Probabilistic Planning for Robotic Exploration

Trey Smith

CMU-RI-TR-07-26

Submitted in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy in Robotics

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

July 2007

Thesis Committee:
Reid Simmons, Chair
Geoffrey Gordon
David Wettergreen
Leslie Pack Kaelbling (Massachusetts Institute of Technology)

ABSTRACT

Robotic exploration tasks involve inherent uncertainty. They typically include navigating through unknown terrain, searching for features that may or may not be present, and intelligently reacting to data from noisy sensors (for example, a search and rescue robot, believing it has detected a trapped earthquake victim, might stop to check for signs of life). Exploration domains are distinguished both by the prevalence of uncertainty and by the importance of intelligent information gathering. An exploring robot must understand what unknown information is most relevant to its goals, how to gather that information, and how to incorporate the results into its future actions.

This thesis has two main components. First, we present planning algorithms that generate robot control policies for partially observable Markov decision process (POMDP) planning problems. POMDP models explicitly represent the uncertain state of the world using a probability distribution over possible states, and they allow the planner to reason about information gathering actions in a way that is decision theoretically optimal. Relative to existing POMDP planning algorithms, our algorithms can more quickly generate approximately optimal policies, taking advantage of innovations in efficient value function representation, heuristic search, and state abstraction. This improved POMDP planning is important both to exploration domains and to a wider class of decision problems.

Second, we demonstrate the relevance of onboard science data analysis and POMDP planning to robotic exploration. Our experiments centered around a robot deployed to map the distribution of life in the Atacama Desert of Chile, using operational techniques similar to a Mars mission. We found that science autonomy and POMDP planning techniques significantly improved science yield for exploration tasks conducted both in simulation and onboard the robot.

ACKNOWLEDGMENTS

Thanks to my advisor, Reid Simmons, for many years of red ink. It's true, it does make you stronger. Thanks to Dave Wettergreen, who as head of the Science on the Fly project has been a co-advisor in all but name for the past few years. Thanks to Geoff Gordon for many helpful discussions, and to Leslie Kaelbling for being so supportive.

Thanks to everyone on the Life in the Atacama and Science on the Fly projects who made the robotic exploration component of this work possible—there are too many to name everyone. Thanks to Nathalie Cabrol, Kim Warren-Rhodes, Jim Dohm, Jen Piatek, Andy Hock, and everyone on the science team for believing in us. Thanks to Dom Jonak, Chris Urmson, Stuart Heys, Chris Williams, Jim Teza, Mike Wagner, Allan Lüders, Francisco Calderón, Vijay Baskaran, Dave Pane, Greg Fisher, Alan Waggoner, and everyone on the engineering team for going the distance. Thanks to Shmuel Weinstein and the Giant Eagle meat department. Thanks most especially to Dave Thompson, my partner in crime for science autonomy.

Thanks to Nick Roy, Joelle Pineau, Sebastian Thrun, Carlos Guestrin, Craig Boutilier, Michael Littman, Pascal Poupart, Matthijs Spaan, Brendan McMahan, and Guy Shani for interesting POMDP discussions. Thanks to Tony Cassandra for freely distributing his POMDP software and models.

Thanks to Suzanne Lyons Muth, Jean Harpley, and Sanae Minick for administrative support that went above and beyond. Thanks to Gaurav Veda for helping out with last-minute edits.

Thanks to many good friends and colleagues for their support over the years, especially Vandi Verma, Brennan Sellner, Mark Maimone, Dave Hershberger, Kiri Wagstaff, Arne Suppe, Paul Tompkins, Matt Deans, Liam Pedersen, Rich Washington, Illah Nourbakhsh, Andrew Moore, Red Whittaker, Mark Stehlik, and all the Generalists and Salonniers.

Finally, thanks to my family for always being there, and to Sabrina for understanding.

