z=rand$_{multinomial}$(O(s',a,$\cdot$))	7
B_{new}=BELIEF-UPDATE(b, a, z) (Eqn 2.7)	8
End	9
Return B_{new}	10

Table 3.5. Algorithm for belief expansion with random action selection

B_{new}=EXPAND$_{SSGA}$(B,V)	1
B_{new}= B	2
Foreach $b \in B$	3
a=argmax$_{\alpha \in \Gamma} \sum s \in S \alpha(s) b(s)$	4
s=rand$_{multinomial}$(b)	5
s'=rand$_{multinomial}$(T(s,a,$\cdot$))	6
z=rand$_{multinomial}$(O(s',a,$\cdot$))	7
B_{new}=BELIEF-UPDATE(b, a, z) (Eqn 2.7)	8
End	9
Return B_{new}	10

Table 3.6. Algorithm for belief expansion with greedy action selection

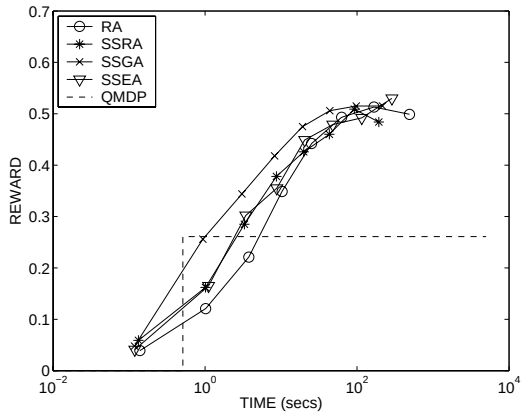
We now revisit the Hallway, Hallway2, and Tag problems from Section 3.2 to compare the performance of these four heuristics. For each problem, we apply PBVI as described in Table 3.2, replacing in turn the EXPAND subroutine (line 9) by each of the four expansion heuristics. The QMDP approximation is included as a baseline comparison. Figure 3.6 shows the computation time versus reward performance for each domain. In general, the computation time is directly proportional to the number of belief points, therefore the heuristic with the best performance is generally the one which can find a good solution with the least number of belief points.

In Hallway and Hallway2, it is unclear which of the four heuristics is best. The random heuristic—**RA**—appears slightly worse in the mid-range, and the greedy heuristic—**SSGA**—appears best in the early range. However, all four approaches need about the same amount of time to reach a good solution. Therefore, we conclude that in relatively small domains, the choice of heuristics does not seem to affect the performance much.

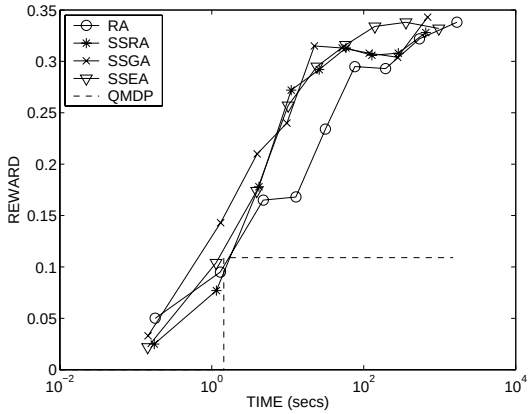
In the larger Tag domain however, the situation is different. With the random heuristic, the reward did not improve regardless of how many belief points were added, and

therefore we do not include it in the results. The exploratory action selection (SSEA) appears to be superior to using random or greedy action selection (SSRA, SSGA). These results suggest that the choice of belief points is crucial when dealing with large problems. SSEA seems more effective than the other heuristics at getting good coverage over the large dimensional beliefs featured in this domain.

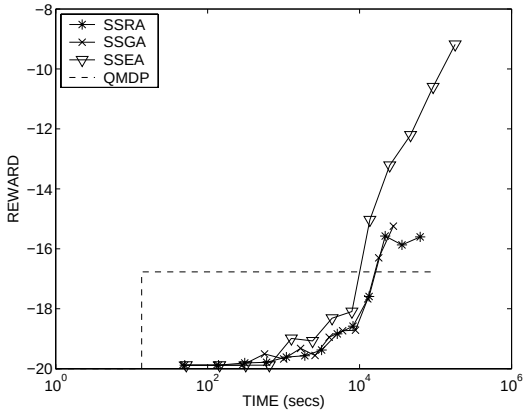
In terms of computational requirement, **SSGA** is the most expensive to compute. However the cost of the belief expansion step is generally negligible compared to the cost of the value update steps, therefore it seems best to use this superior (though more expensive) heuristic.



(a) Hallway



(b) Hallway2



(c) Tag

Figure 3.6. Belief expansion results

3.6. Applying Metric-Trees to PBVI

The point-based algorithm presented in this chapter is an effective approach for scaling up POMDP value function approximation. In PBVI, the value of each action sequence is expressed as an α-vector, and a key step in the value update (Eqn 3.2) requires evaluating many candidate α-vectors (set Γ) at each belief point (set B). This $B \times \Gamma$ (point-to-vector) comparison is usually implemented as a sequential search (exhaustively comparing $\alpha \cdot b$ for every $b \in B$ and every $\alpha \in \Gamma^{a,z}$) and is often the main bottleneck in scaling PBVI to larger domains.

The standard PBVI algorithm mostly ignores the geometrical properties of the belief simplex. In reality, belief points exist in a highly-structured metric space, and there is much to be gained from exploiting this property. For example, given the piecewise linearity and convexity of the value function, it is more likely that two nearby points will share similar values (and policies) than points that are far away. Consequently it could be much more efficient to evaluate an α-vector only once over a set of nearby points, rather than evaluating it by looking at each point individually.

Metric data structures offer a way to organize large sets of data points according to distances between the points (Friedman, Bengley, & Finkel, 1977). By organizing the data appropriately, it is possible to satisfy many different statistical queries over the elements of the set, without explicitly considering all points. The metric-tree (Uhlmann, 1991) in particular offers a very general approach to the problem of structural data partitioning. It consists of a hierarchical tree built by recursively splitting the set of points into spatially tighter subsets, assuming only that the distance between points is a metric.

This section presents an extension of the PBVI approach, in which a metric-tree structure is used to sort belief points spatially, and then to perform fast value function updates over groups of points. Searching over points organized in a metric-tree requires far fewer $B \times \Gamma$ comparisons than with an exhaustive search. This section describes the metric-tree formalism, and proposes a new algorithm for building and searching a metric-tree over belief points.

3.6.1. Building a Metric-Tree from Belief Points

The metric-tree is a hierarchical structure. We assume it has a binary branching structure, and define each node η by the following:

- a set of points, η_B;
- a center, η_c;

- a radius, η_r;
- a min-boundary vector, $\eta_{\min}$;
- a max-boundary vector, $\eta_{\max}$;
- a left child, η^L;
- a right child, η^R.

When building the tree, the top node is assumed to include all points. As the tree is refined, points are partitioned into smaller clusters of nearby points (where *smaller* implies both fewer points in η_B and a tighter radius η_r). Throughout the tree, for any given node η, all points η_B must fall within a distance η_r of the center η_c. The left and right children, η^L and η^R, point to further partitions in the data. The min / max boundary vectors, while not essential to building the tree, are used for fast statistical queries as described below. Assuming these components, we now describe how to build the tree.

Given a node η, the first step toward building children nodes η^L and η^R is to pick two candidate centers (one per child) at opposite ends of the region defined by the original node η:

$$\eta_c^L = \max_{b \in \eta_D} D(\eta_c, b) \tag{3.5}$$

$$\eta_c^R = \max_{b \in \eta_D} D(\eta_c^L, b). \tag{3.6}$$

The next step is to re-allocate the points in η_B between the two children (ties are broken randomly):

$$\begin{aligned} \forall b \in \eta_B : \quad \eta_B^L \leftarrow b \qquad & \text{if } D(\eta_c^L, b) < D(\eta_c^R, b) \\ \eta_B^R \leftarrow b \qquad & \text{if } D(\eta_c^L, b) > D(\eta_c^R, b). \end{aligned} \tag{3.7}$$

This reallocation of points between the left and right nodes resembles a single-step approximation to k-nearest-neighbor (k=2). It is fast to compute and generally effective. Other approaches can be used to obtain a better balanced tree, but seem to have little impact on the performance of the algorithm.

Finally, the center and radius of each child node can be updated to accurately reflect its set of points:

$$\eta_c^L = \text{Center}(\eta_B^L) \qquad \eta_c^R = \text{Center}(\eta_B^R) \tag{3.8}$$

$$\eta_r^L = \max_{b \in \eta_B^L} D(\eta_c^L, b) \qquad \eta_r^R = \max_{b \in \eta_B^R} D(\eta_c^R, b). \tag{3.9}$$

This procedure is repeated recursively until all leaf nodes contain a very small number of points (e. g. less than 5).

The general metric-tree algorithm allows a variety of ways to calculate the center and distance functions. This is generally defined as most appropriate for each instantiation of

the algorithm. For example, we could use one of the points as center. A more common choice is to calculate the centroid of the points:

$$\text{Center}(\eta_B) := \frac{\sum_{b \in \eta_B} b(s)}{n}, \quad \forall s \in S, \tag{3.10}$$

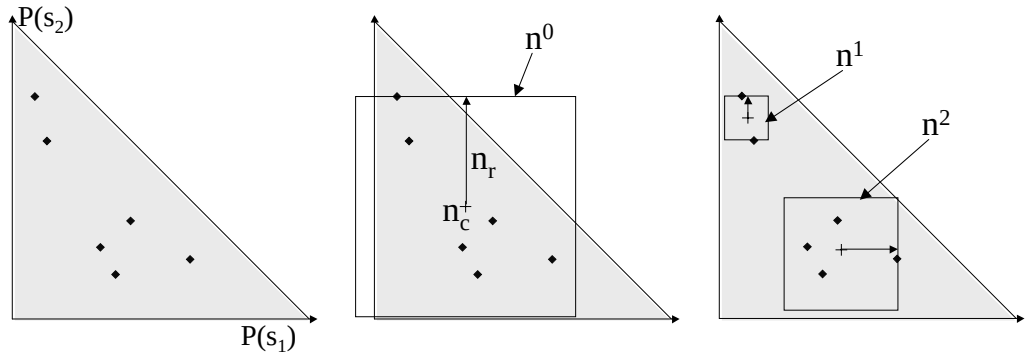
where n is the number of points in η_B. This is what we use, both because it is fast to compute, and because it appears to perform as well as other more complicated choices.

In terms of distance metric, there are a few important considerations. While the *magnitude* of the radius determines the *size* of the region enclosed by each node, the *type* of distance metric determines the *shape* of the region. We select the max-norm:

$$\begin{aligned} D(\eta_c, b) &:= \max_{b \in \eta_B} ||\eta_c - b||_\infty \\ &:= \max_{b \in \eta_B} \max_{s \in S} |\eta_c(s) - b(s)|, \end{aligned} \tag{3.11}$$

because it defines an $|S|$-dimensional hyper-cube of length $2 * \eta_r$. This allows for fast searching over the tree, as described in the next section.

Figure 3.7 gives a graphical illustration of the first two-levels of a tree, assuming a 3-state problem. Given the set of points shown in (a), the top-level node shown in (b) contains all points. The box has the appropriate center and radius as defined in Equations 3.10 and 3.11. When the tree is further refined, points are re-allocated accordingly to the left and right nodes, and the center and radius are updated for each. This is illustrated in Figure 3.7c. The full procedure for building a metric-tree over belief points is summarized in Table 3.7.



(a) Belief points. (b) Top-level node. (c) Level-1 left and right nodes

Figure 3.7. Example of building a tree

η=BUILD-TREE(B)	1
If $\|B\| < threshold$	2
Return NULL;	3
$\eta_B = B$	4
$\eta_c(s) = \frac{\sum_{b \in B} b(s)}{n}, \forall s \in S$	5
$\eta_r = \max_{b \in B} \|\eta_c - b\|_\infty$	6
$\eta_{\min}(s) = \min_{b \in \eta_B} b(s), \forall s \in S$	7
$\eta_{\max}(s) = \max_{b \in \eta_B} b(s), \forall s \in S$	8
$b_L = \text{argmax}_{b \in B} \|\eta_c - b\|_\infty$	9
$b_R = \text{argmax}_{b \in B} \|b_L - b\|_\infty$	10
$B_L = \emptyset$	11
$B_R = \emptyset$	12
For each point $b \in B$	13
If $\|b_L - b\|_\infty < \|b_R - b\|_\infty$	14
$B_L \leftarrow b$	15
Else	16
$B_R \leftarrow b$	17
End	18
η^L = BUILD-TREE(B_L)	19
η^R = BUILD-TREE(B_R)	20
$\eta = \{\eta_B, \eta_c, \eta_r, \eta_{\min}, \eta_{\max}, \eta^L, \eta^R\}$	21
Return η	22

Table 3.7. Algorithm for building a metric-tree over belief points

As mentioned in the very beginning of this section, there are additional statistics that we also store about each node, namely the boundary vectors $\eta_{\min}$ and $\eta_{\max}$. For a given node η containing data points η_B, we compute $\eta_{\min}$ and $\eta_{\max}$, the vectors containing respectively the min and max belief in each dimension:

$$\eta_{\min}(s) = \min_{b \in \eta_B} b(s), \forall s \in S \qquad \eta_{\max}(s) = \max_{b \in \eta_B} b(s), \forall s \in S. \tag{3.12}$$

Unlike the center and radius, which are required in order to build the tree, $\eta_{\min}$ and $\eta_{\max}$ are not essential to the definition of metric-trees, but rather are specific to using trees in the context of belief-state planning. More specifically, they are necessary to evaluate α-vectors over regions of the belief simplex. This is the topic discussed in the next section.

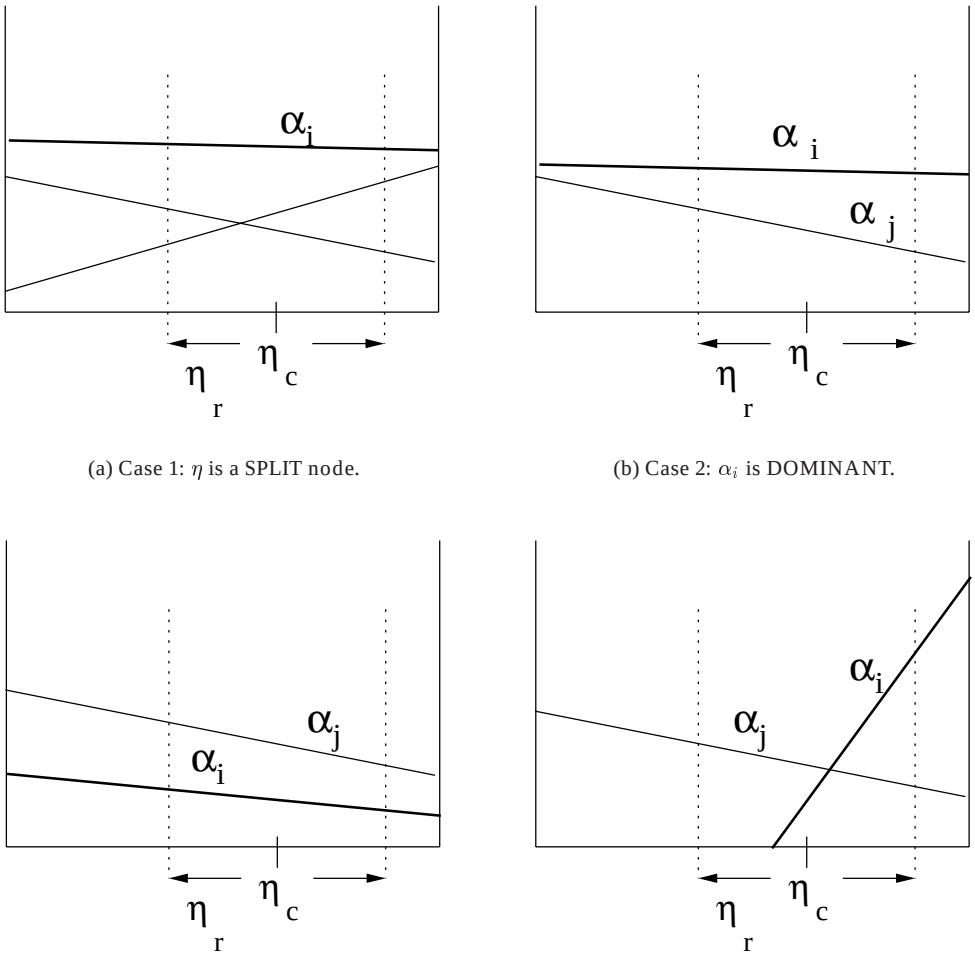
3.6.2. Searching over Sub-Regions of the Simplex

Once the tree is built, it can be used for fast statistical queries. In our case, the goal is to compute $\text{argmax}_{\alpha \in \Gamma^{a,z}} (\alpha \cdot b)$ for all belief points. To do this, we consider the α-vectors one at a time, and for each one decide whether a new candidate α_i is better than any of

the previous vectors $\{\alpha_0 \ldots \alpha_{i-1}\}$. With the belief points organized in a tree, we can often assess this quantity over sets of points by consulting a high-level node η, rather than by assessing it for each belief point separately.

We start at the root node of the tree. There are four different situations we can encounter as we traverse the tree:

1. no single previous α-vector is best for all beliefs below the current node (Fig. 3.8a),
2. the newest vector α_i dominates the previous best vector α_j (Fig. 3.8b),
3. the newest vector α_i is dominated by the best vector α_j (Fig. 3.8c),
4. the newest vector α_i partially dominates the previous best vector α_j (Fig. 3.8d).



(a) Case 1: η is a SPLIT node.

(b) Case 2: α_i is DOMINANT.

(c) Case 3: α_i is DOMINATED.

(d) Case 4: α_i is PARTIALLY DOMINANT.

Figure 3.8. Evaluation of a new vector α at a node η for a 2-state domain

In the first case, we proceed to the children of the current node without performing any test on the current node. In the other three cases there is a single dominant α-vector at the current node, and we need to perform a test to determine which of the three cases is in effect. If we can prove that α_i dominates (Case 2) or is dominated by (Case 3) the previous one, we can prune the search and avoid checking the current node's children; otherwise (Case 4) we must check the children recursively.

We therefore require an efficient test to determine whether one vector, α_i, dominates another, α_j, over the belief points contained within a node. The test must be conservative: it must never erroneously say that one vector dominates another. It is acceptable for the test to miss some pruning opportunities. The consequence is an increase in run-time as we check more nodes than necessary, therefore this is best avoided whenever possible.

Consider $\bar{\alpha} = (\alpha_i - \alpha_j)$. The test we seek must check whether $\bar{\alpha} \cdot b$ is positive or negative at every belief sample b under the current node. All positive means that α_i dominates α_j (Case 2), all negative the reverse (Case 3), and mixed positive and negative means that neither dominates the other (Case 4).

We cannot check $\bar{\alpha} \cdot b$ at every point, since this effectively renders the tree useless. Instead, we test whether $\bar{\alpha} \cdot b$ is positive or negative *over a convex region* R which includes all of the belief points in the current node. The goal is to find the smallest possible convex region, since this will maximize pruning. On the other hand, the region must be sufficiently simple that the test can be carried out efficiently.

We consider four types of region, as illustrated in Figure 3.9:

(a) axis-parallel bounding box defined by $\eta_{\min}(s) \leq b(s) \leq \eta_{\max}(s), \forall s \in S$;
(b) sub-simplex defined by $b(s) \geq \eta_{\min}(s), \forall s \in S$ and $\sum_{s \in S} b(s) = 1$;
(c) inverted sub-simplex defined by: $b(s) \leq \eta_{\max}(s), \forall s \in S$ and $\sum_{s \in S} b(s) = 1$;
(d) multi-sided box defined by the intersection of both sub-simplices defined by: $\eta_{\min}(s) \leq b(s) \leq \eta_{\max}(s), \forall s \in S$ and $\sum_{s \in S} b(s) = 1$.

Let $\bar{b}$ denote a convex region. Then for each of these regions, we can check whether $\bar{\alpha} \cdot \bar{b}$ is positive or negative in time $O(d)$ (where d=#states). For the box (Fig. 3.9a), which is the simplest of the regions, we can check each *dimension* independently as described in Table 3.8. For the two simplices (Figs 3.9b, 3.9c), we can check each *corner* exhaustively as described in Tables 3.9 and 3.10 respectively.

For the last shape (Fig. 3.9d), maximizing with respect to b is the same as computing δ such that $b(s) = \eta_{\min}(s)$ if $\bar{\alpha}(s) < \delta$ and $b(s) = \eta_{\max}(s)$ if $\bar{\alpha}(s) > \delta$. We can find δ in

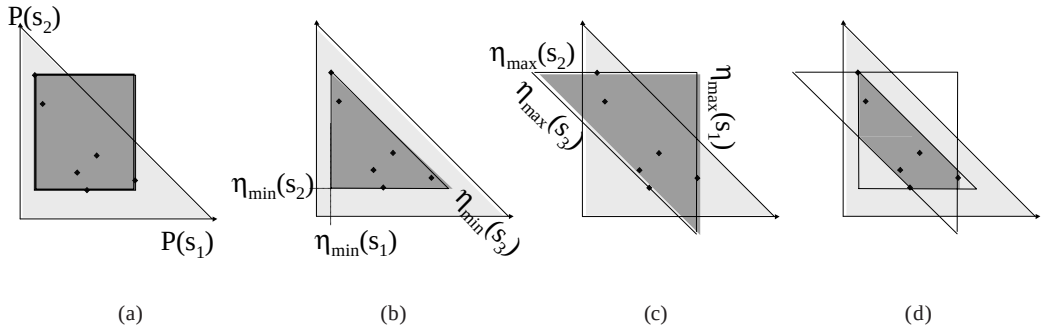


Figure 3.9. Possible convex regions over subsets of belief points for a 3-state domain

expected time $O(d)$ using a modification of the median-find algorithm (Hoare, 1961). The implementation for this last test is described in Tables 3.11 and 3.12.

While all regions can be checked in $O(d)$ expected time, in practice not all algorithms are equally fast. In particular, checking Region 4 (Fig. 3.9d) for each node tends to be significantly slower than the others. While the smaller search region means less searching in the tree, this is typically not sufficient to outweigh the larger per node cost. Empirical results show that simultaneously checking the corners of regions (b) and (c) and then taking the tightest bounds provides the fastest algorithm. This is the approach used to generate the results presented in the next section. It is summarized in Table 3.13.

$type$=CHECK-BOX(η, $\bar{\alpha}$)	1
$valueMin$=$\sum_{s \in S} \bar{\alpha}(s) * \eta_{\min}(s)$	2
$valueMax$=$\sum_{s \in S} \bar{\alpha}(s) * \eta_{\max}(s)$	3
If $valueMax \leq 0$	4
$type$=DOMINATED	5
Else If $valueMin > 0$	6
$type$=DOMINANT	7
Else	8
$type$=PARTIALLY DOMINANT	9
Return $type$	10

Table 3.8. Algorithm for checking vector dominance over region 1

$type$ =CHECK-SIMPLEX-UP(η, $\bar{\alpha}$)	1
$value = \sum_{s \in S} \bar{\alpha}(s) * \eta_{\min}(s)$	2
$valueMin = value + (\text{argmin}_{s \in S}\, \bar{\alpha}(s)) * (1.0 - \sum_{s \in S} \eta_{\min}(s))$	3
$valueMax = value + (\text{argmax}_{s \in S}\, \bar{\alpha}(s)) * (1.0 - \sum_{s \in S} \eta_{\min}(s))$	4
If $valueMax \leq 0$	5
$type$ =DOMINATED	6
Else If $valueMin > 0$	7
$type$=DOMINANT	8
Else	9
$type$ =PARTIALLY DOMINANT	10
Return $type$	11

Table 3.9. Algorithm for checking vector dominance over region 2

$type$ =CHECK-SIMPLEX-DOWN(η, $\bar{\alpha}$)	1
$value = \sum_{s \in S} \bar{\alpha}(s) * \eta_{\max}(s)$	2
$valueMin = value + (\text{argmin}_{s \in S}\, \bar{\alpha}(s)) * (1.0 - \sum_{s \in S} \eta_{\max}(s))$	3
$valueMax = value + (\text{argmax}_{s \in S}\, \bar{\alpha}(s)) * (1.0 - \sum_{s \in S} \eta_{\max}(s))$	4
If $valueMax \leq 0$	5
$type$ =DOMINATED	6
Else If $valueMin > 0$	7
$type$ =DOMINANT	8
Else	9
$type$ =PARTIALLY DOMINANT	10
Return $type$	11

Table 3.10. Algorithm for checking vector dominance over region 3

$Corner$=FIND-CORNER(η, v, $Corner$)	1
If $v > 1.0$	1
$dir = +1$	1
Else	1
$dir = -1$	1
$J = S$, the set of states	1
$value = \|v - 1.0\|$	1
While $value > 0.0$	1
$pivot$ =RAND(J)	1
$I = \emptyset$	1
$I' = \emptyset$	1
Forall $s \in J$	1
If $Corner(s) < Corner(pivot)$	1
If ($dir > 0$ AND $Corner(s) > \eta_{\min}(s)$) OR	1
($dir < 0$ AND $Corner(s) < \eta_{\max}(s)$)	1
I $\leftarrow s$	1
Else If $Corner(s) > Corner(pivot)$	1
$I' \leftarrow s$	1
End	1
$sum = \sum_{s \in I}(\eta_{\max}(s) - \eta_{\min}(s))$	1
If $sum > value$	1
$pivot$ =RAND(I)	1
$J = I$	1
Else	1
$I \leftarrow pivot$	1
Forall $s \in I$	1
If $\eta_{\max}(s) - \eta_{\min} > value$	1
$Corner(s) = Corner(s) - value$	1
$value = 0.0$	1
Else	1
$Corner(s) = \eta_{\min}$	1
$value = value - (\eta_{\max}(s) - \eta_{\min})$	1
End	1
$J = I'$	1
End	1
Return $Corner$	1

Table 3.11. Algorithm for finding corner in region 4

$type$=CHECK-SIMPLEX-INTERSECTION($\eta, \bar{\alpha}$)	1
For $s \in S$	1
If $\bar{\alpha}(s) > 0$	1
$c(s) = \eta_{\max}(s)$	1
Else	1
$c(s) = \eta_{\min}(s)$	1
End	1
$value = \sum_{s \in S} c(s)$	1
$Corner$ =FIND-CORNER($\eta, value, Corner$)	1
$valueMax = \sum_{s \in S} \bar{\alpha}(s) * Corner(s)$	1
For $s \in S$	1
If $\bar{\alpha}(s) > 0$	1
$c(s) = \eta_{\min}(s)$	1
Else	1
$c(s) = \eta_{\max}(s)$	1
End	1
$value = \sum_{s \in S} c(s)$	1
$Corner$ =FIND-CORNER($\eta, value, Corner$)	1
$valueMin = \sum_{s \in S} \bar{\alpha}(s) * Corner(s)$	1
If $valueMax \leq 0$	5
$type$ =DOMINATED	6
Else If $valueMin > 0$	7
$type$ =DOMINANT	8
Else	9
$type$ =PARTIALLY DOMINANT	10
Return $type$	11

Table 3.12. Algorithm for checking vector dominance over region 4

$type$=CHECK-SIMPLEX-BOTH($\eta, \bar{\alpha}$)	1
$value1 = \sum_{s \in S} \bar{\alpha}(s) * \eta_{\min}(s)$	2
$valueMin1 = value + (\text{argmin}_{s \in S}\, \bar{\alpha}(s)) * (1.0 - \sum_{s \in S} \eta_{\min}(s))$	3
$valueMax1 = value + (\text{argmax}_{s \in S}\, \bar{\alpha}(s)) * (1.0 - \sum_{s \in S} \eta_{\min}(s))$	4
$value2 = \sum_{s \in S} \bar{\alpha}(s) * \eta_{\max}(s)$	2
$valueMin2 = value + (\text{argmin}_{s \in S}\, \bar{\alpha}(s)) * (1.0 - \sum_{s \in S} \eta_{\max}(s))$	3
$valueMax2 = value + (\text{argmax}_{s \in S}\, \bar{\alpha}(s)) * (1.0 - \sum_{s \in S} \eta_{\max}(s))$	4
If $min(valueMax1, valueMax2) \leq 0$	5
$type$=DOMINATED	6
Else If $max(valueMin1, valueMin2) > 0$	7
$type$ =DOMINANT	8
Else	9
$type$ =PARTIALLY DOMINANT	10
Return $type$	11

Table 3.13. Final algorithm for checking vector dominance

3.6.3. Experimental Evaluation

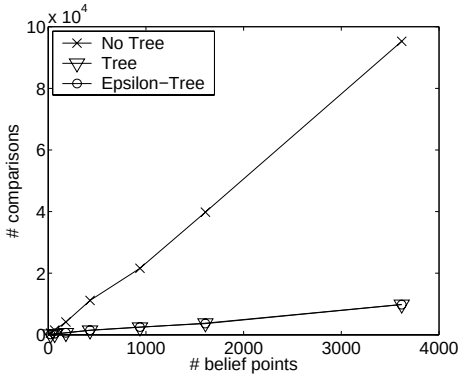
This section presents the results of simulation experiments conducted to test the effectiveness of the tree structure in reducing computational load. The results also serve to illustrate a few interesting properties of metric-trees when used in conjunction with point-based POMDP planning.

We first consider six well-known POMDP problems and compare the number of $B \times \Gamma$ (point-to-vector) comparisons required with and without a tree. The problems range in size from 4 to 870 states. Four of them—Hanks, SACI, Tiger-grid (a.k.a. Maze33), and Hallway—are described in (Cassandra, 1999). The Coffee domain is described in (Poupart & Boutilier, 2003). Tag was first proposed in (Pineau et al., 2003a) and is described in Section 3.5.2 above. While all these problems have been successfully solved by previous approaches, the goal here is to observe the level of speed-up that can be obtained by leveraging metric-tree data structures.

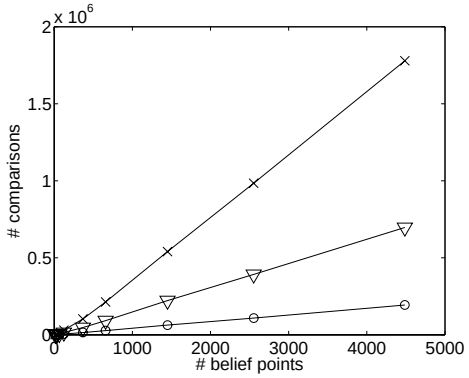
Figure 3.10(a)-(f) shows the number of $B \times \Gamma$ comparisons required, as a function of the number of belief points, for each of these problems. In Figure 3.11(a)-(b) we show the computation time (as a function of the number of belief points) required for two of the problems. In all cases, the *No-Tree* results were generated by applying the standard PBVI algorithm (Section 3.2). The *Tree* results (which count comparisons on both internal and leaf nodes) were generated by embedding the tree searching procedure described in Section 3.6.2 within the same point-based POMDP algorithm. For some of the problems, we also show performance using an ϵ-tree, where the test for vector dominance can reject (i. e. declare α_i is *dominated*, Figure 3.8c) a new vector that is within ϵ of the current best vector.

These results show that, in various proportions, the tree can cut down on the number of comparisons, and thus reduce POMDP computational load. The ϵ-tree is particularly effective at reducing the number of comparisons in some domains (e. g. SACI, Tag). The much smaller effect shown in the other problems may be attributed to a poorly tuned ϵ (we used $\epsilon = 0.01$ in all experiments). The question of how to set ϵ such that we most reduce computation, while maintaining good control performance, tends to be highly problem-dependent.

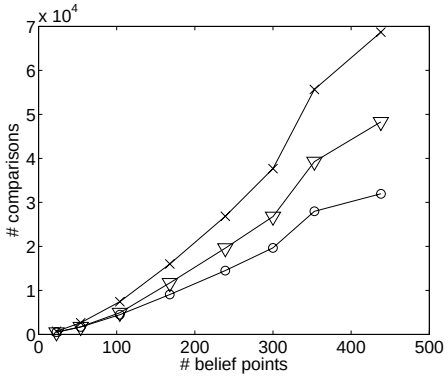
In keeping with other metric-tree applications, our results show that computational savings increase with the number of belief points. It is interesting to see the trees paying off with relatively few data points (most applications of KD-trees start seeing benefits



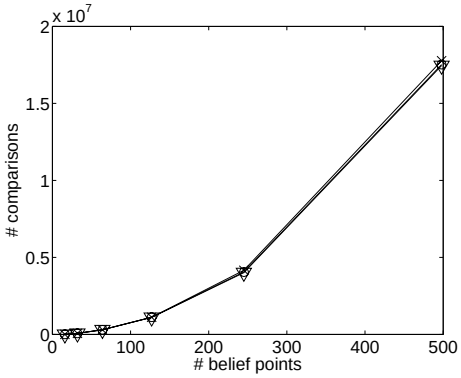
(a) Hanks, $|S|$=4



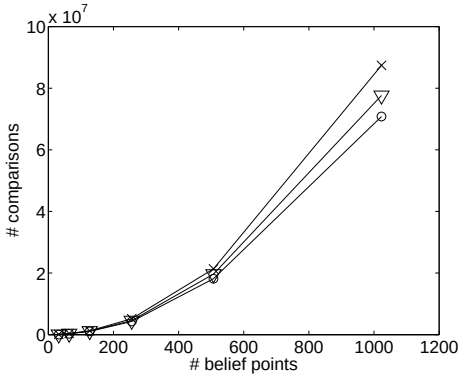
(b) SACI, $|S|$=12



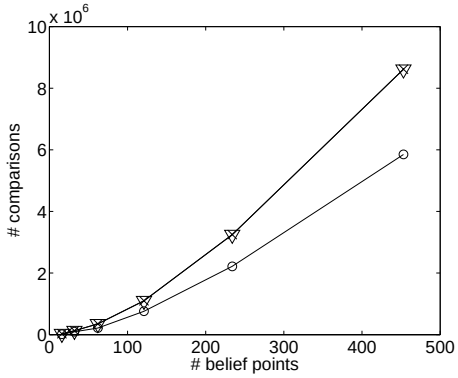
(c) Coffee, $|S|$=32



(d) Tiger-grid, $|S|$=36

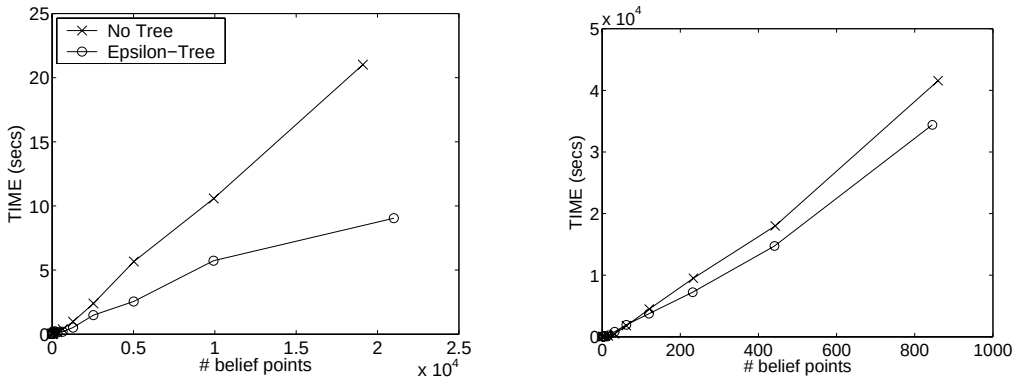


(e) Hallway, $|S|$=60



(f) Tag, $|S|$=870

Figure 3.10. Number of $B \times \Gamma$ comparisons with and without metric-trees



(a) SACI, $|S|$=12 (b) Tag, $|S|$=870

Figure 3.11. Planning time for PBVI algorithm with and without metric-tree

with 1000+ data points). This may be partially attributed to the compactness of our convex test region (Fig. 3.9d), and to the fact that we do not search on split nodes (Fig. 3.8a); however, it is most likely due to the nature of our search problem: many α-vectors are accepted / rejected before visiting *any* leaf nodes, which is different from other metric-tree applications. We are particularly encouraged to see trees having a noticeable effect with very few data points because, in some domains, good control policies can also be extracted with few data points.

We notice that the effect of using trees is negligible in some mid-size problems (e. g. Tiger-grid), while still pronounced in others of equal or larger size (e. g. Coffee, Tag). This is likely due to the intrinsic dimensionality of each problem. For example, the coffee domain is known to have an intrinsic dimensionality of 7 (Poupart & Boutilier, 2003). And while we do not know the intrinsic dimensionality of the Tag domain, many other robot applications have been shown to produce belief points that exist in sub-dimensional manifolds (Roy & Gordon, 2003). Metric-trees often perform well in high-dimensional datasets with low intrinsic dimensionality; this also appears to be true of metric-trees applied to vector sorting. While this suggests that our current algorithm is not as effective in problems with intrinsic high-dimensionality, a slightly different tree structure or search procedure could be more effective in those cases.

3.7. Related Work

There are several approximate value iteration algorithms that are related to PBVI. In particular, there are many grid-based methods that iteratively update the values of discrete

belief points, and thus are quite similar to PBVI. These methods differ in how they partition the belief space into a grid, and in how they update the value function.

Some methods update only the value at each point (Brafman, 1997; Zhou & Hansen, 2001). More similar to PBVI are those approaches that update both the value and gradient at each grid point (Lovejoy, 1991a; Hauskrecht, 2000; Poon, 2001). The actual point-based value update is essentially the same between all of these approaches and PBVI. However the overall algorithms differ in a few important aspects.

Whereas Poon only accepts updates that increase the value at a grid point (requiring special initialization of the value function), and Hauskrecht always keeps earlier α-vectors (causing the set to grow too quickly), PBVI does not have these restrictions.

An important contribution of PBVI is the theoretical guarantees it provides: the theoretical properties described in Section 3.3 are more widely applicable and provide stronger error bounds than what was available prior to this work.

In addition, PBVI has a powerful approach to belief point selection. Many earlier algorithms suggested using random beliefs, or (like Poon's and Lovejoy's) require the inclusion of a large number of fixed beliefs such as the corners of the probability simplex. In contrast, PBVI favors selecting only reachable beliefs (and in particular those belief points that improve its error bounds as quickly as possible). While both Hauskrecht and Poon did consider using stochastic simulation to generate new points, neither found simulation to be superior to random point placements. We hypothesize this may be due to the smaller size of their test domains. Our empirical results clearly show that with a large domain, such as Tag, PBVI's belief-selection is an important factor in the algorithm's performance.

Finally, a very minor difference is the fact that PBVI builds only $|A||Z||\Gamma_{t-1}|$ projections, versus $|A||Z||B|$ (Poon, 2001; Zhang & Zhang, 2001), and thus is faster whenever multiple points support the same α-vector.

The metric-tree approach to belief point searching and sorting is a novel use of this data structure. Metric-trees have been used in recent years for other similar $M \times N$ comparison problems that arise in statistical learning tasks. In particular, instances of metric data structures such as KD-trees, ball-trees and metric-trees have been shown to be useful for a wide range of tasks (e. g. nearest-neighbor, kernel regression, mixture modeling), including some with high-dimensional and non-Euclidean spaces (Moore, 1999).

New approaches building directly on PBVI have been proposed subsequent to this work. This includes an algorithm Vlassis and Spaan (2004) in which point-based value updates are not systematically applied to all points at each iteration. Instead, points are sampled randomly (and updated) until the value of all points has been improved; updating

the α-vector at one point often also improves the value estimate of other nearby points. This modification appreciably accelerates the basic PBVI algorithm for some problems.

3.8. Contributions

This chapter describes a new point-based algorithm for POMDP solving. The main contributions pertaining to this work are summarized in this section.

Anytime planning. PBVI alternates between steps of value updating and steps of belief point selection. As new points are added, the solution improves, at the expense of increased computational time. The trade-off can be controlled by adjusting the number of points. The algorithm can be terminated either when a satisfactory solution is found, or when planning time is elapsed.

Exploration. PBVI proposed a new exploration-based point selection heuristic. The heuristic uses a reachability analysis with stochastic observation sampling to generate belief points that are both reachable and likely. In addition, distance between points is considered to increase coverage of the belief simplex.

Bounded error. PBVI is guaranteed to have bounded approximation error. The error is directly reduced by the addition of belief points. In practice, the bound is often quite loose. However, improvements in the bound can indicate improvements in solution quality.

Improved empirical performance. PBVI has demonstrated the ability to reduce planning time for a number of well-known POMDP problems, including Tiger-grid, Hallway, and Hallway2. By operating on a set of discrete points, PBVI can perform polynomial-time value updates, thereby overcoming the curse of history that paralyzes exact algorithms. The exploratory heuristic used to select points allows PBVI to solve large problems with fewer belief points than previous approaches.

New problem domain. PBVI was applied to a new POMDP planning domain (Tag), for which it generated an approximate solution that outperformed baseline algorithms QMDP and Incremental Pruning. This new domain has since been adopted as a test case for other algorithms (Vlassis & Spaan, 2004; Poupart & Boutilier, 2004).

Metric-tree extension. A metric-tree extension to PBVI was developed, which sorts and searches through points according to their spatial distribution. This allows the modified PBVI to search over sub-regions of the belief simplex, rather than over individual points, thereby accelerating planning over the basic PBVI algorithm.

3.9. Future Work

While PBVI has demonstrated the ability to solve problems on the order of 10^3 states, many real-world domains far exceed this. In particular, it is not unusual for a problem to

be expressed through a number of multi-valued state *features*, in which case the number of states grows exponentially with the number of features. This is of concern because each belief point and each α-vector has dimensionality $|S|$ (where $|S|$ is the number of states) and all dimensions are updated simultaneously. This is an important issue to address to improve the scalability of point-based value approaches.

There are various existing attempts at overcoming the curse of dimensionality, which are discussed in Section 2.2.5. Some of these, in particular the exact compression algorithm of (Poupart & Boutilier, 2003), can be combined with PBVI. However, preliminary experiments in this direction have yielded little performance improvement. Other techniques—e. g. (Roy & Gordon, 2003)—cannot be combined with PBVI without compromising its theoretical properties (as discussed in Section 3.3). The challenge therefore is to devise function-approximation techniques that both reduce the dimensionality effectively, while maintaining the convexity properties of the solution.

A secondary (but no less important) issue concerning the scalability of PBVI pertains to the number of belief points necessary to obtain a good solution. While problems addressed thus far can usually be solved with $O(|S|)$ number of belief points, this need not be true. In the worse case, the number of belief points necessary may be exponential (in the plan length). The work described in this thesis proposes a good heuristic (called SSEA) for generating belief points, however this is unlikely to be the definitive answer to belief point selection. In practical applications, a carefully engineered trade-off between exploratory (i. e. SSEA) and greedy (i. e. SSGA) action selection may yield better results. An interesting alternative may be to add those new reachable belief points that have the largest estimated approximation error. In more general terms, this relates closely to the well-known issue of exploration policies, which arises across a wide array of problem-solving techniques.

CHAPTER 4

A Hierarchical Approach to POMDPs

It is well-known in the AI community that many solution techniques can be greatly scaled by appropriately leveraging structural information. A very common way to use structural information is to follow a divide-and-conquer scheme, where a complex (structured) problem is decomposed into many smaller problems that can be more easily addressed and whose solution can be recombined into a global one.

Until recently, there was no such divide-and-conquer approach for POMDPs. In this chapter, we present a new algorithm for planning in structured POMDPs, which is called PolCA+ (for **Pol**icy-**C**ontingent **A**bstraction). It 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. This is the *policy-contingent* aspect of the algorithm (thus the name). At all levels of the hierarchy, subtasks need only consider a reduced action, state, and observation space. This structural decomposition leads to appreciable computational savings, since local policies can be quickly found for each subtask.

The chapter begins with a discussion of the structural assumptions proper to PolCA+. Section 4.2 then presents the new algorithm in the special case of fully observable MDPs. This version is called PolCA, to avoid confusion with the more general POMDP version known as PolCA+. We differentiate between the two cases because PolCA possesses some important theoretical properties which do not extend to PolCA+; these are discussed in Section 4.2.4. Section 4.3 presents the full PolCA+ algorithm for structured POMDP planning. It also contains empirical results demonstrating the usefulness of the approach on a range of problems.

While this chapter presents a novel approach for handling hierarchical POMDPs, there exists a large body of work dealing with the fully observable case, namely the hierarchical MDP. Of particular interest are MAXQ (Dietterich, 2000), HAM (Parr & Russell, 1998), ALisp (Andre & Russell, 2002), and options (Sutton, Precup, & Singh, 1999), whose objectives and structural assumptions are very similar to PolCA's. Section 4.4 offers an in-depth discussion of the differences and similarities between these and PolCA.

4.1. Hierarchical Task Decompositions

The key concept in this chapter is that one can reduce the complexity of POMDP planning by hierarchically decomposing a problem. Assuming the overall task is such that it naturally maps into a hierarchy of subtasks, then a planner should take advantage of that structure by solving individual subtasks separately, rather than jointly. The computational gains arise from the fact that solving N subtasks can be more efficient that solving a single task that is N times as large.

The fundamental assumption behind hierarchical POMDPs is that the task exhibits natural structure, and that this structure can be expressed by an *action hierarchy*. To better understand the concept of action hierarchy, it is useful to introduce a simple example.

EXAMPLE 4.1.1. *Consider a vacuuming robot that lives in a two-room environment (Fig. 4.1), one of which (room2) has to be kept clean. The robot can move deterministically between the rooms, it can also vacuum, as well as wait (presumably when the vacuuming is done). Whenever the robot vacuums room2, there is a reasonable chance ($Pr = 0.5$) that as a result of this the room will be clean, there is also the possibility ($Pr = 0.4$) that the room will not be clean, and there is a small probability ($Pr = 0.1$) that the robot will accidentally leave the room. The state space is expressed through two fully-observable binary variables: $s_{robot} = \{room1, room2\}$ describes the robot's current position, $s_{status} = \{dirty, clean\}$ describes the current state of room2. For example, Figure 4.1 illustrates state $s_{robot} = room1$, $s_{status} = dirty$. The action set contains four actions: $A = \{left, right, vacuum, wait\}$. The state-to-state transitions are indicated in Figure 4.2 (deterministic self-transitions are not shown). The robot receives a reward of -1 for applying any action, with the exception of the $wait$ action, which is free whenever the room is clean: $R(s_{status} = clean, a = wait) = 0$.*

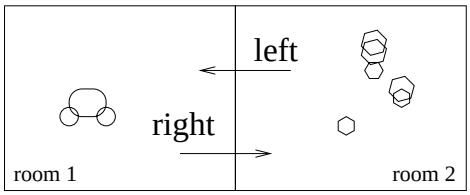


Figure 4.1. Robot vacuuming task

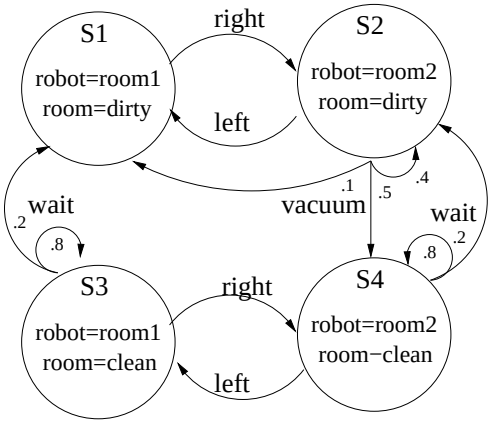


Figure 4.2. Robot vacuuming task transition model

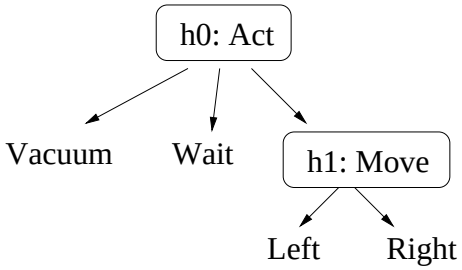


Figure 4.3. Robot vacuuming task hierarchy

As shown in Figure 4.3, an action hierarchy is represented by a tree. At the top level, the root of the tree represents the overall task, as defined by the POMDP (e. g. the Act task in Fig. 4.3). At the bottom level, each individual leaf corresponds to a *primitive action* $a \in A$ (e. g. $A = \{Left, Right, Vacuum, Wait\}$). These primitive actions represent the lowest level of policy choice. In between, all internal nodes in the tree represent *subtasks* (e. g. $Move$). Subtasks, denoted h, are defined over limited sets of other subtasks and / or primitive actions, as specified by their children in the tree. For example subtask $h1 : Move$ has action set $A_{Move} = \{Left, Right\}$, and subtask $h0 : Act$ has action set $A_{Act} = \{Move, Vacuum, Wait\}$.

It is worth noting that 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. This is the case of $Move$, which appears in the action set of subtask Act, but is also a subtask unto itself.

The hierarchical approach discussed in this chapter depends on three important assumptions related to domain knowledge. First and foremost, it assumes that the hierarchical subtask decomposition is provided by a designer. This constitutes prior knowledge brought to bear on the domain to facilitate planning. This assumption is consistent with prior work on hierarchical MDPs (Parr & Russell, 1998; Sutton et al., 1999; Dietterich, 2000; Andre & Russell, 2002), as discussed near the end of this chapter.

Second, each subtask in the hierarchy is assumed to have local (non-uniform) reward. This is common in hierarchical MDPs (Dietterich, 2000), and is necessary in order to optimize a local policy for each subtask. In general, the local reward is equal to the true reward $R(s, a)$. In subtasks where all available actions have equal reward (over all states), we must add a pseudo-reward to specify the desirability of satisfying the subgoal. A common choice is to let the states that satisfy the subtask's goal have a pseudo-reward of 0. This is the case of the $Move$ subtask above, where both $Left$ and $Right$ have reward $R = -1$. In this case, we propose $\bar{R}_{Move}(s_{robot} = room2) = 0$ since that is the subtask goal. The Act subtask on the other hand does not require a pseudo-reward since $Wait$ and $Vacuum$ have different reward signals. Pseudo-rewards do not alter the reward received at execution time. They are simply used as shaping constraints during policy optimization. They are unnecessary in most multi-goal robot problems where each subtask contains one or many different goals (e. g. the Nursebot domain described in Chapter 5). However, they are needed for some multi-step single-goal domains.

Finally, PolCA+ assumes a known POMDP model of the original flat (non-hierarchical) problem. In the case of the robot vacuuming task, the dynamics of the domain are illustrated in Figure 4.2. In general, the model can be estimated from data or provided by a designer. While this is a departure from reinforcement-learning methods, it is consistent with most work on POMDP approximations. More importantly, it greatly contributes to the effectiveness of PolCA+ since it allows us to automatically discover state and observation abstraction for each subtask. The state and observation reduction follows directly from the action reduction, and therefore can be discovered automatically as an integral part of PolCA+. This leads to important computational advantage, without any performance loss, since the value of a given subtask often depends only on a subset of all state / observation features.

Getting back to the example above, it seems obvious that the $Move$ subtask need only consider the s_{robot} state feature (since the effects of both $Left$ and $Right$ are independent of the room's state of cleanliness). At first glance, both state features appear useful for the top-level subtask; this is formally determined below when we discuss PolCA+'s procedure for automatic state abstraction.

The notion of task hierarchies raises two fundamental questions: how can we exploit task hierarchies in POMDP planning, and how can we combine many subtask-specific plans when it comes time for 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 an important issue, namely how can we tie in the value functions of low-level subtasks when optimizing high-level subtasks. 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. These questions will be examined in depth in this chapter.

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 known as PolCA, which shares some similarities with earlier hierarchical MDP algorithms. In the subsequent section, we present the complete (and more general) POMDP formulation known as PolCA+.

4.2. PolCA: A Hierarchical Approach to 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.

For each subtask, the goal of hierarchical POMDP planning is to optimize a corresponding *local policy*:

DEFINITION 4.2.1. *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.

PolCA relies on a set of formal structural assumptions. First, that there exists a task graph H, where each leaf node represents a *primitive* action a, from the original MDP action set A. Each internal node has the dual role of representing both a distinct subtask (we use notation h for a subtask) whose action set is defined by its immediate children in the hierarchy, as well as an *abstract* action (we use a bar, as in $\bar{a}$, to denote abstract actions) in the context of the above-level subtask. A subtask h is formally defined by

- $A_h = \{\bar{a}_j, \ldots, a_p, \ldots\}$, the set of actions that are allowed in subtask h. Based on the hierarchy, there is one action for each immediate child of h.
- $\bar{R}_h(s, a)$, the local reward function. Each subtask in the hierarchy must have local (non-uniform) reward in order to optimize a local policy.

4.2.1. Planning Algorithm

Table 4.1 describes our hierarchical MDP planning algorithm. The main function is called using the parameterized MDP model M as its first argument and the hierarchy H as its second argument. It computes the set of local policies (one per subtask) using four simple steps, each of which is explained in further detail below.

4.2.1.1. 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). In our approach, the hierarchy state can correspond to any of the internal nodes.

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

Table 4.1. Main PolCA planning function

The new state space $H \circ S$ 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 $H \circ S = \{h_0 \circ s_0, \ldots, h_0 \circ s_n, \ldots\ldots, h_m \circ s_0, \ldots, h_m \circ s_n\}$.

4.2.1.2. Step 2—*Set parameters*. The purpose of the second step is to appropriately translate the transition and reward parameters specified in $M = \{S, A, R, T\}$ to the structured problem representation. For any given subtask h, with state set $S = \{h \circ s_0, h \circ s_1, \ldots\}$ and action set $A_h = \{a_k, \ldots, \bar{a}_p, \ldots\}$ there are two cases to consider. For the primitive actions $\{a_k, \ldots\}$, it is sufficient to copy the original transition and reward parameters from the model in M. This is described in Equations 4.1-4.2. For the abstract actions $\{\bar{a}_p, \ldots\}$, which invoke lower-level subtasks $\{h_p, \ldots\}$, it is necessary to infer parameters based on the policy of the corresponding subtask. This is described in Equations 4.3-4.4.

CASE 1 - PRIMITIVE ACTIONS: $\forall a_k \in A_h, a_k \in A, \forall (s_i, s_j) \in S$:

$$T(h \circ s_i, a_k, h \circ s_j) = T(s_i, a_k, s_j) \quad (4.1)$$

$$R(h \circ s_i, a_k) = \bar{R}_h(s_i, a_k) \quad (4.2)$$

CASE 2 - ABSTRACT ACTIONS: $\forall \bar{a}_p \in h, \forall (s_i, s_j) \in S$:

$$T(h \circ s_i, \bar{a}_p, h \circ s_j) = T(s_i, \pi^*_{h_p}(s_i), s_j), \quad (4.3)$$

$$R(h \circ s_i, \bar{a}_p) = \bar{R}_h(s_i, \pi^*_{h_p}(s_i)) \quad (4.4)$$

Equations 4.3-4.4 depend on $\pi^*_{h_p}$—the final policy of subtask h_p—which enforces the policy-contingent aspect of PolCA. Since parameter setting for $\bar{a}_p$ occurs *after* h_p has been solved, state abstraction in h needs to preserve sufficient information to represent the final policy $\pi^*_{h_p}$, but not *any* policy π_{h_p}.

4.2.1.3. Step 3—*Minimize states*. The goal of this step is to learn a minimization function $f_h(s)$ mapping individual states to clusters of states. State abstraction (also called state clustering or 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 existing MDP model minimization algorithm (Dean & Givan, 1997). The

algorithm has three parts. In Part I, a set of overly-general state 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. The formal algorithm is as follows.

To infer $f_h(s) \rightarrow c$, the function mapping states $\{h \circ s_0, h \circ s_1, \dots\}$ to the (expanding) set of clusters $S_h = \{c_0, c_1, \dots\}$:

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

$$R(h \circ s_i, a) = R(h \circ s_j, a), \forall a \in A_h. \tag{4.5}$$

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

$$\sum_{s' \in c'} T(h \circ s_i, a, h \circ s') = \sum_{s' \in c'} T(h \circ s_j, a, h \circ s'), \quad \forall (s_i, s_j) \in c, \forall c' \in S_h, \forall a \in A_h. \tag{4.6}$$

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

$$c \rightarrow \{c_k, c_{k+1}, \dots\}, \tag{4.7}$$

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

Once Part II returns no unstable cluster, the algorithm is terminated. MDP model parameters are then re-expressed over clusters:

$$T(h \circ c_i, a, h \circ c_j) = \sum_{s' \in c_j} T(h \circ s, a, h \circ s'), \forall a \in A_h \tag{4.8}$$

$$R(h \circ c_i, a) = R(h \circ s, a), \text{ for any } s \in c_i, \forall a \in A_h. \tag{4.9}$$

This algorithm exhibits the following desirable properties which were initially discussed in (Dean & Givan, 1997; Dean, Givan, & Leach, 1997):

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

As an aside, it is also possible to compute an abstraction of $f_h(s, a)$ (abstracting over $Q(s, a)$), instead of just $f_h(s)$ (which abstracts over $V(s)$). This is often used when hand-crafting abstraction functions in hierarchical MDPs (Dietterich, 2000; Andre & Russell, 2002). 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. 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.

4.2.1.4. Step 4—*Solve subtask*. The purpose of Step 4 is to learn the value function and policy for subtask h. The state clustering step described in Step 3 ensures that all states in a cluster share the same value (see Property 1 above). Therefore, we can apply dynamic programming updates over clusters:

$$V(h \circ c_i) = \max_{a \in A_h} [R(h \circ c_i, a) + \gamma \sum_{c_j \in S_h} T(h \circ c_i, a, h \circ c_j) V(h \circ c_j)]. \tag{4.10}$$

The repeated application of this value update is guaranteed to converge. The final value function solution is contained in the value function of the top subtask: $V^*(h_0 \circ s)$. In practice, Steps 3 and 4 are often interleaved.

4.2.2. PolCA Planning: An example

We can now revisit the robot vacuuming domain (Example 4.1.1) and go through the four steps of PolCA planning for this simple case. The model and hierarchy are reproduced at the top of Figure 4.4.

In Step 1, the structured state space augments the 4-state problem by adding a subtask identifier variable $h = \{Act, Move\}$. Moving on to Step 2, subtask h_{Move} is considered first because it is the lower-level one. It starts by undergoing parameterizing, where parameters conditioned on actions $\{L = Left, R = Right\}$ are translated from the original model to this subtask. Next, the model minimization procedure reduces the state for h_{Move} from four states to two clusters, because state feature s_{status} can be safely ignored. Finally, value iteration yields the local policy for this subtask, where as expected $\pi(h_{Move} \circ s) = Right, \forall s \in S$.

PolCA can now move on to subtask h_{Act}. In the parameterization step, PolCA transposes parameters from the original model to describe actions $\{V = Vacuum, W = Wait\}$ and uses the policy of h_{Move} (illustrated in the bottom left corner) to model abstract action $\{\bar{M} = Move\}$. In Step 3, clustering on subtask h_{Act} leaves the state space intact, meaning that all four states are necessary. In Step 4, value iteration yields the local policy for h_{Act}, as illustrated in the bottom right corner of Figure 4.4. As expected, when the robot is in room 1 and room 2 is dirty, the robot moves right. When in room 2, the robot applies the vacuum action until it becomes clean. Whenever the room is clean, the robot simply waits.

Once we have generated policies for both subtasks, we have achieved a full hierarchical planning solution for this problem. This concludes our discussion of this example.

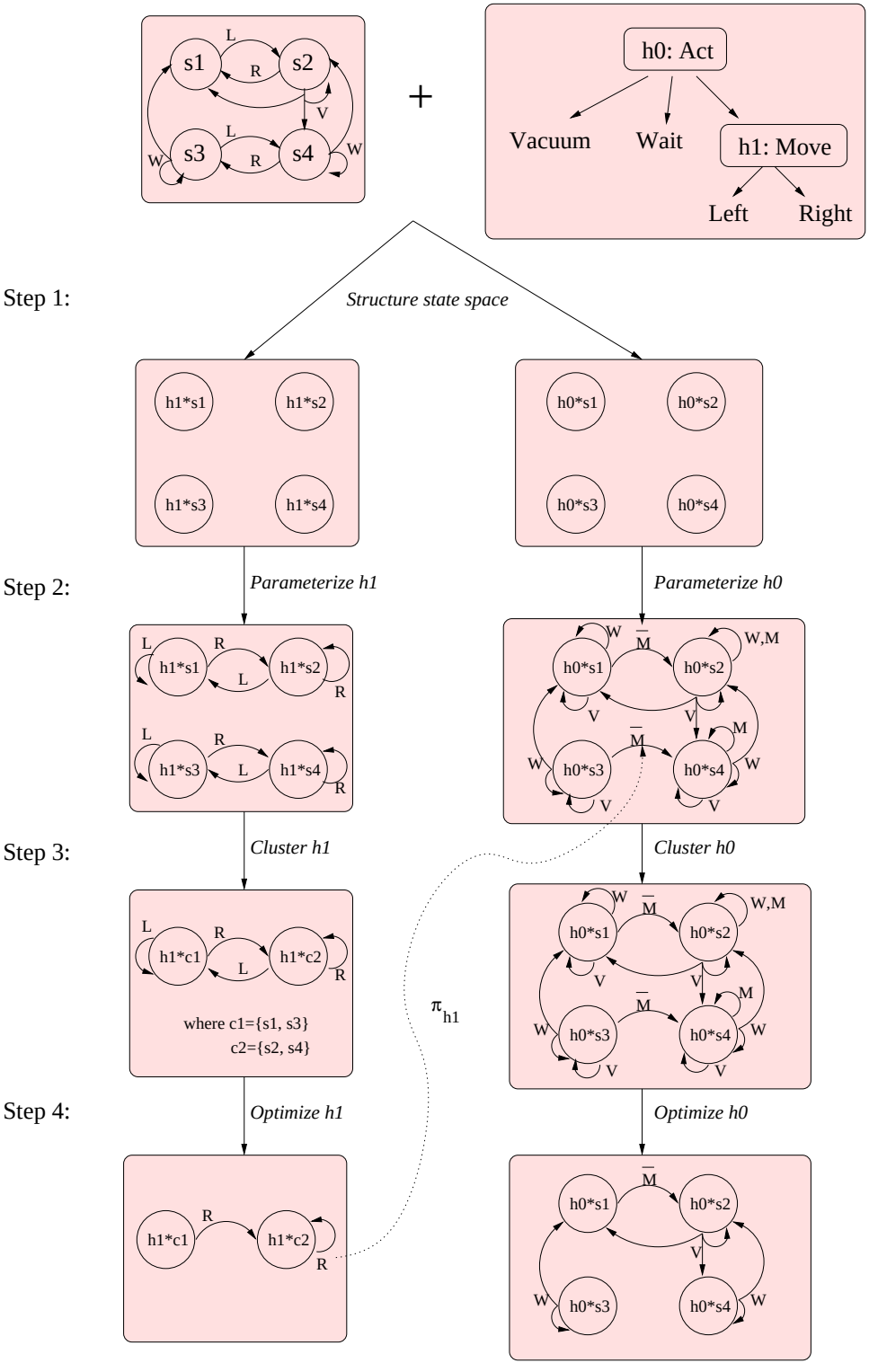


Figure 4.4. Hierarchical planning for the robot vacuuming example

4.2.3. 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. It does not pre-compute a global policy explicitly, but rather uses an online recursive procedure to generate the next policy action at each time step.

Execution corresponds to a trace down the subtask tree. The algorithm is described in Table 4.2. 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. It starts by consulting the local policy for the root task; this process yields a policy action, 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 would be impractical in POMDPs where we cannot guarantee detection of terminal states, and though this is not a concern for PolCA, it is important to remember that PolCA was designed only as a special case of the more general PolCA+ algorithm, where polling execution is crucial.

Polling execution can be more expensive than standard execution, because it requires consulting multiple local policies at each time step. However, this is often offset by the fact that each local policy is small relative to the (uncomputed) global policy. In general, policy execution speed is not a serious concern for most POMDP applications.

EXECUTE-PolCA(h, s_t)	0
Let $a_t = \pi_h(s_t)$	1
While a_t is an abstract action (i. e. $\bar{a}_t$)	2
Let h be the subtask spanned by $\bar{a}_t$	3
Let $a_t = \pi_h(s_t)$	4
End	5
Return a_t	6

Table 4.2. PolCA execution function

4.2.4. Theoretical Implications

One very important reason why this chapter discusses PolCA in its MDP formulation (separately from PolCA+) is because it holds theoretical properties that do not carry over to the fully general POMDP case. To better discuss the theoretical performance of PolCA, it is useful to introduce the following two definitions, which are adapted from Dietterich (2000):

DEFINITION 4.2.2. *Given Π_H, the class of all policies consistent with hierarchy H, a policy $\pi^* \in \Pi_H$ is said to be* HIERARCHICALLY OPTIMAL *if no other policy $\pi \in \Pi_H$ achieves more reward than π^*.*

DEFINITION 4.2.3. *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 among 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.*

The main difference between the two cases is a function of *context*. A recursively optimal solution guarantees optimality of a subtask's local policy, conditioned on that of its descendants. This is obtained when subtask policies are optimized without regard to the context in which each subtask is called. In contrast, hierarchical optimality guarantees optimality over the set of all policies consistent with the hierarchy. This is achieved by keeping track of all possible contexts for calling subtasks, which is key when subtasks have multiple goal states. In general, hierarchical optimal implies recursive optimality, though the reverse is not true. There is a trade-off between solution quality and representation: while in some domains hierarchical optimality offers a better solution, this comes at the expense of lesser state abstraction. Thus, recursive optimality is often more scalable.

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 4.1 computes π_H^*, a recursively optimal policy for M that is consistent with H.*

Proof: Let us first prove that Theorem 1 holds for the case where the planning algorithm is applied without state abstraction (i. e. Step 3 in Table 4.1). This is done using structural induction, which requires first showing 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 action. Applying Steps 2 and 4 from Table 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 hierarchically 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^+ whose value differs from that of π_h^* by ϵ, such that $\exists s \in S$ where:

$$V^{\pi_h^+}(s) \;=\; V^{\pi_h^*}(s) + \epsilon. \tag{4.11}$$

and consequently:

$$\max_{a \in A}\left[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^{\pi_h^+}(s')\right] \;=\; \max_{a \in A}\left[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^{\pi_h^*}(s')\right] + \epsilon. \tag{4.12}$$

Now, substituting Equation 4.11 into Equation 4.12 and simplifying:

$$\max_{a \in A}\left[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')(V^{\pi_h^*}(s') + \epsilon)\right] \;\geq\; \max_{a \in A}\left[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^{\pi_h^*}(s')\right] + \epsilon \tag{4.13}$$

$$\max_{a \in A}\left[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^{\pi_h^*}(s')\right] + \gamma\epsilon \;\geq\; \max_{a \in A}\left[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^{\pi_h^*}(s')\right] + \epsilon \tag{4.14}$$

$$\gamma\epsilon \;\geq\; \epsilon. \tag{4.15}$$

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 strictly on the proof of Dean and Givan (1997), which shows that the model minimization algorithm preserves policy quality. □

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 by restricting PolCA to this weaker form of optimality, it is often possible to achieve much greater state abstraction. This observation is not limited to PolCA. Work on the MAXQ formalism (Dietterich,

2000), which is also limited to recursive optimality, showed similar scalability. This issue is further explored in the experimental section below.

4.2.5. MDP Simulation Domain: Taxi Problem

We conclude this section by presenting a comparison of PolCA with competing hierarchical MDP algorithms: HSMQ (Dietterich, 2000), MAXQ (Dietterich, 2000) and ALisp (Andre & Russell, 2002). For this, we select the Taxi domain, a commonly used problem in the hierarchical MDP literature first proposed by Dietterich (2000). The overall task (Fig. 4.5) 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 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}. The initial state is selected randomly, however it is fully observable and transitions are deterministic. Motion 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.

Figure 4.6 represents the MAXQ control hierarchy used for this domain. The structured state space for this domain—called *Taxi1*—is formed by five 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*—which is identical to *Taxi1*, except that the passenger can now start from any location on the grid, compared to only {Y, B, R, G} in Taxi1.

Without any structural assumption, representing the solution for Taxi1 and Taxi2 requires respectively 3000 Q-values (500 states x 6 primitive actions) and 15600 Q-values (2600 states x 6 primitive actions). Figures 4.7 and 4.8 compare state abstraction results for each task, in terms of the number of parameters necessary to learn the solution. This gives a direct indication of the computation time for each algorithm.

In the results below, PolCA-Q indicates the standard PolCA algorithm, modified to cluster on Q-values (see end of Section 4.2.1.3), since all other approaches considered also have the ability to cluster on Q-values, rather than on $V(s)$.

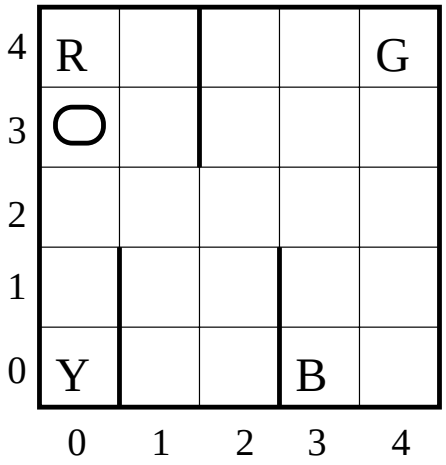


Figure 4.5. Taxi domain: Physical configuration

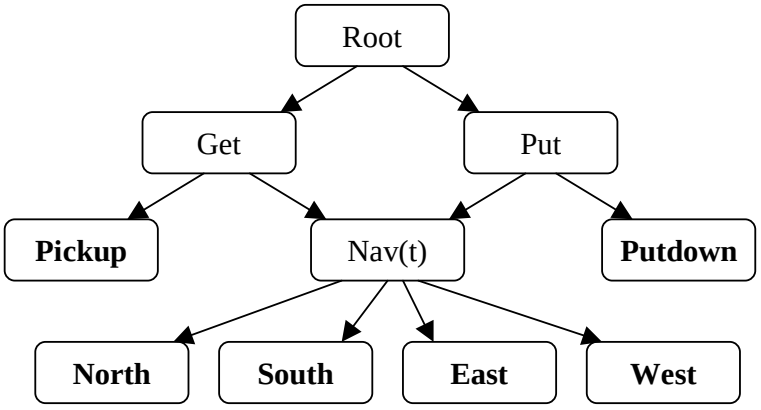


Figure 4.6. Taxi domain: Task hierarchy

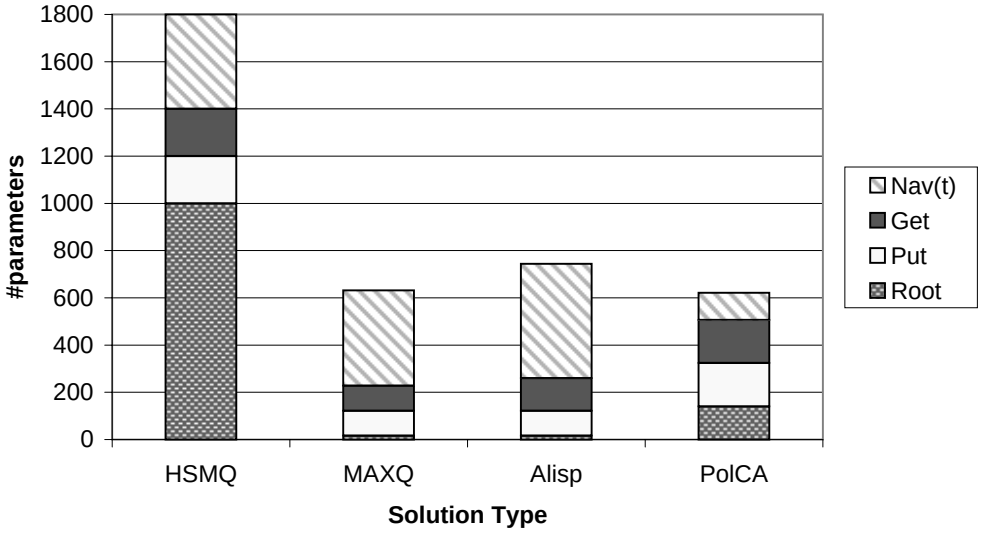


Figure 4.7. Number of parameters required to find a solution for Taxi1 task

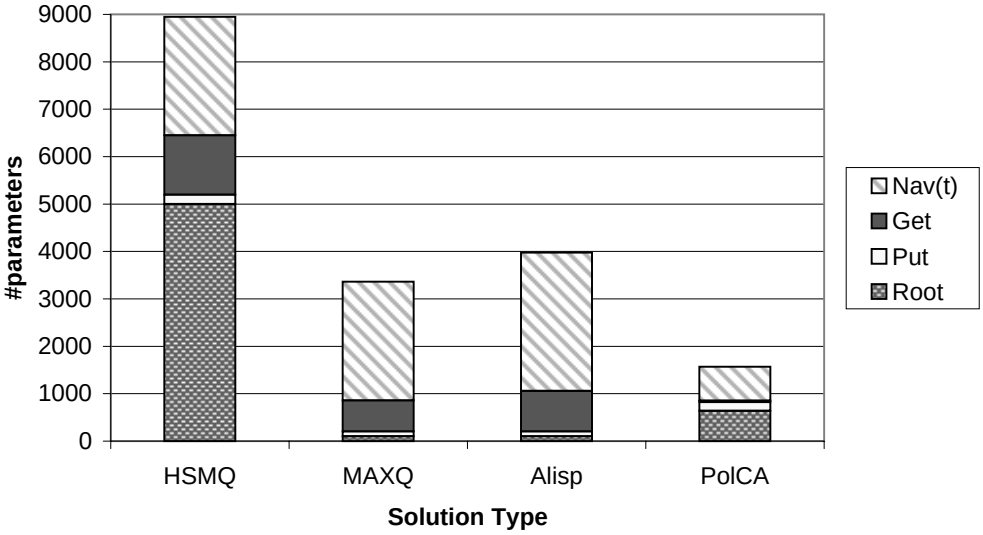


Figure 4.8. Number of parameters required to find a solution for Taxi2 task

In both versions of the problem, all four approaches (HSMQ, MAXQ, ALisp and PolCA) yield considerable savings, compared to the full set of parameters required for an optimal MDP solution. The abstraction results for Q-learning, HSMQ, MAXQ and ALisp in Taxi1 are published results (Dietterich, 2000; Andre & Russell, 2002); Taxi2 results are hand-crafted following a careful reading of each algorithm.

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 two because it represents external completion costs Q_e. Meanwhile PolCA gets further abstraction in low-level subtasks (e. g. $h_{Nav(\cdot)}$) because it automatically exploits spatial 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 $h_{Nav(\cdot)}$ 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.

All algorithms produce the same hierarchically-optimal policy on both of these tasks.

4.2.6. Conclusion

This concludes the discussion of the PolCA approach. The next section generalizes the concepts presented above to the case of partially observable domains, in the context of the PolCA+ approach. The experimental evaluation accorded to PolCA+ is much more extensive as it is the major focus of this chapter.

4.3. PolCA+: Planning for Hierarchical POMDPs

This section constitutes the cornerstone of this chapter. It presents the full PolCA+ algorithm, a POMDP generalization of the fully observable PolCA introduced in Section 4.2. Much of the algorithm remains unchanged from its MDP formulation, however some important modifications are necessary to accommodate partial observability.

Any attempt at proposing a hierarchical POMDP approach must overcome two obstacles. First, both planning and execution must be expressed with respect to *beliefs*, as opposed to *states*. Second, it is unreasonable to assume that terminal states (subgoals or otherwise) can be *detected*, which is a key assumption of many hierarchical MDP approaches. (This is not to say that terminal states cannot be *specified*, but that in the general POMDP 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).

The structural assumptions necessary for hierarchical POMDP planning are identical to the hierarchical MDP assumptions. Formally stated, given a task hierarchy H, each internal node represents a separate POMDP subtask h defined by:

- $A_h = \{\bar{a}_j, \ldots, a_p, \ldots\}$, the set of actions which are allowed in subtask h. Based on the hierarchy, there is one action for each immediate child of h.
- $\bar{R}_h(s,a)$, the local reward function. Each subtask in the hierarchy must have local (non-uniform) reward in order to optimize a local policy.

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

4.3.1. Planning Algorithm

The POMDP formulation of the planning algorithm remains largely unchanged from the MDP version (Table 4.1). 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 4.3 presents the hierarchical POMDP planning algorithm. The main function is called using the POMDP model $M = \{S, A, Z, b_0, T, O, R\}$ as the first argument and the hierarchical constraints $H = \{h_0, \ldots, h_m\}$ as the second argument.

PLAN-PolCA+(M,H)	0
STEP 1: Re-formulate structured state space: $H \circ S$	1
For each subtask $h \in H$, following a bottom-up ordering	2
STEP 2: Set parameters: $T(h \circ s, a, h \circ s'), O(h \circ s, a, z), R(h \circ s, a)$	3
STEP 3: Minimize states: $S \rightarrow S_h$	4
STEP 3b: Minimize observations: $Z \rightarrow Z_h$	5
STEP 4: Solve subtask: $M_h \rightarrow \pi_h^*$	6
End	7
End	8

Table 4.3. Main PolCA+ planning function

4.3.1.1. Step 1—*Re-formulate structured state space.* The first step is identical in both MDP and POMDP formulations (Section 4.2.1.1). Simply stated, a new state space $H \circ S$ is equal to the cross product between the original state space $S = \{s_0, \ldots, s_n\}$ and hierarchy state $H = \{h_0, \ldots, h_m\}$.

4.3.1.2. Step 2—*Set parameters.* This step translates the POMDP parameters specified in $M = \{S, A, Z, b_0, T, O, R\}$ to the structured state space $H \circ S$. The specification of the reward and transition parameters is identical to the MDP case (Section 4.2.1.2), and we now add the specification of the observation parameters.

Given a POMDP $M = \{S, A, Z, b_0, T, O, R\}$, to set parameters for subtask h we use Equations 4.16–4.22:

$$b_0(h \circ s_i) = b_0(s_i) \tag{4.16}$$

CASE 1 - PRIMITIVE ACTIONS: $\forall a_k \in A_h, a_k \in A, \forall (s_i, s_j) \in S, \forall z \in Z$:

$$T(h \circ s_i, a_k, h \circ s_j) = T(s_i, a_k, s_j) \tag{4.17}$$

$$O(h \circ s_i, a_k, z) = O(s_i, a_k, z) \tag{4.18}$$

$$R(h \circ s_i, a_k) = \bar{R}_h(s_i, a_k) \tag{4.19}$$

CASE 2 - ABSTRACT ACTIONS: $\forall \bar{a}_p \in A_h, a_p \notin A, \forall (s, s') \in S, \forall z \in Z$:

$$T(h \circ s_i, \bar{a}_p, h \circ s_j) = T(s_i, \pi^*_{h_p}(s_i), s_j), \tag{4.20}$$

$$O(h \circ s_i, \bar{a}_p, z) = O(s_i, \pi^*_{h_p}(s_i), z), \tag{4.21}$$

$$R(h \circ s_i, \bar{a}_p) = \bar{R}_h(s_i, \pi^*_{h_p}(s_i)) \tag{4.22}$$

As explained in Section 4.2.1.2, $\bar{a}_p$ 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. Unlike in the special-case MDP version, where parameter assignment preserved (recursive) optimality, the parameter assignment used here for abstract actions constitutes an approximation. The approximation arises in the treatment of abstract actions. An action $\bar{a}_p$ is modeled according to the policy at each *state* - when in the general case the policy can vary over the entire *belief*.

4.3.1.3. Step 3—*Minimize states.* The state clustering procedure for POMDPs extends the MDP model minimization by Dean and Givan (1997) to also consider observation probabilities when checking for stability between clusters. As in MDPs (Section 4.2.1.3), 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 $f_h(s) \rightarrow c$, the function mapping states $\{h \circ s_0, h \circ s_1, \dots\}$ to the (expanding) set of clusters $S_h = \{c_0, c_1, \dots\}$:

I - INITIALIZE STATE CLUSTERING: see Equation 4.5.

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

$$\sum_{s' \in c'} T(h \circ s_i, a, h \circ s') O(h \circ s', a, z) = \sum_{s' \in c'} T(h \circ s_j, a, h \circ s') O(h \circ s', a, z), \quad \forall (s_i, s_j) \in c, \forall c' \in S_h, \forall a \in A_h, \forall z \in Z. \tag{4.23}$$

III - IF A CLUSTER IS UNSTABLE, THEN SPLIT IT: see Equation 4.7

4.3.1.4. Step 3b—*Minimize observations*. This step automatically determines a clustering function $g_h^a(z)$ over observations. Observations can be clustered whenever they have similar emission probabilities, since it means that they provide equivalent information. As with state clustering, automatic observation abstraction is done on a subtask-per-subtask basis. However, in the case of observations, rather than learn a single clustering function per subtask, we learn one clustering function per action per subtask. This can mean greater model reduction in cases where multiple observations have similar emission probabilities with respect to some actions, but not all. Observation abstraction is especially useful to accelerate problem solving since the complexity of even one-step exact POMDP planning is exponential in the number of observations (Eqn 2.18).

To find the set of clusters $Z_h^a = \{k_0, k_1, \dots\}$, we start by assigning each observation to a separate cluster. We can then greedily merge any two clusters k and k' that provide equivalent information:

$$\exists \kappa \text{ s.t. } \sum_{s \in c} O(h \circ s, a, z_i) = \kappa \sum_{s \in c} O(h \circ s, a, z_j), \quad z_i \in k, \quad z_j \in k', \forall a \in A_h. \tag{4.24}$$

until there are no two clusters that meet this criteria.

It is important to point out that this approach does not only merge observations with *identical* emission probabilities, but also those with *proportionally equivalent* emission probabilities. This is appropriate because observations in POMDPs serve as indicators of the relative likelihood of each state.

4.3.1.5. Step 4—*Solve subtask*. This step focuses on optimizing 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 Z_h^a, have been found.

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

$$b_0(c) = \sum_{s \in c} b_0(s), \quad \forall c \in S_h \tag{4.25}$$

$$T(c, a, c') = \sum_{s' \in c} T(h \circ s, a, h \circ s'), \quad s \in c, \quad \forall (c, c') \in S_h, \quad \forall a \in A_h \tag{4.26}$$

$$O(c, a, k) = \sum_{s' \in c} \sum_{z \in k} O(h \circ s', a, z), \quad \forall k \in Z_h^a, \quad \forall c \in S_h, \quad \forall a \in A_h \tag{4.27}$$

$$R(c, a) = R(h \circ s, a), \quad s \in c, \quad \forall c \in S_h, \quad \forall a \in A_h. \tag{4.28}$$

Planning over clusters of states and observations can be realized by using any POMDP solver. For very small problems, it is possible to find an exact solution, using for example the Incremental Pruning algorithm (Cassandra et al., 1997). For larger domains, approximate algorithms are preferable. For example we have used the PBVI algorithm (Chapter 3), the Augmented-MDP algorithm (Roy & Thrun, 2000), and the QMDP fast approximation (Littman et al., 1995a).

On a side-note, when combining PolCA+ with the PBVI approximation, it is crucial to always generate belief points using the full action set A rather than the subtask-specific subset A_h. Failing to do so would cause a subtask to optimize its local policy only over beliefs that are reachable via its own action set, despite the fact that the subtask may be invoked in very different situations. The computational overhead of generating points is negligible and therefore this does not reduce the time gain of the hierarchy.

4.3.2. 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 4.3.1. *Given a subtask h, we say that $b^h(c)$, the belief defined over clusters $c \in S_h$, is a* LOCAL BELIEF.

Rather than update the local belief for each subtask separately, using the latest pair (a_{t-1}, z_t), we instead update the global belief $b_t()$ according to Equation 2.7. As the policy lookup traverses the tree, the local belief for each subtask, $b_t^h()$, is extracted from the global belief:

$$b_t^h(c) = \sum_{s \in c} b_t(s), \quad \forall c \in S_h, \tag{4.29}$$

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

Table 4.4 describes the complete hierarchical POMDP execution algorithm. 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. This completes our exposition of the general PolCA+ algorithm.

EXECUTE-PolCA+(h, b_t)	0
Let $b_t^h(c) = \sum_{s \in c} b_t(s), \quad \forall c \in S_h$	1
Let $a_t = \pi_h(b_t^h)$	2
While a_t is an abstract action (i. e. $\bar{a}_t$)	3
Let h be the subtask spanned by $\bar{a}_t$	4
Let $b_t^h(c) = \sum_{s \in c} b_t(s), \quad \forall c \in S_h$	5
Let $a_t = \pi_h(b_t^h)$	6
End	7
Return a_t	8
End	9

Table 4.4. PolCA+ execution function

4.3.3. Theoretical Implications

Unlike in MDPs, where the solution can be to shown to be recursively optimal, few theoretical claims can be made regarding the quality of the hierarchical POMDP solution found by PolCA+. In fact, we can easily demonstrate that the final solution will generally be sub-optimal, simply by considering Equations 4.20–4.22. This way of parameterizing abstract actions constitutes an approximation for the simple reason that the subtask policy π_h is only considered at the corners of its belief state (i. e. when the belief is restricted to a single state—$\pi_h(s)$). This ignores any other policy action that may be called in beliefs where there is uncertainty (i. e. $b(s) < 1, \forall s \in S$). The approximation is necessary to ensure that subtask h can be treated as a standard 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 $T(b,a,b')$, $O(b,a,z)$). 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 new model minimization procedures. First, there is a POMDP extension of the state minimization algorithm by Dean and Givan (1997). Second, there is a separate algorithm to perform observation minimization. It is important to demonstrate that those algorithmic procedures are sound with respect to POMDP solving, independent of any hierarchical context.

THEOREM 4.3.1. *Exact POMDP state abstraction. Let* $M = \{S, A, Z, b_0, T, O, R\}$ *be a POMDP. Then, the state minimization algorithm of Section 4.3.1.3 preserves sufficient information to learn* π^**, the optimal policy for* M*.*

Proof: We consider two states s_i and s_j, with matching cluster assignments:

$$c = f(s_i) = f(s_j)$$

obtained by the POMDP state clustering algorithm of Section 4.3.1.3. We use a proof by induction to show that any two beliefs $b = \{b_0, \ldots, b_i, b_j, \ldots\}$ and $b' = \{b_0, \ldots, b'_i, b'_j, \ldots\}$ that differ only in their probability over states s_i and s_j will have identical value $V(b) = V(b')$.

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

$$V_0(b) = \max_{a \in A} \left[b(s_i)R(s_i, a) + b(s_j)R(s_j, a) + \sum_{s \in S, s \neq \{s_i, s_j\}} b(s)R(s, a) \right] \quad (4.30)$$

$$V_0(b') = \max_{a \in A} \left[b'(s_i)R(s_i, a) + b'(s_j)R(s_j, a) + \sum_{s \in S, s \neq \{s_i, s_j\}} b(s)R(s, a) \right]. \quad (4.31)$$

Assuming that $f(s_i) = f(s_j)$, then by Equation 4.5 we can substitute $R(s_j) \leftarrow R(s_i)$ in Equation 4.31:

$$V_0(b') = \max_{a \in A} \left[(b'(s_i) + b'(s_j))\, R(s_i, a) + \sum_{s \in S, s \neq \{s_i, s_j\}} b(s)R(s, a) \right]. \quad (4.32)$$

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 4.32:

$$V_0(b') = \max_{a \in A} \left[(b(s_i) + b(s_j))\, R(s_i, a) + \sum_{s \in S, s \neq \{s_i, s_j\}} b(s)R(s, a) \right], \quad (4.33)$$

leading to the conclusion that:

$$V_0(b) = V_0(b'). \quad (4.34)$$

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

$$V_{t-1}(b) = V_{t-i}(b'). \quad (4.35)$$

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

$$V_t(b_t) = \max_{a \in A} \left[\sum_{s \in S} b_t(s)R(s, a) + \gamma \sum_{z \in Z} Pr(z \mid a, b_t)V_{t-1}(b_{t-1}) \right] \quad (4.36)$$

$$V_t(b'_t) = \max_{a \in A} \left[\sum_{s \in S} b'_t(s)R(s, a) + \gamma \sum_{z \in Z} Pr(z \mid a, b'_t)V_{t-1}(b'_{t-1}) \right]. \quad (4.37)$$

By using Equation 4.34 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 4.37:

$$V_t(b'_t) = \max_{a\in A}\left[\sum_{s\in S} b_t(s)R(s,a) + \gamma\sum_{z\in Z} Pr(z \mid a, b'_t)V_{t-1}(b'_{t-1})\right]. \quad (4.38)$$

Next, we use the POMDP stability criterion (Eqn 4.23) in conjunction with Equation 4.35 and the belief update equation (Eqn 2.7) to infer that $b'_{t-1} = b_{t-1}$, conditioned on each observation $z \in Z$, and therefore:

$$V_t(b'_t) = \max_{a\in A}\left[\sum_{s\in S} b_t(s)R(s,a) + \gamma\sum_{z\in Z} Pr(z \mid a, b_t)V_{t-1}(b_{t-1})\right], \quad (4.39)$$

leading to the conclusion that $V_t(b) = V_t(b')$. □

THEOREM 4.3.2. *Exact POMDP observation abstraction. Consider two POMDPs* $M = \{S, A, Z, b_0, T, O, R\}$ *and* $M' = \{S, A, Z', b_0, T, O', R\}$ *with respective optimal solutions* V *and* V'*, where* $Z = \{z_0, z_1, \ldots, z_n, z_{n+1}, z_{n+2}\}$*,* $Z' = \{z_0, z_1, \ldots, z_n, k\}$*, and* $\exists a \in A$ *such that*

$$O'(s,a,k) = O(s,a,z_{n+1}) + O(s,a,z_{n+2}), \forall s \in S. \quad (4.40)$$

If $\exists \kappa \in \mathbb{R}$ *such that:*

$$O(s,a,z_{n+1}) = \kappa O(s,a,z_{n+2}), \forall s \in S, \quad (4.41)$$

meaning that z_{n+1} *and* z_{n+2} *having matching cluster assignment* k*.*
Then

$$V(b) = V'(b), \forall b \in B. \quad (4.42)$$

Proof: Using a proof by induction, first consider:

$$V_0(b) = \max_{a\in A}\left[\sum_{s\in S} b_0(s)R(s,a)\right]$$

$$V'_0(b) = \max_{a\in A}\left[\sum_{s\in S} b_0(s)R(s,a)\right]$$

and it therefore follows that: $V_0(b) = V'_0(b)$. (4.43)

We now assume that:

$$V_{t-1}(b) = V'_{t-1}(b). \quad (4.44)$$

Before proceeding with the proof for $V_t(b) = V'_t(b)$, we first establish that:

$$\tau(b,a,k) = \tau(b,a,z_{n+1}). \quad (4.45)$$

We consider

$$
\begin{array}{lcll}
\tau(b,a,k)(s) & = & \text{c} \sum_{s'} O(s',a,k)T(s,a,s')b(s') & \text{Eqn 2.7} \\
& = & \text{c} \sum_{s'} \left(O(s',a,z_{n+1}) + O(s',a,z_{n+2})\right) T(s,a,s')b(s') & \text{Eqn 4.40} \\
& = & \text{c} \sum_{s'} (1+\kappa) O(s',a,z_{n+1})T(s,a,s')b(s') & \text{Eqn 4.41} \\
& = & \text{c'} \sum_{s'} O(s',a,z_{n+1})T(s,a,s')b(s') & \text{normalizing constant} \\
& = & \tau(b,a,z_{n+1})(s), \forall s \in S & \text{Eqn 2.7.}
\end{array}
$$

Similar steps can be used to show that:

$$\tau(b,a,z_{n+1}) = \tau(b,a,z_{n+2}). \tag{4.46}$$

Now, we proceed to show that:

$$V_t(b) = V'_t(b). \tag{4.47}$$

We begin with

$$
\begin{array}{lcll}
\text{V}_t(b) & = & \max_{a \in A} \left[\sum_{s \in S} b(s)R(s,a) + \gamma \sum_{z \in Z} Pr(z \mid a,b) V_{t-1}(\tau(b,a,z))\right] & \text{Eqn 2.11} \\
& = & \max_{a \in A} \Big[\sum_{s \in S} b(s)R(s,a) + \gamma \sum_{z=\{z_0,\ldots z_n\}} Pr(z \mid a,b) V_{t-1}(\tau(b,a,z)) & \\
& & \quad + \gamma \;\; V_{t-1}(\tau(b,a,z_{n+1})) \sum_{s \in S} O(s,a,z_{n+1})b(s) & \\
& & \quad + \gamma \;\; V_{t-1}(\tau(b,a,z_{n+2})) \sum_{s \in S} O(s,a,z_{n+2})b(s)\Big] & \text{expanding} \\
& = & \max_{a \in A} \Big[\sum_{s \in S} b(s)R(s,a) + \gamma \sum_{z=\{z_0,\ldots z_n\}} Pr(z \mid a,b) V_{t-1}(\tau(b,a,z)) & \\
& & \quad + \gamma \;\; V_{t-1}(\tau(b,a,z_{n+1})) \sum_{s \in S} O(s,a,z_{n+1})b(s) & \\
& & \quad + \gamma \;\; V_{t-1}(\tau(b,a,z_{n+2})) \sum_{s \in S} \left(O(s,a,k) - O(s,a,z_{n+1})\right) b(s)\Big] & \text{Eqn 4.40} \\
& = & \max_{a \in A} \Big[\sum_{s \in S} b(s)R(s,a) + \gamma \sum_{z=\{z_0,\ldots z_n\}} Pr(z \mid a,b) V_{t-1}(\tau(b,a,z)) & \\
& & \quad + \gamma \;\; V_{t-1}(\tau(b,a,z_{n+1})) \sum_{s \in S} O(s,a,z_{n+1})b(s) & \\
& & \quad + \gamma \;\; V_{t-1}(\tau(b,a,z_{n+1})) \sum_{s \in S} \left(O(s,a,k) - O(s,a,z_{n+1})\right) b(s)\Big] & \text{Eqn 4.46} \\
& = & \max_{a \in A} \Big[\sum_{s \in S} b(s)R(s,a) + \gamma \sum_{z=\{z_0,\ldots z_n\}} Pr(z \mid a,b) V_{t-1}(\tau(b,a,z)) & \\
& & \quad + \gamma \;\; V_{t-1}(\tau(b,a,z_{n+1})) \sum_{s \in S} O(s,a,k)b(s)\Big] & \text{simplifying} \\
& = & \max_{a \in A} \Big[\sum_{s \in S} b(s)R(s,a) + \gamma \sum_{z=\{z_0,\ldots z_n\}} Pr(z \mid a,b) V_{t-1}(\tau(b,a,z)) & \\
& & \quad + \gamma \;\; V_{t-1}(\tau(b,a,k)) \sum_{s \in S} O(s,a,k)b(s)\Big] & \text{Eqn 4.45} \\
& = & \max_{a \in A} \left[\sum_{s \in S} b(s)R(s,a) + \gamma \sum_{z \in Z'} Pr(z \mid a,b) V_{t-1}(\tau(b,a,z))\right] & \\
& = & \text{V'}_t(b) & \text{Eqn 2.11}
\end{array}
$$

We conclude that no loss of performance results from clustering observations that satisfy Equation 4.24. □

The remainder of this chapter explores the empirical performance of PolCA+. We consider a number of contrasting POMDP problems, and compare the PolCA+ algorithm with other well-established POMDP solving algorithms, both exact and approximate.

4.3.4. Simulation Domain 1: Part-Painting Problem

The first task considered is based on the part-painting problem described by Kushmerick, Hanks, and Weld (1995). 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 objective of this domain is to process 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, in fact the state set includes only four states: $\{$*unflawed-unblemished-unpainted, unflawed-unblemished-painted, flawed-unblemished-painted, flawed-blemished-unpainted*$\}$. In addition, the domain has four actions: $A=\{$*inspect, paint, ship, reject*$\}$ and two observations: $Z=\{$*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:

$$Pr(s_{painted} = 1 \mid a = paint, s_{painted} = 0) = 0.9, \tag{4.48}$$

$$Pr(s_{painted} = 0 \mid a = paint, s_{painted} = 0) = 0.1. \tag{4.49}$$

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

$$Pr(s_{blemished} = 0 \mid a = paint, s_{blemished} = 1) = 0.9, \tag{4.50}$$

$$Pr(s_{blemished} = 1 \mid a = paint, s_{blemished} = 1) = 0.1. \tag{4.51}$$

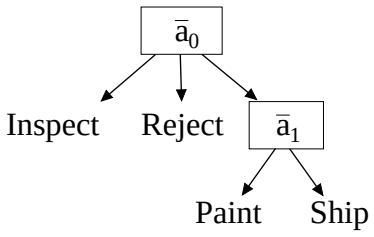


Figure 4.9. Action hierarchy for part-painting task

PolCA+ operates by leveraging structural constraints. Figure 4.9 shows the action hierarchy considered for this task. Though there are many possible hierarchies, this seemed like a reasonable choice given minimum knowledge of the problem.

As explained in Section 4.3.1.5, PolCA+ uses a value function estimator as a subcomponent. For this experiment, four different choices are considered: Incremental Pruning (Section 2.2.1), PBVI (Chapter 3), QMDP (Section 2.2.4, Eqn 2.32) and MLS (Section 2.2.4, Eqn 2.29). We test PolCA+ in combination with each of the four different planners.

Table 4.5 contains the results of these experiments. The main performance metrics considered are the computation time and the value accumulated over multiple simulation trials. The reward column presents the (discounted) sum of rewards for a 500-step simulation run, averaged over 1000 runs. This quantifies the online performance of each policy. Clearly, the choice of solver can have a large impact on both solution time and performance. An exact solver such as Incremental Pruning generally affords the best solution, albeit at the expense of significant computation time. In this case, PolCA+ combined with any of Incremental Pruning, PBVI, or QMDP finds a near-optimal solution. The good performance of the QMDP can be attributed to the fact that this domain contains a single information-gathering action.

In addition, for a problem of this size, we can look directly at the policy yielded by each planning method. As indicated in the *Policy* column, the different algorithms each learn one of three policies. Figure 4.10 illustrates the corresponding policies (nodes show actions; arrows indicate observations when appropriate; dotted lines indicate a task reset, which occurs after a part has been rejected or shipped).

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 Fig. 4.9) is seen in policy π^+.

Finally, as reported in Table 4.5, using a hierarchical decomposition in conjunction with Incremental Pruning can actually cause the computation time to *increase*, compared to straightforward Incremental Pruning. This occurs because the problem is so small and because it offers no state or observation abstraction; results on larger problems presented below clearly show the time savings attributed to hierarchical assumptions.

Problem Solution $\lvert S\rvert$=4, $\lvert A\rvert$=4, $\lvert Z\rvert$=2	CPU time (secs)	Reward	Policy
Incremental Pruning	2.6	3.3	π^*
PolCA+ w/Incremental Pruning	21.6	3.2	π^+
PolCA+ w/PBVI	2.5	3.2	π^+
PolCA+ w/QMDP	< 0.01	3.2	π^+
PolCA+ w/MLS	< 0.01	-0.97	π^-

Table 4.5. Performance results for part-painting task

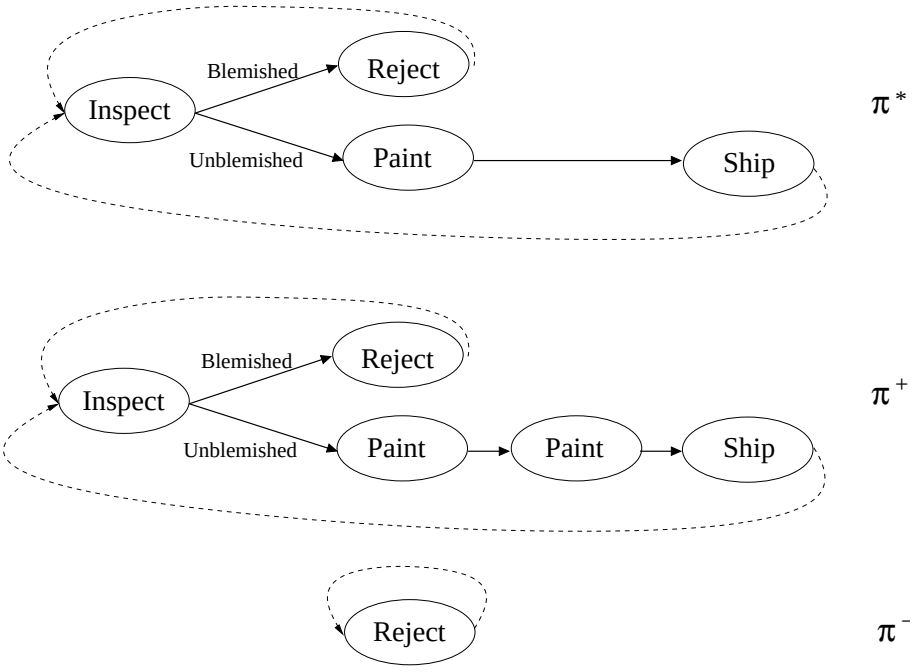


Figure 4.10. Policies for part-painting task

4.3.5. Simulation Domain 2: Cheese-Taxi Problem

This section addresses a robot navigation task that is a cross between the taxi problem presented in Section 4.2.5 and another problem called the cheese maze (McCallum, 1996). The problems are combined to join the state uncertainty aspects proper to the cheese maze and the hierarchical structure proper to the taxi task.

The problem features a taxi agent operating in a world that has the configuration of the cheese maze (Fig. 4.11), where the agent must pickup a passenger located at state *s10* and then proceed to deliver the passenger to a (randomly selected) 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 $\{s0, s1, \ldots, s10\}$ 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: $\{o1, o2, o3, o4, o5, o6, o7, destinationS0, destinationS4, null\}$.

S0 O1	S1 O2	S2 O3	S3 O2	S4 O4
S5 O5		S6 O5		S7 O5
S8 O6		S10 O7		S9 O6

Figure 4.11. 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 by 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 unique identifiers. 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 encodes the effect of both deterministic motion actions, and a stochastic destination choice. For example, motion actions have the expected transition effects:

$$Pr(s' = s2 \mid a = North, s = s6) = 1,$$

and so on. The choice of passenger destination (*s0* or *s4*) is randomly selected when the passenger is picked-up. And whenever the taxi has the passenger onboard and is in cells *s2* or *s6*, there is a small probability that the passenger will change his / her mind and suddenly select the other destination:

$$Pr(destination = \textit{s0} \mid a = \textit{North}, location = \textit{s6}, destination = \textit{s0}) = 0.95,$$

$$Pr(destination = \textit{s0} \mid a = \textit{North}, location = \textit{s6}, destination = \textit{s4}) = 0.05,$$

and so on. This possibility is added simply to increase the difficulty of the task.

The agent incurs a -1 reward for any motion or query action. A final reward of $+20$ is received for delivering the passenger at the correct location. A -10 reward is incurred for applying the *Pickup* or *Putdown* action incorrectly.

There are two 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. Second, the passenger's destination can only be observed by using the *Query* action.

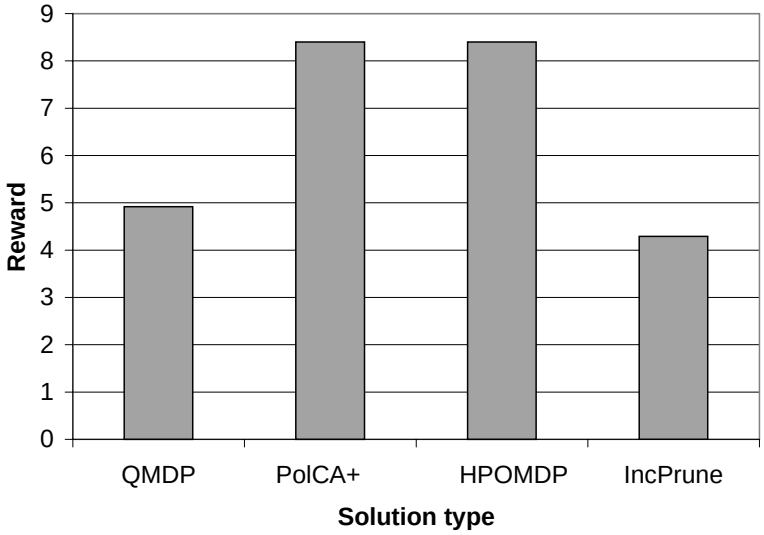
The transition and reward parameters used here are consistent with the original taxi task; the observations parameters (with the exception of the *Query* action) are borrowed directly from the original cheese maze. Finally, we also adopt the taxi task's usual hierarchical action decomposition, as shown in Figure 4.6.

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. $h_{Nav(\cdot)}$ has a uniform reward function $R_{Nav(\cdot)}(s,a) = -1, \forall(s,a)$). Thus, we reward achievement of partial goals in the $h_{Nav(\cdot)}$ subtask by using the pseudo-reward function:

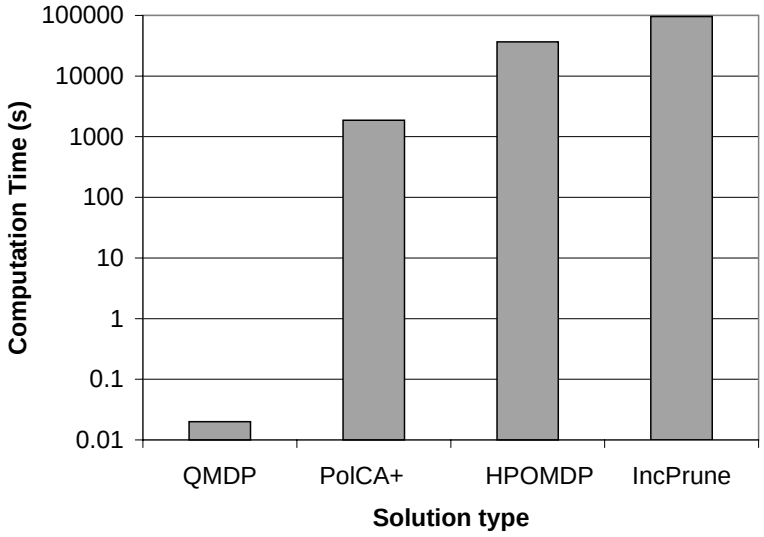
$$\bar{R}_{Nav(S0)}(s = S0, a) = 0, \ \ \forall a \in A_{Nav(S0)}$$

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

Figure 4.12 presents results for the cheese-taxi domain, for each of the POMDP solving algorithms. Figure 4.12a illustrates the sum of rewards to accomplish the full task, averaged over 1000 trials, whereas Figure 4.12b illustrates the computation time necessary to reach the solution. These figures include results for two different hierarchical POMDP solutions (PolCA+ and HPOMDP). PolCA+ is the full algorithm as described in Section 4.3.1, with exact solving of subtasks. HPOMDP uses the same hierarchical algorithm, but without any state or observation abstraction.



(a) Reward profile



(b) Computation time

Figure 4.12. Results for solving the cheese-taxi task

The QMDP policy and the truncated exact POMDP policy perform equally poorly. In the case of QMDP, this is due to its inability to disambiguate the final destination. The QMDP policy correctly guides the agent to pickup the passenger, but it never drops off the passenger at either location. Meanwhile the exact POMDP algorithm is theoretically able to find the shortest action sequence, but it would require much longer planning time to do so. It was terminated after over 24 hours of computation, having completed only 5 iterations of exact value iteration.

PolCA+ and HPOMDP produce the same policy. Following this policy, the agent correctly applies an initial sequence of motion actions, simultaneously disambiguating the taxi's original position and making progress to the passenger's station at $S10$. Once the passenger location is reached, the agent applies the *Pickup* action and navigates to $S2$ before applying the *Query* action. It then proceeds to the correct passenger destination.

The computation time comparison is shown in Figure 4.12b. It should be pointed out that the exact POMDP solution was truncated after many hours of computation, before it had converged to a solution. The *Root* and *Put* subtasks in both PolCA+ and HPOMDP 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 4.12a.

As expected, results for both HPOMDP and PolCA+ are identical in terms of performance (since PolCA+ used lossless state and observation abstraction), but require a longer solution time in the case of HPOMDP. Both PolCA+ and HPOMDP use the action decomposition of Figure 4.6.

The computation time data in Figure 4.12b allows us to distinguish between the time savings obtained from the hierarchical decomposition (the difference between POMDP and HPOMDP) versus the time savings obtained from the automated state/observation abstraction (the difference between HPOMDP and PolCA+). In this domain, the hierarchy seems to be the dominant factor. In terms of abstraction, it is worth noting that in this domain, the savings come almost entirely from state abstraction. The only observation abstraction available is to exclude zero-probability observations, which has only negligible effect on computation time. The state abstraction savings on the other hand are appreciable, due to symmetry in the domain and in the task objective.

We conclude that the PolCA+ algorithm is able to solve this problem, where partial observability features prominently. The action decomposition and state abstraction combine to provide a good solution in reasonable time.

4.3.6. Simulation Domain 3: A Game of Twenty-Questions

One of the main motivating applications for improved POMDP planning is that of robust dialogue modeling (Roy, Pineau, & Thrun, 2000). When modeling a robot interaction manager as a POMDP, as we do in the next chapter, the inclusion of information-gathering actions is crucial to a good policy, since human-robot interactions are typically marred by ambiguities, errors and noise. In this section, we consider a new POMDP domain that is based on an interactive game called *Twenty-questions* (Burgener, 2002), also known as *"Animal, Vegetable, or Mineral?"* This simple game provides us with a stylized (and naturally scalable) version of an interaction task. Studying this game allows for systematic comparative analysis of POMDP-based dialogue modeling, 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 play a useful role for the prototyping of dialogue management systems, much like the role that the often-used maze navigation task has played for robot navigation domains.

The game Twenty-questions is typically formulated as a 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 compute 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 it a turtle?*). The list of *questions* serves to disambiguate between state-objects (e. g. *Is it green? Is it a fruit? Is it a mineral?*), though noisy answers can complicate the matter. The observation space contains only three items: $\{yes, no, noise\}$, corresponding to possible verbal responses from the non-POMDP player who 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, the conventional rules of the game prescribe that state transitions be restricted to self-transitions, since the game usually assumes a stationary object. Given this stationarity assumption, it is likely that a decision-tree (Quinlan, 1986) could successfully solve the problem. To make it more interesting as a POMDP domain, we add a small probability of randomly transitioning from the current state-object to another one, in effect allowing the first player to change his/her mind about the target object in the middle of play. Though not traditionally part of this game, adding stochasticity in the state transitions makes this a much more challenging problem (in the same way that chasing a moving target is harder than seeking a fixed one). We assume that after each question, the state stays the same with probability 0.9, and uniformly randomly changes to any of the other states with cumulative probability 0.1.

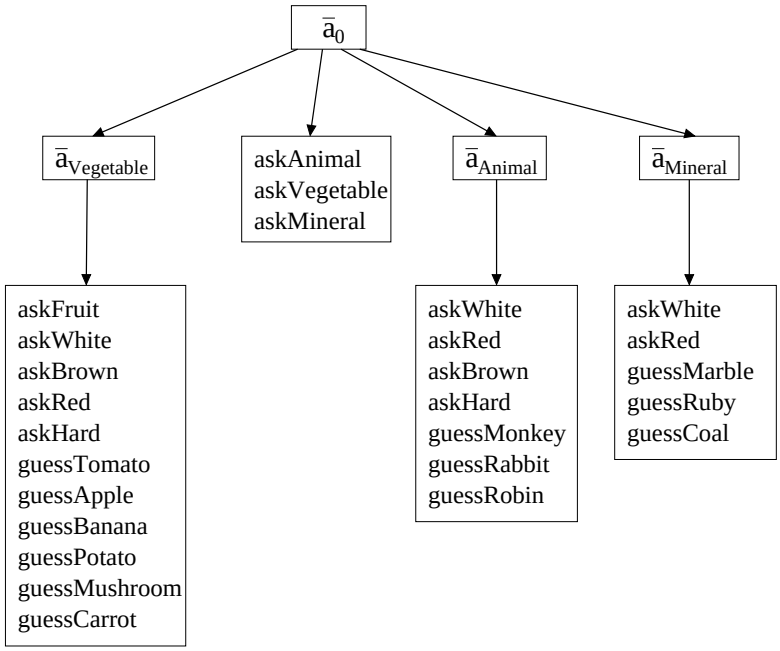
The reward is consistently -1 for all *question*-actions, whereas *guess*-actions yield a $+5$ reward if the guess is correct and a -20 reward otherwise. The task is reset every time the policy selects a *guess*-action. Finally, the observation probabilities for each *question*-action noisily reflect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

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

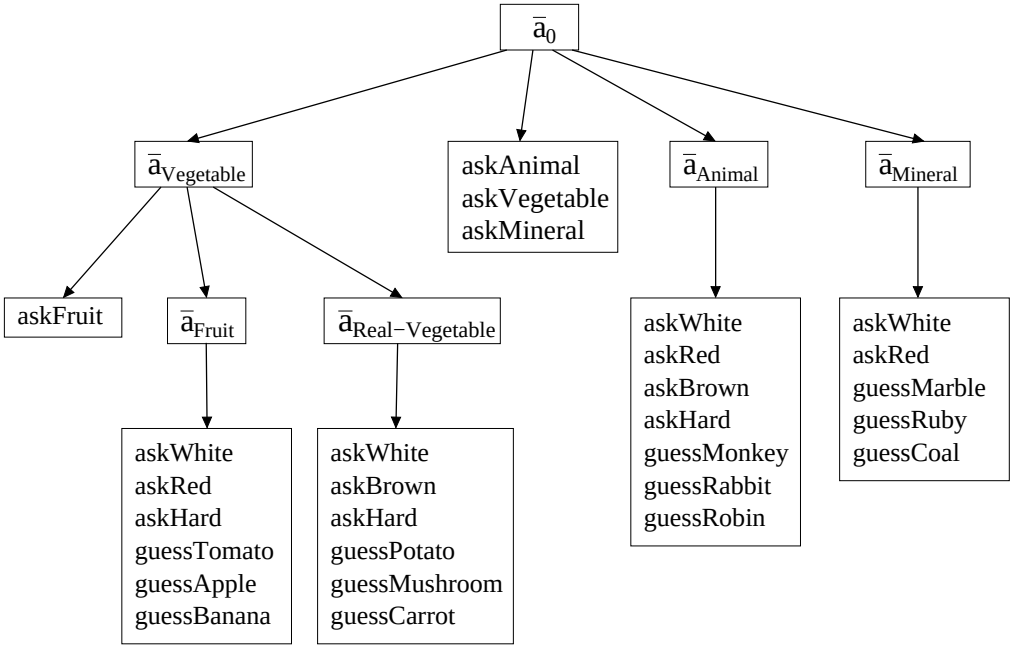
We implemented a 12-object version of this domain. The POMDP representation contains 12 states (one per object), 20 actions (12 guesses + 8 questions), and 3 observations (*yes, no, noise*). We considered two alternate hierarchical decompositions for this domain. Figure 4.13a illustrates the first decomposition (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 $h_{\textit{vegetable}}$.[1] We therefore proposed the second decomposition (referred to as D2), which is illustrated in Figure 4.13b. This decomposition further partitions the action space of the $h_{\textit{vegetable}}$ subtask, to produce two new lower-level subtasks: $h_{\textit{real-vegetable}}$ and $h_{\textit{fruit}}$.

The PolCA+ planning algorithm was applied twice, once for each decomposition. Policies were also generated using alternate algorithms, including QMDP (Section 2.2.4), FIB (Section 2.2.4), and Incremental Pruning (Section 2.2.1). 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.

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

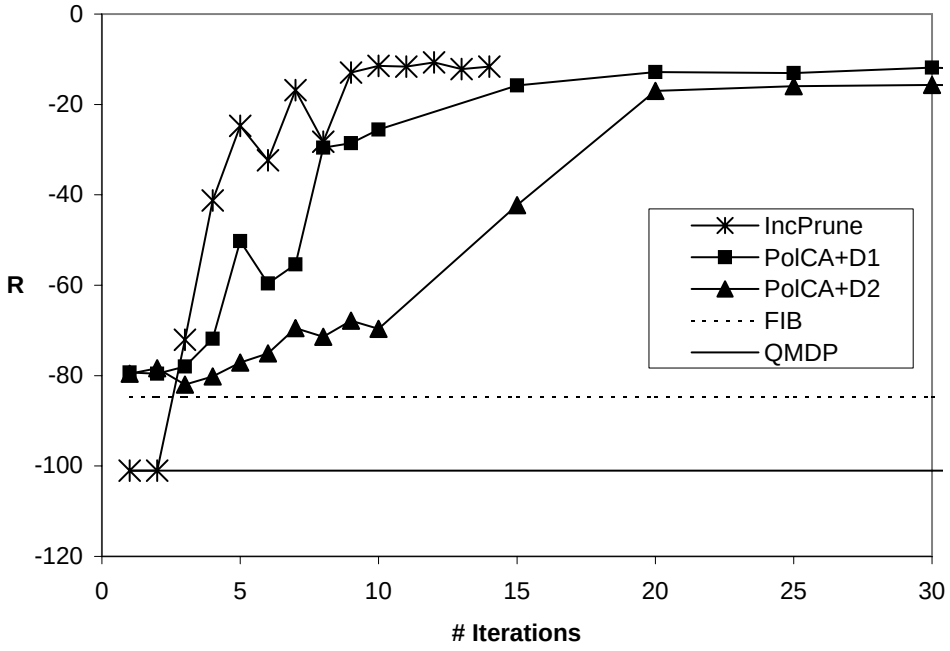


(a) D1 hierarchy

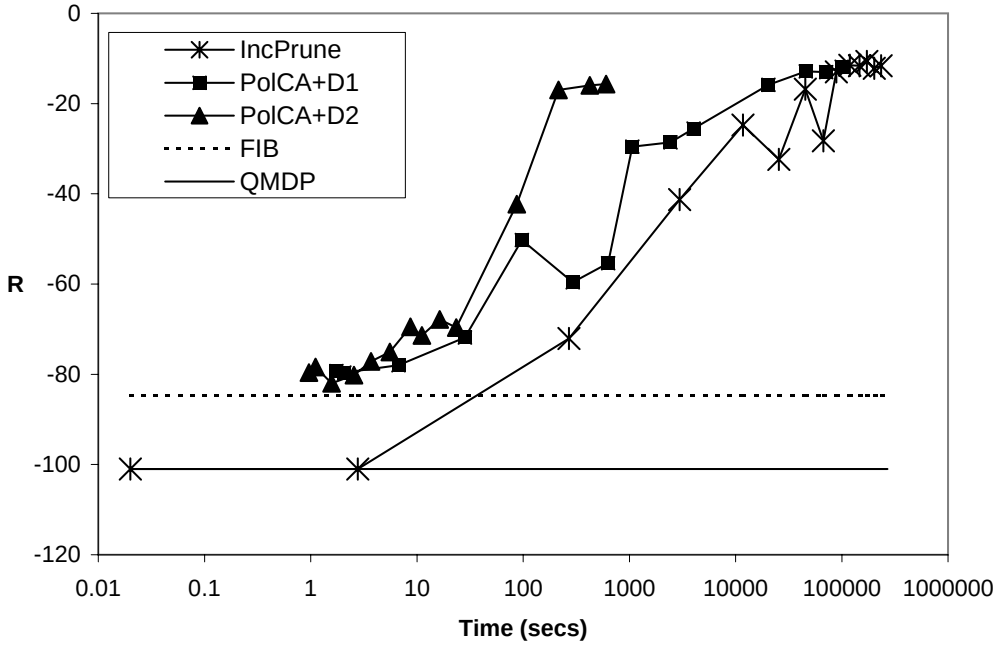


(b) D2 hierarchy

Figure 4.13. Action hierarchies for twenty-questions domain



(a) Performance as a function of value iterations



(b) Performance as a function of computation time

Figure 4.14. Simulation results for the twenty-questions domain

Figure 4.14a 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 (PolCA+D1, PolCA+D2), the full number of iterations was completed for each subtask. The QMDP and FIB results are plotted as constants, representing optimized performance. These results clearly illustrate the failures of the QMDP and FIB algorithms when faced with a POMDP domain where explicit information-gathering is required. Looking closely at the policies generated by QMDP and FIB, we note 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 is never able to take a correct guess, and in each trial spends 100 time steps asking random questions without any useful effect. As expected, the performance of Incremental Pruning (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 4.14b presents the same results as in Figure 4.14a, but now plotting the reward performance as a function of computation time. All POMDP computations, including for hierarchical subtasks, assume a pruning criterion of $\epsilon = 0.5$. 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 structural assumptions.

We conclude that PolCA+'s hierarchical decomposition preserves sufficient richness in representation to successfully address dialogue-type POMDPs. Furthermore, through careful 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), preference elicitation (Boutilier, 2002), and decision trees (Quinlan, 1986). However, the stochasticity in state transitions make decision trees a poor choice for this specific formulation of the twenty-questions domain.

4.4. Related Work

Various techniques have been developed that exploit intrinsic properties of a domain to accelerate problem-solving. Hierarchical decomposition techniques in particular accelerate planning for complex problems by leveraging domain knowledge to set intermediate goals. They typically define separate subtasks and constrain the solution search space.

This insight has been exploited in classical planning, starting with abstraction for STRIPS-like planners (Sacerdoti, 1974), and followed by the well-studied hierarchical task networks (HTNs) (Tate, 1975). In HTNs, the planning problem is decomposed into a network of *tasks*. High-level abstract tasks are represented through preconditions and effects, as well as methods for decomposing the task into lower-level subtasks. Low-level tasks contain simple primitive actions. In general, HTN planning has been shown to be undecidable. More recent algorithms combine HTN structural assumptions with partial-order planners, in which case problems are decidable (Barrett & Weld, 1994; Ambros-Ingerson & Steel, 1988). HTN planning has been used in large-scale applications (Bell & Tate, 1985). However it is best suited for deterministic, fully observable domains.

The two dominant paradigms for large-scale MDP problem solving are based on *function approximation* and *structural decomposition*. PolCA belongs to the second group. The literature on structural decomposition in MDPs is extensive and offers a range of alternative algorithms for improved planning through structural decomposition (Singh, 1992; Dayan & Hinton, 1993; Kaelbling, 1993; Dean & Lin, 1995; Boutilier, Brafman, & Geib, 1997; Meuleau, Hauskrecht, Kim, Peshkin, Kaelbling, Dean, & Boutilier, 1998; Singh & Cohn, 1998; Wang & Mahadevan, 1999). A common strategy is to define subtasks via partitioning the state space. This 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 (Eqn 2.18), PolCA+ relies on a structural decomposition of the task / action space.

Approaches most related to PolCA include MAXQ (Dietterich, 2000), HAM (Parr & Russell, 1998), ALisp (Andre & Russell, 2002), and options (Sutton et al., 1999). These all favor an action-based decomposition over a state-based partition. As in PolCA, these approaches assume that the domain knowledge necessary to define the subtask hierarchy is provided by the designer. Subtasks are formally defined using a combination of elements, including: initial states, expected goal states, fixed / partial policies, reduced action sets, and local reward functions.

In the options framework, subtasks consist of fixed-policy multi-action sequences. These temporally abstract subtasks, when incorporated within the reinforcement-learning framework, can accelerate learning while maintaining the guarantee of hierarchical optimality. The options framework has been extended to include automatic state abstraction (Jonsson & Barto, 2001) and thus improve its scalability. An important impediment in applying it to real-world domains is its inability to handle partial observability.

Parr and Russell's Hierarchy of Abstract Machines (HAM) defines each subtask using a non-deterministic finite-state controller. HAM can be optimized using either (model-based) dynamic programming or (model-free) reinforcement-learning to produce a hierarchically optimal solution. HAM does not explicitly leverage possibilities for state abstraction, which is a concern for scalability. The other limitation is the fact that HAM cannot easily handle partial observability.

Dietterich's MAXQ method probably shares the most similarities with PolCA. It assumes an action hierarchy like PolCA's, and defines each subtask using a combination of termination predicate (e. g. end state—which PolCA does not require) and local reward function (which PolCA requires). Both MAXQ and PolCA take advantage of state abstraction. MAXQ assumes a hand-crafted abstraction whereas PolCA automatically finds the abstraction. We believe the automatic decomposition is preferable because 1) it prevents user-introduced errors and 2) applied in a policy-contingent way (i. e. only once lower-level subtasks have been solved) it yields more abstraction. The implication, however, is that MAXQ can operate in a model-free RL setting. PolCA on the other hand requires a full model to learn the abstraction and to optimize its policy. Both approaches achieve a recursively optimal policy. The main advantage of PolCA (in addition to the automated policy-contingent state abstraction) is its natural extension to partially observable domains.

Finally, in Andre and Russell's ALisp, structural constraints take the form of partially specified agent programs. The partial specification is formulated as choice points where reduced action sets (with both primitive and abstract actions) are considered. It is most promising in that it subsumes MAXQ, HAM and options. However, it has not been extended to the partial observability case.

Most of the structural approaches discussed here were formulated specifically for MDPs. Nonetheless they share many similarities with PolCA+, in particular with regards to structural assumptions. Recent years have seen the development of a few hierarchical POMDP approaches (Hernandez-Gardiol & Mahadevan, 2001; Theocharous et al., 2001; Wiering & Schmidhuber, 1997; Castañon, 1997). However these are quite different from PolCA+ in terms of structural assumptions. They are discussed in Section 2.2.7.

4.5. Contributions

This chapter describes a hierarchical decomposition approach for solving structured MDP and POMDP problems. PolCA / PolCA+ share significant similarities with previous hierarchical MDP algorithms. However, we improve on these approaches in a number of ways that are essential for robotic problems.

Model minimization. First, PolCA requires less information from the human designer: s / he must specify an action hierarchy, but not the abstraction function. The automatic state abstraction is performed using an existing algorithm (Dean & Givan, 1997), which had not been previously used in the context of hierarchies. As part of this work, the algorithm of Dean and Givan was also extended to the partially observable (POMDP) case. The automated state clustering algorithm described in Section 4.2.1.3 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 (Eqn 2.18) 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 a lossless 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.

Observation abstraction. This chapter describes a novel approach to performing observation minimization. This is new to the POMDP literature. It is particularly useful for real-world applications where a large number of distinct observations can effectively be condensed in a few bits of useful discriminative information.

Policy-contingent abstraction. PolCA introduces the notion of policy-contingent abstraction. This hypothesizes that the abstract states at higher levels of the hierarchy should be left unspecified until policies at lower levels of the hierarchy are fixed. By contrast, the usual approach of specifying a *policy-agnostic* (i. e. correct for all possible policies) abstraction function often cannot obtain as much model reduction. The benefit of policy-contingent abstraction is faster planning time. The downside is the possible cost in performance (discussed in Section 4.2.4) which comes from fixing some aspects of the global policy before learning others.

POMDP hierarchical planning. Finally, PolCA extends easily to partially observable planning problems, which is of utmost importance for robotic problems. 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: typically exponential in the problem size. Thus, the potential savings one may attain through the structural decomposing of a POMDP problem are much larger.

4.6. Future Work

The algorithms described in this chapter make several key assumptions. The most important is the reliance on a human designer to provide the structural decomposition beforehand. Research on the simpler MDP paradigm has shown promise for finding good decompositions automatically (Pickett & Barto, 2002; Hengst, 2002; Ryan, 2002; McGovern & Barto, 2001; Thrun & Schwartz, 1995). The question of automatically finding task hierarchies for POMDPs remains open.

A second assumption concerns the fact that the hierarchical planning algorithm presented in this paper requires having non-trivial local reward functions in each subtask. While this poses no problem for multi-goal domains where the reward function naturally provides local reward information, it is a concern when dealing with single goal domains where, for example, only the final goal completion is rewarded. The taxi task (Section 4.2.5) is an example of such a problem. Such cases require the use of a pseudo-reward function. This property is shared with a rich body of work on MDPs (though exceptions exist), and can be thought of as another opportunity to bring to bear background knowledge a human designer might have. Nonetheless it may be useful to automatically extract subtasks with their local reward information. This is clearly related to the question of automatic subtask discovery. In the future, it is also possible that work on reward shaping (Ng, Harada, & Russell, 1999) will offer some insight into automatically defining appropriate pseudo-reward functions.

To conclude, 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.

CHAPTER 5

EXPERIMENTS IN ROBOT CONTROL

HIGH-level robot control has been a popular topic in AI, and decades of research have led to a reputable collection of robotic software architectures (Arkin, 1998; Brooks, 1986). Yet, very few of these architectures are robust to uncertainty. This chapter examines the role that POMDP planning can play in designing and fielding robust robot architectures.

The PolCA+ approach described in Chapter 4 offers a new perspective on robot architectures. Like most architectures, it provides guidelines for specifying and/or optimizing local controllers, as well as the framework to bring them together. However, unlike its predecessors, these activities are coordinated in such a way as to overcome uncertainty in both sensors and effectors. This is not a trivial task, especially when the uncertainty can occur across controller boundaries. PolCA+ is uniquely equipped to provide a scalable, robust, and convenient solution to the problem of high-level robot control.

The primary application domain for this work is that of a nursing assistant robot. The goal is to field an autonomous mobile robot that can serve as assistant and companion to an elderly person with physical and cognitive disabilities. From a technical standpoint, one of the key challenges with this project is to design a system that goes far beyond simple path planning, to also include control pertaining to user interaction, activity scheduling, and large-scale navigation. Section 5.1 describes how PolCA+ can be used to produce a multi-level structured approach in which to cast this problem.

While PolCA+ provides the backbone for structural decision-making, it offers some flexibility regarding how specific subtasks are solved. At the lower level of control, some of the tasks that arise from the hierarchy are still relatively large. For example, one aspect of the nursing home problem requires the robot to find a person wandering in the environment. Over a large area, this can require a large state space. Such a subtask cannot

be solved exactly, but offers ample opportunity to apply the PBVI algorithm of Chapter 3. This topic is covered in Section 5.2.

Section 5.3 concludes the chapter with a discussion of related work in the area of robot architectures.

While earlier chapters of this thesis focused on algorithmic development for POMDP planning, this chapter provides an in-depth look at the impact that these techniques can have in real-world applications. The experimental results presented here conclusively demonstrate the effectiveness of PolCA+ and PBVI for optimizing complex robot controllers.

5.1. Application Domain: Nursebot Project

The primary application domain is that of a mobile robotic assistant, designed to assist elderly individuals experiencing mild cognitive and physical impairments with their daily activities. In this case, a POMDP-based high-level robot controller was implemented onboard a robot platform and used to select appropriate actions and reason about perceptual uncertainty. The experiments described here were conducted as part of a larger project dedicated to the development of a prototype nursing assistant robot (Montemerlo, Pineau, Roy, Thrun, & Verma, 2002; Pollack, Engberg, Matthews, Thrun, Brown, Colbry, Orosz, Peintner, Ramakrishnan, Dunbar-Jacob, McCarthy, Montemerlo, Pineau, & Roy, 2002; Pineau, Montermerlo, Pollack, Roy, & Thrun, 2003b). 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.

From the many services a nursing-assistant robot could provide (Engelberger, 1999; Lacey & Dawson-Howe, 1998), the work reported here considers the task of reminding people of events and guiding them through their living environment. Both of these tasks are particularly relevant for the elderly community. Decreased memory capacity is a common effect of age-related cognitive decline, which often leads to forgetfulness about routine daily activities (e. g. taking medications, attending appointments, eating, drinking, bathing, toileting) thus the need for a robot that can offer cognitive reminders. In addition, nursing staff in assisted living facilities frequently need to escort elderly people walking, either to get exercise, or to attend meals, appointments or social events. The fact that many elderly people move at extremely slow speeds (e. g. 5 cm / sec) makes this one of the most labor-intensive tasks in assisted living facilities. It is also important to note that the help provided is often not strictly of a physical nature. Rather, nurses often provide important cognitive help, guidance and motivation, in addition to valuable social interaction.

Several factors make these tasks challenging ones for a robot to accomplish successfully. First, many elderly have difficulty understanding the robot's synthesized speech; some have difficulty articulating appropriate responses in a computer-understandable way. In addition, physical abilities vary drastically across individuals, social behaviors are far from uniform, and it is especially difficult to explicitly model people's behaviors, expectations, and reactions to the robot.

The robot Pearl (shown in Fig. 5.1) is the primary test-bed for the POMDP-based behavior management system. It is a wheeled robot with an onboard 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 onboard PCs connected to the Internet via wireless Ethernet.

Figure 5.1. Pearl, the robotic nursing assistant, interacting with elderly people at a nursing facility

In this domain, the PolCA+ framework of Chapter 4 can be used to build and optimize a high-level decision-making system that operates over a large set of robot activities, both verbal and non-verbal. Key actions 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.

5.1.1. POMDP Modeling

To formally test the performance of the PolCA+ algorithm in this domain, consider the following scenario. The robot Pearl is placed in an assisted living facility, where it is required to interact with elderly residents. Its primary goal is to remind them of, and take them to, scheduled physiotherapy appointments. Its secondary goal is to provide them with interesting information. In the course of the scenario, Pearl has to navigate to a resident's room, establish contact, possibly accompany the person to the physiotherapy center, and eventually return to a recharging station. The task also requires the robot to answer simple information requests by the test subject, for example providing the time or the weather forecast. Throughout this process, Pearl's high-level behavior (including both speech and motion commands) is completely governed by the POMDP interaction manager.

For this scenario, the robot interface domain is modeled using 576 states, which are described using a collection of multi-valued state features. Those states are not directly observable by the robot interface manager; however, the robot is able to perceive a number of distinct observations. The state and observation features are listed in Table 5.1.

Observations are perceived through 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 is processed and correlated to a map to determine when the robot has reached a known landmark (e. g. *Laser=robotAtHome*). Similarly, in the case of a user-emitted speech signal, a keyword filter is applied to the output of the speech recognizer (e. g. *"Give me the weather forecast for tomorrow."* $\rightarrow$ *Speech=weather*). In general, the speech recognition and touchscreen input are used as redundant sensors to each other, generating very much the same information. The *Reminder* observations are received from a high-level intelligent scheduling module. This software component, developed by McCarthy and Pollack (2002), reasons temporally about the user's activities, preferences and behaviors, with the goal of issuing appropriately timed cognitive reminders to warn the person of upcoming scheduled events (e. g. medication, doctor's appointment, social activities, etc.).

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

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**
Reminder	g_none, g_physio, g_vitamin, g_checklist
Speech	yes, no, time, weather, go, unknown
Touchscreen	t_yes, t_no, t_time, t_weather, t_go
Laser	atRoom, atPhysio, atHome
InfraRed	user, no_user
Battery	high, low

Table 5.1. Component description for human-robot interaction scenario

- COMMUNICATE={RemindPhysio, RemindVitamin, UpdateChecklist, CheckPersonPresent, TerminateGuidance, TellTime, TellWeather, ConfirmGuideToPhysio, VerifyInfoRequest, ConfirmWantTime, ConfirmWantWeather, ConfirmGoHome, ConfirmDone}
- MOVE={GotoPatientRoom, GuideToPhysio, GoHome}
- OTHER={DoNothing, RingDoorBell, RechargeBattery}

Each discrete action enumerated above invokes a scripted sequence of low-level operations on the part of the robot (e. g. *GiveWeather* 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 involve a combination of redundant speech synthesis and touchscreen display, such that the selected information or question is presented to the test subject through both modalities simultaneously. Given the sensory limitations common in our target population, the use of redundant audio-visual communication is important, both for input to, and output from, the robot. The actions in the *Move* category are 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).

PolCA+ requires both an action hierarchy and model of the domain to proceed. The hierarchy (shown in Fig. 5.2) was designed by hand. Though the model could be learned from experimental data, the prohibitive cost of gathering sufficient data from our elderly users makes this an impractical solution. Therefore, the POMDP model parameters were selected by a designer. The reward function is chosen to reflect the relative costs of applying actions in terms of robot resources (e. g. robot motions actions are typically costlier than

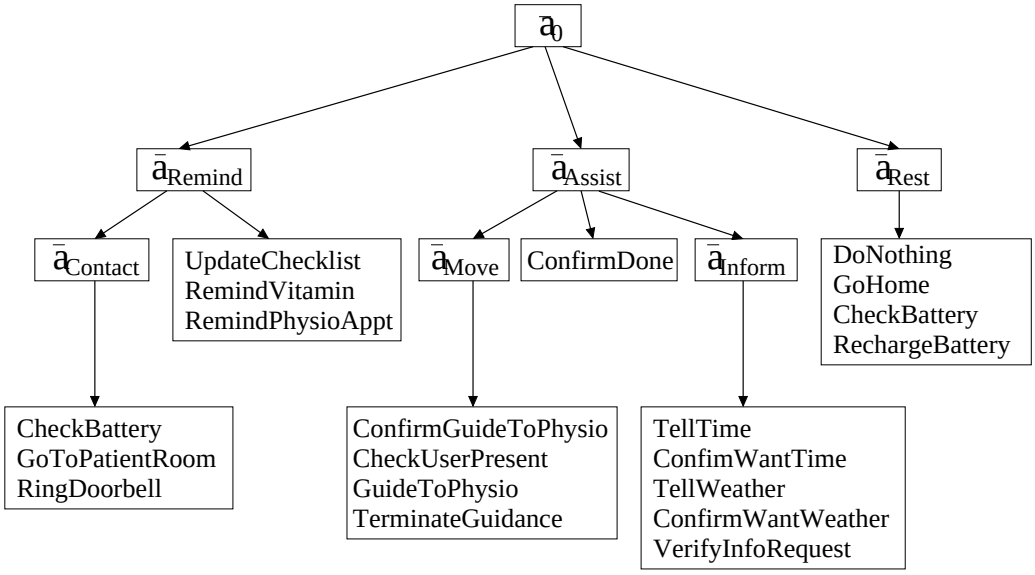


Figure 5.2. Action hierarchy for Nursebot domain

spoken verification questions), as well as reflecting the appropriateness of the action with respect to the state. For example:

- positive rewards are given for correctly satisfying a goal, e. g.

  R($a = TerminateGuidance$)=+50

  if $s(MotionGoal = toPhysio,\ RobotLocation = physio,\ PersonLocation = physio)$
- large negative rewards are given for applying an action unnecessarily, e. g.

  R($a = GuidetoPhysio$)=−200

  if $s(MotionGoal = none)$
- small negative rewards are given for verification questions, e. g.

  R($a = ConfirmGuidetoPhysio$)=−1

  given any state condition.

The problem domain described here is well within the reach of existing MDP algorithms. However, the main challenge is the fact that the robot's sensors are subject to substantial noise, and therefore the state is not fully observable. Noise in the robot's sensors arise mainly from its speech recognition software. For example, a robot may easily mistake phrases like *"get me the time"* and *"get me my medicine"*, but whereas one involves motion, the other does not. Thus, considering state uncertainty is of great importance in this domain. In particular, it is important to trade-off the cost of asking a clarification question, versus that of accidentally executing the wrong command. Uncertainty also arises as a result of human behavior, for example when a user selects the wrong option on the touch

pad, or changes his/her mind. Finally, and to a much lesser degree, noise arises from in the robot's location sensors. In any of these eventualities, MDP techniques are inadequate to robustly control the robot. The PolCA+ algorithm on the other hand can significantly improve the tractability of POMDP planning, to the point where we can rely on POMDP-based control for this real-world domain.

5.1.2. Experimental Results

Because of the difficulties involved with conducting human subject experiments, only the final PolCA+ policy was deployed onboard the robot. Nonetheless, its performance can be compared in simulation with that of other planners. We first compare state abstraction possibilities between PolCA (which falsely assumes full observability) and PolCA+ (which considers similarity in observation probabilities before clustering states). This is a direct indicator of model reduction potential, and equivalently, planning time. Figure 5.3 shows significant model compression for both PolCA and PolCA+ compared to the no-abstraction case (*NoAbs*). Differences between PolCA and PolCA+ arise when certain state features, though independent with respect to transitions and rewards, become correlated during belief tracking through the observation probabilities.

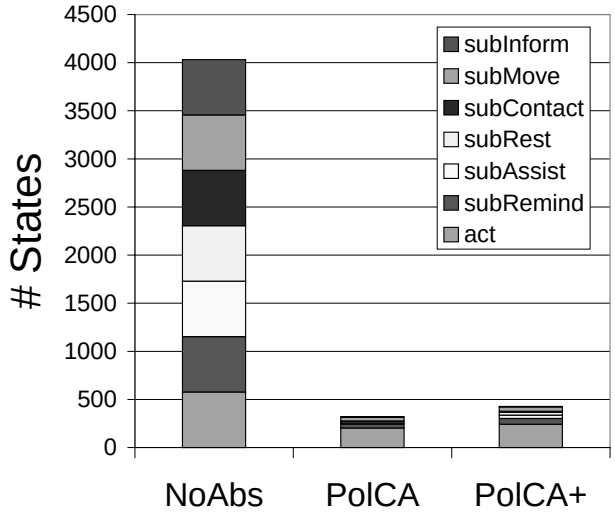


Figure 5.3. Number of parameters for Nursebot domain

Second, we compare the reward gathered over time by each policy. As shown in Figure 5.4, PolCA+ clearly outperforms PolCA in this respect. A closer look at the performance of PolCA reveals that it often answers a wrong query because it is unable to appropriately select among clarification actions. In other instances, the robot prematurely terminates an interaction before the goal is met, because the controller is unable to ask the user whether

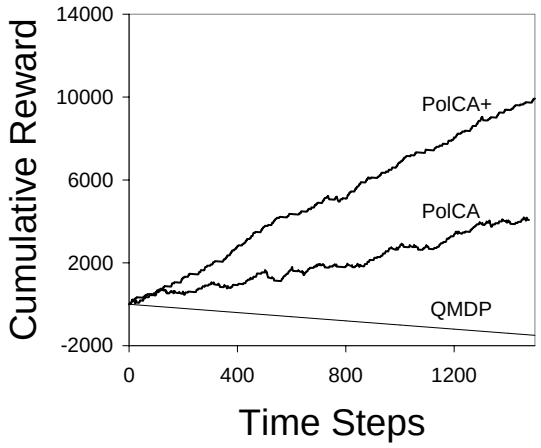


Figure 5.4. Cumulative reward over time in Nursebot domain

s/he is done. In contrast, PolCA+ resorts to confirmation actions to avoid wrong actions, and satisfy more goals. Also included in this comparison is QMDP (see Section 2.2.4). On this task, it performs particularly poorly, repeatedly selecting to *doNothing* because of its inability to selectively gather information on the task at hand.

In terms of computation time, PolCA+ reached its solution in 18 minutes. In comparison, Incremental Pruning (an exact POMDP solver), could only complete 2 iterations of value iteration in 24hrs, and thus would probably take many years to reach a reasonable solution. The many-fold improvement found in PolCA+ is for the most part due to its structural assumptions (hierarchy+abstraction); some of the improvement is also achieved by using an AMDP solver at the highest level of the hierarchy (lower subtasks are solved exactly). The PolCA solution was computed in only 4 seconds, whereas the (unstructured) QMDP solution took 39 seconds.

The PolCA+ policy was the only one implemented onboard the robot. It was tested during two days of experiments with elderly residents at a local nursing home. Through the course of twelve interaction scenarios, Pearl was able to successfully deliver scheduled reminders, guide residents to physiotherapy appointments, and satisfy information requests. The robustness of the PolCA+ policy to uncertainty was demonstrated by its use of clarification questions whenever a user's intentions were unclear.

Overall, the policy generated using PolCA+ successfully controlled the robot without any human intervention, in all guidance experiments. As a result, all six test subjects were able to complete the full experimental scenario after receiving only limited training (a five-minute introduction session). All subjects were uniformly positive about the experience. Table 5.2 shows a typical interaction between the robot and user, in terms of the

Table 5.2. A sample dialogue with a test subject

Observation	Action	Reward
(null)	DoNothing	-1
Reminder=*g_physio*	gotoPatientRoom	5
Laser=*atRoom*	RingBell	5
Speech=*yes*	RemindPhysio	50
Speech=*unknown*	**ConfirmGuideToPhysio**	-5
Speech=*yes*	**CheckBattery**	-5
Battery=*high*	GuideToPhysio	50
Laser=*atPhysio*	**CheckUserPresent**	-1
IR=*no_user*	**CheckUserPresent**	-1
IR=*user*	**CheckUserPresent**	-5
IR=*user*	TerminateGuidance	50
Speech=*unknown*	**ConfirmDone**	-1
Speech=*no*	**VerifyInfoRequest**	-1
Speech=*weather*	**ConfirmWantWeather**	-1
Speech=*unknown*	**VerifyInfoRequest**	-1
Speech=*weather*	**ConfirmWantWeather**	-1
Speech=*yes*	TellWeather	50
Speech=*unknown*	**ConfirmDone**	-1
Speech=*yes*	GoHome	5
Laser=*atHome*	RechargeBattery	20

observations received by the controller and the actions selected in response, as well as the corresponding reward signals. Actions in bold font are clarification actions, generated by the POMDP because of high uncertainty.

Step-by-step images corresponding to the interaction between Pearl and one of the test subjects are shown in Figure 5.5. The sequence of images illustrates the major stages of a successful delivery: 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 regarding the weather forecast. Throughout this interaction, communication took place through speech and the touch-sensitive display.

5.1.3. Discussion

Throughout the experiment, speech recognition performance was particularly poor due to the significant amount of ambient noise, however the redundancy offered by the touch-screen allowed users to communicate with the dialogue manager without difficulty. In addition, during early experiments, the robot lacked the ability to adapt its speed to that of the person. While guiding someone with reduced mobility to the physiotherapy center, it would simply run away because it could not monitor the person's progress. This was corrected by the addition of a second laser in the back of the robot, allowing it to adapt its speed appropriately.

(a) Pearl approaching elderly

(b) Reminding of appointment

(c) Guidance through corridor

(d) Entering physiotherapy dept.

(e) Asking for weather forecast

(f) Pearl leaves

Figure 5.5. Example of a successful guidance experiment

This experiment constitutes encouraging evidence that, with appropriate approximations, POMDP control can be feasible and useful in real-world robotic applications.

5.2. Application domain: Finding Patients

The Nursebot domain described above covers a large spectrum of robot abilities. To complete the full scenario, the robot must combine knowledge from a number of different sensors, and prioritize goals between the various modules. In order to keep the problem manageable, the focus is placed on *high-level* control. This means that many state variables and control actions operate at very high-level resolution. For example, locations are identified through a number of landmarks (e. g. *patient's room*, *physiotherapy office*, *robot's home base*), and motion commands operate at an equally high resolution (e. g. *GoToPatientRoom*, *GuideToPhysio*, *GoHome*). While these assumptions can be sufficient for some relatively structured interactions, in general it should be expected that the user can and will wander around the facility.

This section takes a closer look at the question of how the robot can find a non-stationary patient. This subtask of the Nursebot domain shares many similarities with the Tag problem presented in Section 3.5.2. In this case, however, a robot-generated map of a real physical environment is used as the basis for the spatial configuration of the domain. This map is shown in Figure 5.6. The white areas correspond to free space, the black lines indicate walls (or other obstacles) and the dark gray areas are not visible or accessible to the robot. One can easily imagine the patient's room and physiotherapy unit lying at either end of the corridor, with a common area shown in the upper-middle section.

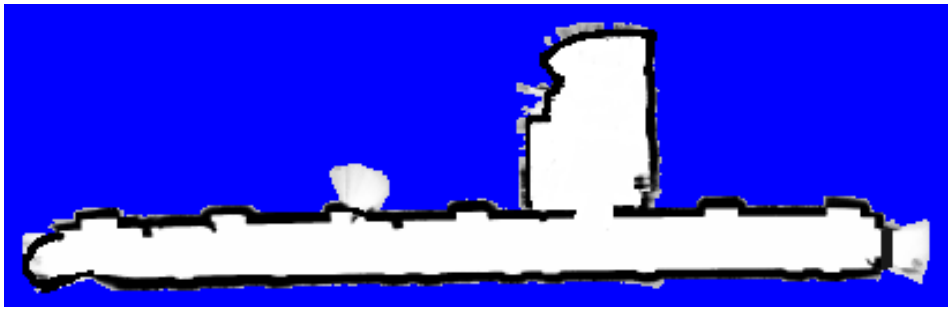


Figure 5.6. Map of the environment

The overall goal is for the robot to traverse the domain in order to find the missing patient and then deliver a message. The robot must systematically explore the environment, reasoning about both spatial coverage and human motion patterns in order to find the wandering person.

5.2.1. POMDP Modeling

The problem domain is represented jointly by two state features: *RobotPosition*, *PersonPosition*. Each feature is expressed through a discretization of the environment. Most of the experiments below assume a discretization of 2 meters, which means 26 discrete cells for each feature, or a total of 676 states.

It is assumed that the person and robot can move freely throughout this space. The robot's motion is deterministically controlled by the choice of action (*North*, *South*, *East*, *West*). The robot has a fifth action (*DeliverMessage*), which concludes the scenario when used appropriately (i. e. when the robot and person are in the same location).

The person's motion is stochastic and falls in one of two modes. Part of the time, the person moves according to Brownian motion (e. g. moves in each cardinal direction with $Pr = 0.1$, otherwise stays put). At other times, the person moves directly away from the robot. The Tag domain of Section 3.5.2 assumes that the person always moves always moves away the robot. This is not realistic when the person cannot see the robot. The current experiment instead assumes that the person moves according to Brownian motion when the robot is far away, and moves away from the robot when it is closer (e. g. < 4m).

In terms of state observability, there are two components: what the robot can sense about its own position, and what it can sense about the person's position. In the first case, the assumption is that the robot knows its own position at all times. While this may seem like a generous (or optimistic) assumption, substantial experience with domains of this size and maps of this quality have demonstrated very robust localization abilities (Thrun et al., 2000). This is especially true when planning operates at relatively high resolution (2 meters) compared to the localization precision (10 cm). While exact position information is assumed for *planning* in this domain, the *execution* phase does update the belief using full localization information, which includes positional uncertainty whenever appropriate.

Regarding the detection of the person, the assumption is that the robot has no knowledge of the person's position unless s/he is within a range of 2 meters. This is plausible given the robot's sensors. However, even in short-range, there is a small probability ($Pr = 0.01$) that the robot will miss the person and therefore return a false negative.

In general, one could make sensible assumptions about the person's likely position (e. g. based on a knowledge of their daily activities), however we currently have no such information and therefore assume a uniform distribution over all initial positions. The person's subsequent movements are expressed through the motion model described above.

The reward function is straightforward: $R = -1$ for any motion action, $R = 10$ when the robot decides to *DeliverMessage* and it is in the same cell as the person, and $R = -100$ when the robot decides to *DeliverMessage* in the person's absence. The task terminates when the robot successfully delivers the message (i. e. $a = DeliverMessage$ and $s_{robot} = s_{person}$). We assume a discount factor of 0.95.

5.2.2. Experimental Results

The subtask described here, with its 626 states, is beyond the capabilities of exact POMDP solvers. Furthermore, as will be demonstrated below, MDP-type approximations are not equipped to handle uncertainty of the type exhibited in this task. The main purpose of this section is therefore to evaluate the effectiveness of the PBVI approach described in Chapter 3 to address this problem. While the results on the Tag domain (Section 3.5.2) hint at the fact that PBVI may be able to handle this task, the more realistic map and modified motion model provide new challenges.

PBVI is applied to the problem as stated above, alternating value updates and belief point expansions until (in simulation) the policy is able to find the person on $> 90\%$ of trials (trials were terminated when the person is found or it exceeds 100 steps). The planning phase required 40000 seconds (approx. 11 hours) on a 1.2 GHz Pentium II.

The resulting policy is illustrated in Figure 5.7. This figure shows six snapshots obtained from a single run. In this particular scenario, the person starts at the far end of the left corridor. The person's location is not shown in any of the figures since it is not observable by the robot. The figure instead shows the *belief* over person positions, represented by a distribution of point samples (grey dots in Fig. 5.7). Each point represents a plausible hypothesis about the person's position. The figure also shows the robot starting at the far right end of the corridor (Fig. 5.7a). The robot moves toward the left until the room's entrance (Fig. 5.7b). It then proceeds to check the entire room (Fig. 5.7c). Once certain that the person is nowhere to be found, it exits the room (Fig. 5.7d), and moves down the left branch of the corridor, where it finally finds the person at the very end of the corridor (Fig. 5.7e).

This policy is optimized for any start positions (for both the person and the robot). The scenario shown in Figure 5.7 is one of the longer execution traces since the robot ends up searching the entire environment before finding the person. It is interesting to compare the choice of action between snapshots (b) and (d). The robot position in both is practically identical. Yet in (b) the robot chooses to go up into the room, whereas in (d) the robot chooses to move toward the left. This is a direct result of planning over *beliefs*, rather than

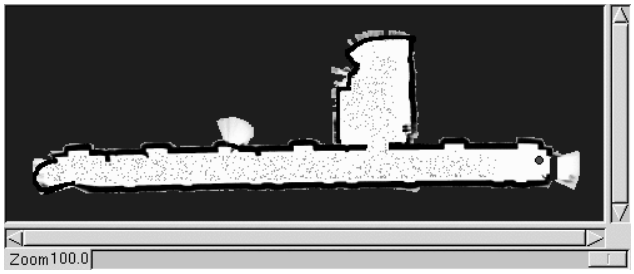
over *states*. The belief distribution over person positions is clearly different between those two cases, and therefore the policy specifies a very different course of action.

The sequence illustrated in Figure 5.7 is the result of planning with over 3000 belief points. It is interesting to consider what happens with fewer belief points. Figure 5.8 shows such a case. The scenario is the same, namely the person starts at the far left end of the corridor and the robot start at the far right end (Fig. 5.8a). The robot then navigates its way to the doorway (Fig. 5.8b). It enters the room and looks for the person in a portion of the room (Fig. 5.8c). Unfortunately an incomplete plan forces it into a corner (Fig. 5.8d) where it stays until the scenario is forcibly terminated. Using this policy (and assuming uniform random start positions for both robot and person), the person is only found in 40% of trials, compared to 90% using the policy shown in Figure 5.7. Planning in this case was done with 443 belief points, and required approximately 5000 seconds.

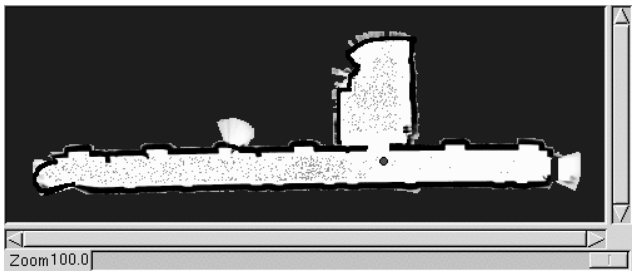
Figure 5.9 looks at the policy obtained when solving this same problem using the QMDP heuristic. Once again, six snapshots are offered from different stages of a specific scenario, assuming the person started on the far left side and the robot on the far right side (Fig. 5.9a). After proceeding to the room entrance (Fig. 5.9b), the robot continues down the corridor until it *almost* reaches the end (Fig. 5.9c). It then turns around and comes back toward the room entrance, where it stations itself (Fig. 5.9d) until the scenario is forcibly terminated. As a result, the robot cannot find the person when s/he is at the left edge of the corridor. What's more, because of the running-away behavior adopted by the subject, even when the person starts elsewhere in the corridor, as the robot approaches the person will gradually retreat to the left and similarly escape from the robot. Planning with the QMDP heuristic required 200 seconds.

Even though QMDP does not explicitly plan over *beliefs*, it can generate different policy actions for cases where the state is identical but the belief is different. This is seen when comparing Figure 5.9 (b) and (d). In both of these, the robot is identically located, however the belief over person positions is different. In (b), most of the probability mass is to the left of the robot, therefore it travels in that direction. In (d), the probability mass is distributed evenly between the three branches (left corridor, room, right corridor). The robot is equally pulled in all directions and therefore stops there. This scenario illustrates some a strength of QMDP. Namely, there are many cases where it is not necessary to explicitly reduce uncertainty. However, it also shows that more sophisticated approaches are needed to handle some cases.

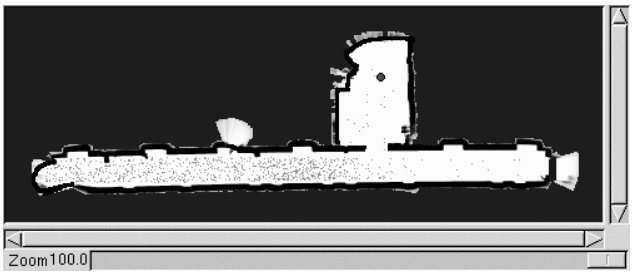
(a) t=1



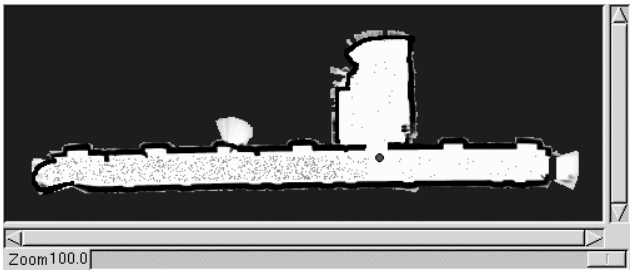
(b) t=7



(c) t=12



(d) t=17



(e) t=29

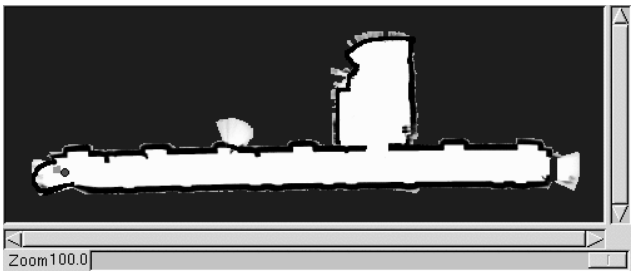
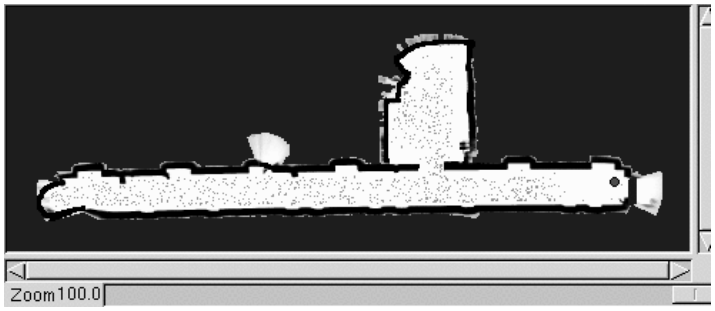
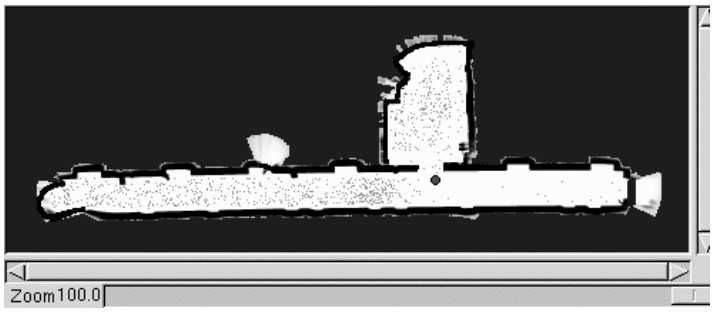


Figure 5.7. Example of a PBVI policy successfully finding the patient

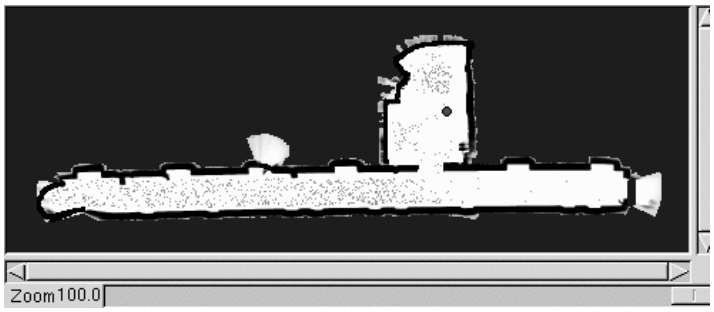
(a) t=1



(b) t=7



(c) t=10



(d) t=12

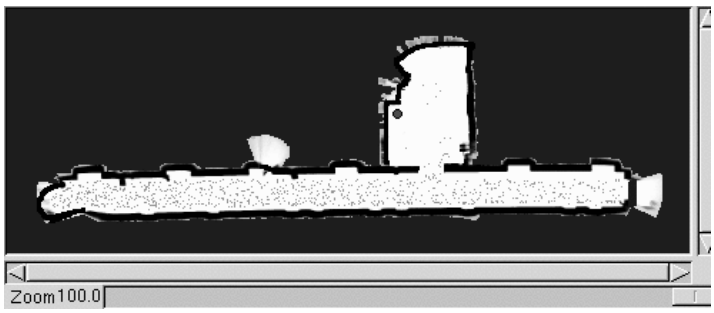
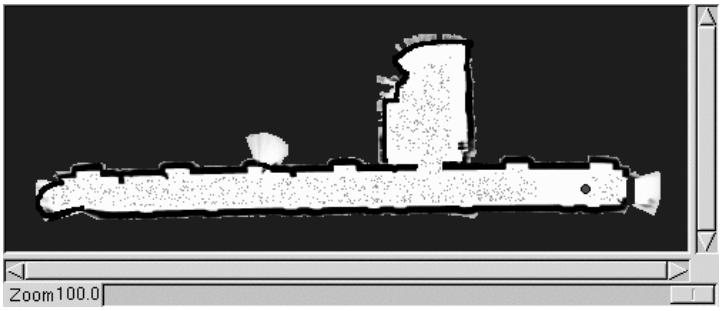
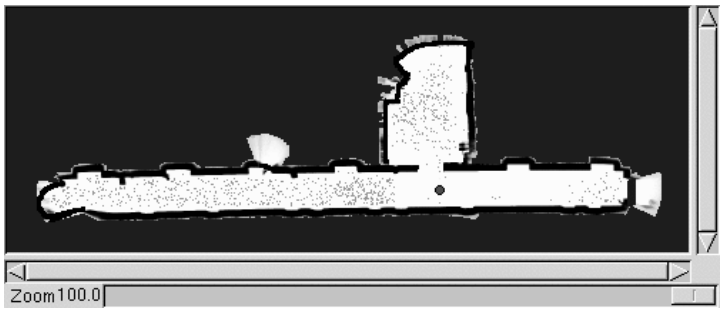


Figure 5.8. Example of a PBVI policy failing to find the patient

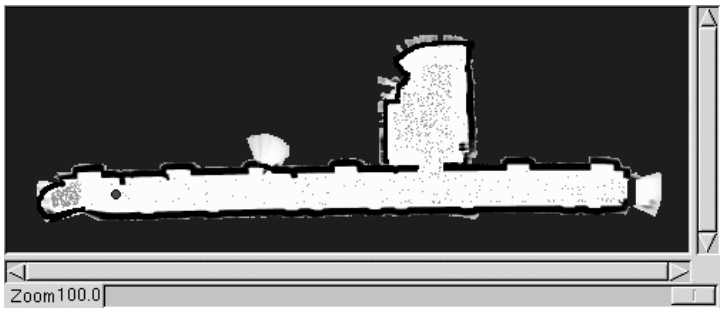
(a) t=1



(b) t=7



(c) t=17



(d) t=27

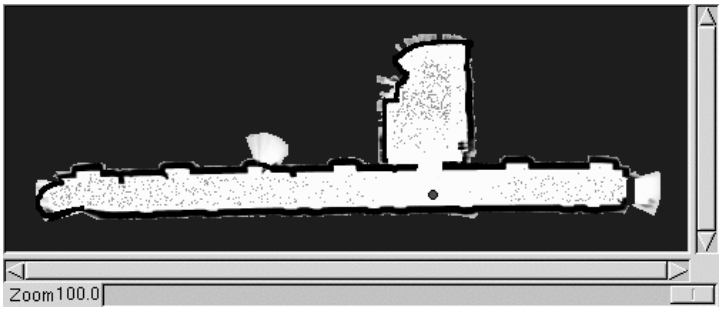


Figure 5.9. Example of a QMDP policy failing to find the patient

5.2.3. Discussion

These experiments conclusively demonstrate that PBVI is the appropriate tool for solving large subtasks. A few issues are still outstanding.

As mentioned in Chapter 3, whenever PBVI is used to solve a subtask h within PolCA+, it is crucial that PBVI use belief points generated using the *full set of actions* (A), rather than the reduced subtask specific action set (A_h). Using the reduced set A_h could produce a useful solution in many instances. But it is likely that there exists some belief that is not reachable using A_h, from which the parent subtask could decide to call h. In such an instance, the local policy π_h would perform very poorly.

The fact that the belief point *expansion* phase has to occur over the entire belief space does not in any way reduce the savings gained from PolCA+'s hierarchy and abstraction during the *planning* phase. Since planning is by far the slower of the two, this question of global versus local belief expansion is a very minor issue with respect to computation time. One obvious advantage of generating beliefs globally is that the belief points can then be re-used for all subtasks.

5.3. Related work

PolCA+ is a new paradigm for robot control architectures. There is a rich literature in this area, and some of the most successful robot systems rely on structural assumptions very similar to PolCA+'s to tackle large-scale control problems (Arkin, 1998; Russell & Norvig, 2002).

The Subsumption architecture (Brooks, 1986) builds scalable control systems by combining simple reactive controllers. Complex tasks can be partitioned among a hierarchy of such controllers. A controller is usually expressed through a finite state machine, where nodes contain tests used to condition action choice on sensor variables. Appropriate controller-specific state abstraction can be leveraged to improve scalability. The Subsumption architecture, and other similar approaches, rely on human designers to specify all structural constraints (hierarchy, abstraction), and in some cases even the policies contained in each finite state machine. This can require significant time and resources, and often lead to sub-optimal solutions. Another limitation results from the fact that the test nodes in the reactive controllers are usually conditioned on raw sensor input.

A related approach is the popular three-layer architecture (Firby, 1989; Gat, 1998). As the name implies, it assumes a three-level hierarchy. At the bottom is the reactive layer, which provides fast low-level control that is tightly coupled to recent sensor observations. At the top is the deliberative layer where search routines handle global plans. In between

those two is the executive layer, which tracks the internal state (based on sensor information) and uses it to translate the goals from the top-level into low-level reactive behaviors. This general approach provides the basic architecture for a large number of robot systems (Connell, 1991; Gat, 1992; Elsaessar & Slack, 1994; Firby, 1996; Bonasso, Firby, Gat, Kortemkamp, Miller, & Slack, 1997).

GOLOG (Levesque, Reiter, Lesperance, Lin, & Scherl, 1997) is not strictly a robot architecture, but rather a robotic programming language, which has been used for high-level control of indoor robots. In GOLOG the task is expressed through a control program that integrates reactivity and deliberation within a single framework. A designer must provide a model of the robot and its environment. S/he also has the option of including partial policies. A planning routine optimizes other action choices.

All the approaches discussed here assume full observability. This means that they are best suited to domains where sensor data is sufficiently reliable and complete for good decision-making. For domains where this is not the case, PolCA+'s ability to handle uncertainty, perform automated state abstraction, and optimize policies, are significant improvements over earlier robot architectures.

5.4. Contributions

Using the structural framework of PolCA+, it is possible to build a flexible multi-level robot control architecture that handles uncertainty obtained through both navigation sensors (e. g. laser range-finder) and interaction sensors (e. g. speech recognition and touchscreen). In combination with PBVI, it can solve even large subtasks. We believe PolCA+'s ability to perform automated state abstraction and policy learning, as well as handle uncertainty, are significant improvements over earlier robot architectures.

To the best of our knowledge, the work presented in this chapter constitutes the first instance of a POMDP-based architecture for robot control. It was a key element for the successful performance of the Nursebot in a series of experiments with elderly users.

5.5. Future work

The experiments described in this chapter are the early steps of the Nursebot project. A substantial program of research and prototyping is necessary on the path toward having fully autonomous robotic assistants living alongside elderly people.

CHAPTER 6

CONCLUSION

THE problem of planning under uncertainty is relevant to a large number of fields, from manufacturing to robotics to medical diagnosis. In the area of robotics, it is generally understood to mean the ability to produce action policies that are robust to sensory noise, imprecise actuators and so on. This is imperative for robot systems deployed in real-world environments. For example, a robot designed as an assistant or companion for a human user clearly needs an action strategy that allows it to overcome unpredictable human behavior and mis-communications, while accomplishing its goal.

The Partially Observable Markov Decision Process offers a rich framework for performing planning under uncertainty. It can be used to optimize sequences of actions with respect to a reward function, while taking into account both effect and state uncertainty. POMDPs can be used to model a large array of robot control problems. However, finding a solution in reasonable time is often impossible, even for very small problems. One of the key obstacles to increased scalability of POMDPs is the curse of history, namely the fact that the number of information states grows exponentially with the planning horizon.

It is the focus of this thesis to develop computationally tractable solutions for large POMDP problems, and to demonstrate their effectiveness in robotic applications. In support of this goal, this document describes two algorithms that exploit structural properties to overcome the curse of history, and produce scalable approximate solutions for POMDP problems.

6.1. PBVI: Point-based value iteration

The first of the two algorithms is named PBVI. It combines an explorative sampling of the set of information states with fast point-based dynamic programming updates. Its explorative belief-point selection ensures good coverage of the belief simplex, and therefore

good performance under a wide range of uncertainty conditions with relatively few points. The dynamic programming updates can be computed efficiently since they are expressed over a fixed (small) number of points.

PBVI builds on a number of earlier approximation algorithms, which use similar point-based dynamic programming updates. The main contribution here is in how such point-based updates can be combined with an exploratory belief sampling heuristic. The result is an anytime algorithm that produces solutions that have bounded error with respect to the optimal.

Part of the appeal of PBVI is in the relative simplicity of the algorithm. It can be implemented quickly, given a basic understanding of POMDPs. And other than the domain model, the algorithm itself requires very few parameters to run.

It is an effective algorithm for solving POMDP problems on the order of 10^3 states. It can address a wide range of problems, with varying levels of uncertainty, from the localization uncertainty exhibited by the maze domains (Section 3.5.1), to the global search required to find a missing person (Section 5.2).

It is less effective for problems requiring very large (multi-feature) state spaces, since dynamic programming updates operate over the full-dimensional belief simplex. It does not yet take advantage of dimensionality reduction or function-approximation techniques, though these suggest a promising direction for future extensions.

PBVI's current heuristic for selecting belief points is somewhat primitive: simulate single-step forward belief propagation using all actions and keep the new belief that is farthest from the current set of beliefs. It is remarkably effective compared to other equally naive heuristics (e. g. simulate single-step forward belief propagation using a random action). But, it is likely that more sophisticated and better performing techniques can be devised. The objective, when selecting a new belief sampling heuristic, will be to find one that reduces the number of belief points while preserving (or improving) solution quality.

6.2. PolCA+: Policy-contingent abstraction

The second algorithm, PolCA+, addresses complex problems by partitioning them into smaller ones that can be solved quickly. The decomposition constraints are expressed through an action-based subtask hierarchy. Each subtask is defined over a reduced set of actions, states, and observations. Subtasks are solved individually, and their solutions are re-combined (according to the hierarchy) to produce a global solution.

PolCA+ builds on earlier hierarchical MDP approaches, which adopt a similar action-based hierarchy. The main innovation of PolCA+ is two-fold. First, it introduces the concept of policy-contingent abstraction. In short, this means that whenever a lower-level subtask is solved before its parent, the parent subtask will be afforded greater state abstraction. Greater state abstraction generally means faster planning time. Second, PolCA+ insures that the elements required for partial observability are in place (single-step parameterization of abstract actions, observation abstraction, polling execution). The impact of this approach is clear, namely increased robustness for partially observable domains, which covers a large number of robotic tasks.

The driving force behind PolCA+ is the well-known principle of divide-and-conquer. As such, PolCA+ is best suited for domains that exhibit natural structure. It gains computational advantage through both the action hierarchy (which yields subtasks with small action sets) and through subtask-specific state/observation abstraction. PolCA+ is most effective when there are tight local couplings between actions and states. This means problems where certain actions affect certain states, and these nodes of inter-dependent states and actions are relatively small.

Fortunately, many real-world domains possess such structure. A prime example is that of the nursing assistant robot, which is discussed at length in this thesis. In that case, the structure comes from the different modules featured in the robot (e. g. communication interface, navigation, scheduling), each of which focuses on a small number of relevant actions and state features. Applying PolCA+ to this domain produces a high-level robot controller that can satisfy a number of tasks, while handling uncertainty pertaining to the environment, the human user, and the robot itself. This domain is by no means unique. Many other robots are faced with multi-task domains that could be addressed through structural decomposition.

PolCA+ has much in common with some of the existing structured robot control architectures, for example the Subsumption architecture. The structural assumptions are similar, and the overall goal is the same, namely to produce scalable robot controllers. However PolCA+ brings additional insight, namely the realization that it is imperative to consider uncertainty at all levels of control. It is not sufficient to rely on low-level reactive controllers to handle unexpected events. Because it considers uncertainty at the highest-level of control, PolCA+ provides a framework where one can effectively reason about global uncertainty, as well as prioritize and switch between subtasks. In addition, PolCA+ is able to automatically find state abstraction and optimize subtask policies, while other architectures rely on designers to provide these.

6.3. Summary

Most POMDPs of the size necessary for good robot control are far too large to be solved exactly. However, many problems naturally exhibit strong structural properties. By designing algorithms that exploit such structure, it is possible to produce high quality approximate solutions in reasonable time.

This thesis considers the leveraging of structural constraints in POMDPs from many angles, from sparse belief space sampling, to explicit action hierarchy, to automatic state minimization and observation abstraction. These provide powerful approximation possibilities for POMDP solving. Taken together, these techniques are key to the design and development of planning and control systems that are scalable, modular, and robust to uncertainty.

Bibliography

Akella, S., Huang, W. H., Lynch, K. M., & Mason, M. T. (1997). Sensorless parts orienting with a one-joint manipulator. In *Proceedings of the 1997 IEEE International Conference on Robotics & Automation (ICRA)*, pp. 2383–2390.

Ambros-Ingerson, J., & Steel, S. (1988). Integrating planning, execution and monitoring. In *Proceedings of the Seventh National conference on Artificial Intelligence (AAAI)*, pp. 735–740.

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.

Ästrom, K. J. (1965). Optimal control of markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications*, *10*, 174–205.

Bagnell, J. A., & Schneider, J. (2001). Autonomous helicopter control using reinforcement learning policy search methods. In *Proceedings of the 2001 IEEE International Conference on Robotics & Automation (ICRA)*, pp. 1615–1620.

Baird, L. C., & Moore, A. W. (1999). Gradient descent for general reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, Vol. 11, pp. 968–974.

Barrett, A., & Weld, D. S. (1994). Task-decomposition via plan parsing. In *Proceedings of the Twelfth National conference on Artificial Intelligence (AAAI)*, pp. 1117–1122.

Baxter, J., & Bartlett, P. L. (2000). GPOMDP: An on-line algorithm for estimating performance gradients in POMDP's, with applications. In *Machine Learning: Proceedings of the 2000 International Conference (ICML)*, pp. 41–48.

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.

Bell, C., & Tate, A. (1985). Using temporal constraints to restrict search in a planner. In *Proceedings of the Third Alvey IKBS SIG Workshop*.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press.

Bertoli, P., Cimatti, A., & Roveri, M. (2001). Heuristic search + symbolic model checking = efficient conformant planning. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 467–472.

Bertsekas, D. P., & Tsitsiklis, J. (1996). *Neuro-Dynamic Programming*. Athena Scientific.

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

Blum, A. L., & Furst, M. L. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, pp. 281–300.

Blythe, J. (1998). *Planning under Uncertainty in Dynamic Domains*. Ph.D. thesis, Carnegie Mellon University, Department of Computer Science.

Bonasso, R. P., Firby, R. J., Gat, E., Kortemkamp, D., Miller, D. P., & Slack, M. G. (1997). Experiences with an architecture for intelligent reactive agents. *Journal of Experimental and Theoretical AI*, *9*(2), 237–256.

Bonet, B. (2002). An epsilon-optimal grid-based algorithm for partially obserable Markov decision processes. In *Machine Learning: Proceedings of the 2002 International Conference (ICML)*, pp. 51–58.

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

Boutilier, C. (2002). A POMDP formulation of preference elicitation problems. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI)*, pp. 239–246.

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., Dean, T., & Hanks, S. (1999). Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, *11*, 1–94.

Boutilier, C., & Poole, D. (1996). Computing optimal policies for partially observable decision processes using compact representations. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI)*, pp. 1168–1175.

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. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, *2*(1), 14–23.

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 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.

Castañon, D. A. (1997). Approximate dynamic programming for sensor management. In *Conference on Decision and Control*.

Chapman, D. (1987). Planning for conjunctive goals. *Artificial Intelligence*, *32*(3), 333–377.

Cheng, H.-T. (1988). *Algorithms for Partially Observable Markov Decision Processes*. Ph.D. thesis, University of British Columbia.

Connell, J. (1991). SSS: A hybrid architecture applied to robot navigation. In *Proceedings of the 1991 IEEE International Conference on Robotics & Automation (ICRA)*, pp. 2719–2724.

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., & Kanazawa, K. (1988). Probabilistic temporal reasoning. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI)*, pp. 524–528.

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

Draper, D., Hanks, S., & Weld, D. (1994). A probabilistic model of action for least-commitment planning with information gathering. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 178–186.

Elsaessar, C., & Slack, M. (1994). Integrating deliberative planning in a robot architecture. In *Proceedings of the AIAA Conference on Intelligent Robots in Field, Factory, Service and Space (CIRFFSS)*, pp. 782–787.

Engelberger, G. (1999). *Handbook of Industrial Robotics*. John Wiley and Sons.

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.

Fine, S., Singer, Y., & Tishby, N. (1998). The hierarchical hidden Markov model: Analysis and applications. *Machine Learning*, *32*, 41–62.

Firby, R. J. (1989). *Adaptive execution in dynamic domains*. Ph.D. thesis, Yale University.

Firby, R. J. (1996). Programming chip for the IJCAI-95 robot competition. *AI Magazine*, 71–81.

Friedman, J. H., Bengley, J. L., & Finkel, R. A. (1977). An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, *3*(3), 209–226.

Gat, E. (1992). Integrating planning and reaction in an heretogeneous asynchronous architecture for controlling mobile robots. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI)*, pp. 809–815.

Gat, E. (1998). *Artificial Intelligence and Mobile Robots*, chap. Three-layer architectures, pp. 195–210. AAAI Press.

Goldman, R. P., & Boddy, M. S. (1994). Conditional linear planning. In *Proceedings of the Second International Conference on AI Planning Systems (AIPS)*, pp. 80–85.

Goldman, R. P., & Boddy, M. S. (1996). Expressive planning and explicit knowledge. In *Proceedings of the Third International Conference on AI Planning Systems (AIPS)*, pp. 110–117.

Hansen, E. A. (1998). Solving POMDPs by searching in policy space. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 211–219.

Hauskrecht, M. (1997). Incremental methods for computing bounds in partially observable Markov decision processes. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI)*, pp. 734–739.

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

Hengst, B. (2002). Discovering hierarchy in reinforcement learning with HEXQ. In *Machine*

Learning: Proceedings of the 2002 International Conference (ICML), pp. 243–250.

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

Hoare, C. A. R. (1961). Find (algorithm 65). *Communications of the ACM*, *4*, 321–322.

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

Jonsson, A., & Barto, A. G. (2001). Automated state abstraction for options using the U-Tree algorithm. In *Advances in Neural Information Processing Systems (NIPS)*, Vol. 13, pp. 1054–1060.

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.

Kakade, S. (2002). A natural policy gradient. *Advances in Neural Information Processing Systems (NIPS)*, *14*, 1531–1538.

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

Kautz, H., & Selman, B. (1992). Planning as satisfiability. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI)*, pp. 359–379.

Kearns, M., Mansour, Y., & Ng, A. Y. (2000). Approximate planning in large POMDPs via reusable trajectories. *Advances in Neural Information Processing Systems (NIPS)*, *12*, 1001–1007.

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

Lacey, G., & Dawson-Howe, K. M. (1998). The application of robotics to a mobility aid for the elderly blind. *Robotics and Autonomous Systems*, *23*, 245–252.

Levesque, H. J., Reiter, R., Lesperance, Y., Lin, F., & Scherl, R. B. (1997). GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, *31*(1-3), 59–84.

Littman, M. L. (1996). *Algorithms for Sequential Decision Making*. Ph.D. thesis, Brown University.

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

Littman, M. L., Cassandra, A. R., & Kaelbling, L. P. (1995b). Learning policies for partially

obsevable environments: Scaling up. Tech. rep. CS-95-11, Brown University, Department of Computer Science.

Littman, M. L., Sutton, R. S., & Singh, S. (2002). Predictive representations of state. In *Advances in Neural Information Processing Systems (NIPS)*, Vol. 14, pp. 1555–1561.

Lovejoy, W. S. (1991a). Computationally feasible bounds for partially observed Markov decision processes. *Operations Research*, *39*(1), 162–175.

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

McAllester, D., & Roseblitt, D. (1991). Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI)*, pp. 634–639.

McCallum, A. K. (1996). *Reinforcement Learning with Selective Perception and Hidden State*. Ph.D. thesis, University of Rochester.

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)*, pp. 243–252.

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. (1999). Very fast EM-based mixture model clustering using multiresolution KD-trees. In *Advances in Neural Information Processing Systems (NIPS)*, Vol. 11, pp. 543–549.

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.

Ng, A. Y., & Jordan, M. (2000). PEGASUS: A policy search method for large MDPs and POMDPs. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 405–415.

Ng, A. Y., Parr, R., & Koller, D. (2000). Policy search via density estimation. In *Advances in Neural Information Processing Systems (NIPS)*, Vol. 12.

Nourbakhsh, I., Powers, R., & Birchfield, S. (1995). Dervish: An office-navigation robot. *AI Magazine, Summer*, 53–60.

Olawsky, D., & Gini, M. (1990). Deferred planning and sensor use. In *Innovative Approaches to Scheduling and Control: Proceedings of 1990 DARPA Workshop*, pp. 166–174.

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.

Penberthy, J. S., & Weld, D. (1992). UCPOP: A sound, complete, partial order planning for ADL. In *Proceedings of the Third International Conference on Knowledge Representation and Reasoning*, pp. 103–114.

Peot, M., & Smith, D. E. (1992). Conditional nonlinear planning. In *Proceedings of the First International Conference on AI Planning Systems (AIPS)*, pp. 189–197.

Peshkin, L., Meuleau, N., & Kaelbling, L. (1999). Learning policies with external memory. In *Machine Learning: Proceedings of the 1999 International Conference (ICML)*, pp. 307–314.

Pickett, M., & Barto, A. G. (2002). PolicyBlocks: An algorithm for creating useful macro-actions in reinforcement learning. In *Machine Learning: Proceedings of the 2002 International Conference (ICML)*, pp. 506–513.

Pineau, J., Gordon, G., & Thrun, S. (2003a). Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1025–1032.

Pineau, J., Montermerlo, M., Pollack, M., Roy, N., & Thrun, S. (2003b). Towards robotic assistants in nursing homes: challenges and results. *Robotics and Autonomous Systems, 42*(3-4), 271–281.

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.

Poon, K.-M. (2001). A fast heuristic algorithm for decision-theoretic planning. Master's thesis, The Hong-Kong University of Science and Technology.

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.

Poupart, P., & Boutilier, C. (2003). Value-directed compression of POMDPs. In *Advances in Neural Information Processing Systems (NIPS)*, Vol. 15.

Poupart, P., & Boutilier, C. (2004). Bounded finite state controllers. In *Advances in Neural Information Processing Systems (NIPS)*, Vol. 16.

Pryor, L., & Collins, G. (1996). Planning for contingencies: A decision-based approach. *Journal of Artificial Intelligence Research*, *4*, 287–339.

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.

Rosencrantz, M., Gordon, G., & Thrun, S. (2003). Locating moving entities in dynamic indoor environments with teams of mobile robots. In *Second International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pp. 233–240.

Rosencrantz, M., Gordon, G., & Thrun, S. (2004). Learning low dimensional predictive representations. In *Machine Learning: Proceedings of the 2004 International Conference (ICML)*.

Roy, N. (2003). *Finding approximate POMDP solutions through belief compression*. Ph.D. thesis, Carnegie Mellon University.

Roy, N., & Gordon, G. (2003). Exponential family PCA for belief compression in POMDPs. In *Advances in Neural Information Processing Systems (NIPS)*, Vol. 15, pp. 1043–1049.

Roy, N., Pineau, J., & Thrun, S. (2000). Spoken dialog management using probabilistic reasoning. 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). Motion planning through policy search. In *Proceedings of*

the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 2419–2424.

Russell, S., & Norvig, P. (2002). *Artificial Intelligence: A Modern Approach (2nd edition)*. 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)*, pp. 522–529.

Sacerdoti, E. D. (1974). Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, *5*(2), 115–135.

Simmons, R., & Koenig, S. (1995). Probabilistic navigation in partially observable environments. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1080–1087.

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.

Singh, S., Littman, M. L., Jong, N. K., Pardoe, D., & Stone, P. (2003). Learning predictive state representations. In *Machine Learning: Proceedings of the 2003 International Conference (ICML)*, pp. 712–719.

Smith, D. E., & Weld, D. S. (1998). Conformant Graphplan. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI)*, pp. 889–896.

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

Sondik, E. J. (1978). The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. *Operations Research*, *23*(2), 282–304.

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.

Tate, A. (1975). *Using goal structure to direct search in a problem solver*. Ph.D. thesis, University of Edinburgh.

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., Fox, D., Burgard, W., & Dellaert, F. (2000). Robust Monte Carlo localization for

mobile robots. *Artificial Intelligence*, 99–141.

Thrun, S., & Schwartz, A. (1995). Finding structure in reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, Vol. 7, pp. 385–392.

Uhlmann, J. K. (1991). Satisfying general proximity / similarity queries with metric trees. *Information Processing Letters*, *40*, 175–179.

Vlassis, N., & Spaan, M. T. J. (2004). A fast point-based algorithm for POMDPs. In *Proceedings of the Belgian-Dutch Conference on Machine Learning*.

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.

Warren, D. H. (1976). Generating conditional plans and programs. In *Proceedings of the AISB Summer Conference*, pp. 344–354.

Weld, D. S. (1999). Recent advances in AI planning. *AI Magazine*, *20*(2), 93–123.

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.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*, 229–256.

Zhang, N. L., & Liu, W. (1996). Planning in stochastic domains: Problem characteristics and approxiimation. Tech. rep. HKUST-CS96-31, Dept. of Computer Science, Hong Kong University of Science and Technology.

Zhang, N. L., & Zhang, W. (2001). Speeding up the convergence of value iteration in partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, *14*, 29–51.

Zhou, R., & Hansen, E. A. (2001). An improved grid-based approximation algorithm for POMDPs. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 707–716.

ROBOTICS INSTITUTE, CARNEGIE MELLON UNIVERSITY, 5000 FORBES AVE., PITTSBURGH, PA 15213,
E-mail address: jpineau@cs.cmu.edu

Typeset by $\mathcal{A}_{\mathcal{M}}\mathcal{S}$-LaTeX