DEDICATION

For my father, the towering presence.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Robotic exploration is an important emerging technology in applications that range from planetary science to urban search and rescue to oceanography (Squyres et al., 2004; Casper and Murphy, 2003; Leonard et al., 2007). In all of these domains, communication difficulties can make it impractical to directly supervise robots as they explore.

Planetary surface exploration in particular faces new autonomy challenges as robot hardware becomes more capable. Future Mars rovers will be able to navigate multiple kilometers in a single command cycle, likely passing several features worthy of detailed study during each traverse. In order to take full advantage of these opportunities to increase science yield, we would like the robot to plan its information gathering intelligently. That is, it should follow a path likely to pass by interesting features, focus preliminary scanning on likely areas, and perform followup sampling when it detects something interesting (Wagner et al., 2001; Castaño et al., 2003a).

In robotic exploration, uncertainty about the state of the world provides a constant challenge to planning. A good exploration policy should:

1. Achieve baseline performance in both nominal and off-nominal cases. It is not enough to generate an action sequence that achieves exploration goals when execution proceeds as expected. The planner must reason about unexpected outcomes and ensure that there is a fallback plan.

2. Balance cost against reward. It is important to model the relative probabilities of different outcomes to achieve good average-case performance.

3. Gather information as needed. Balancing information-gathering against other actions requires the planner to reason not just about future world states, but

also about how the robot's future beliefs will affect its actions. This makes planning much more difficult.

The focus of this work is to develop probabilistic planning techniques that address these issues and significantly improve the efficiency of robotic exploration.

The first challenge in planning under uncertainty is choosing an appropriate planning formulation. Many approaches have been proposed, embodying somewhat different modeling assumptions. Examples include contingent planning with STRIPS-like operators (Draper et al., 1994), assumptive planning (Stentz, 1994), and various heuristic or greedy approaches (Pedersen, 2000; Roy and Thrun, 1999).

Our work formulates exploration planning problems using partially observable Markov decision process (POMDP) models. The POMDP framework is popular because it cleanly models probabilistic state transitions and noisy sensing, and provides a decision-theoretically optimal metric for evaluating policies that include information-gathering actions (Kaelbling et al., 1998).

Tractability is an important concern in POMDP planning. A POMDP policy must specify the robot's next action for every possible history of actions and observations so far. Unfortunately, the number of such histories is exponential in the length of the planning horizon ("the curse of history"). Alternatively, history can be summarized using the probability distribution over possible states as a sufficient statistic. But there are an infinite number of such distributions, drawn from a continuous space with dimensionality that scales with the number of states ("the curse of dimensionality"). Either way, optimal POMDP planning is intractable (Littman, 1996).

Faced with these complexity results, the POMDP community has turned its attention to efficient approximate planning, resulting in a wide range of algorithms (Aberdeen, 2003; Poupart and Boutilier, 2004). We focus on POMDP value iteration, which works by approximating the optimal value function through relaxation of the Bellman equation (Sondik, 1971; Cassandra et al., 1997). Our work relates to a subclass of value iteration algorithms that focus their effort using "point-based" updates that improve the value function in the neighborhood of a specific point (Pineau et al., 2006; Spaan and Vlassis, 2005). We improve the performance of these algorithms through innovations in the areas of value function representation, heuristic search, and state abstraction. Our techniques can provide approximate solutions with strong regret bounds for some POMDPs that are orders of magnitude too large to be tractable with previous approaches.

1.1 Thesis Statement

Thesis: *Probabilistic planning that reasons about information-gathering actions*

can enable more effective robotic exploration.

We support this statement by presenting algorithms that make large-scale POMDP planning more tractable. We applied these algorithms to robotic exploration problems; policies generated using POMDP planning were shown to be significantly more efficient than manually generated policies both in simulation and onboard a robot.

1.2 Document Outline

- Chapter 2 provides background on MDP and POMDP planning.
- Chapter 3 presents focused value iteration, a unifying framework for understanding a variety of MDP and POMDP value iteration algorithms. The framework cleanly separates the key issues of value function representation and heuristic search for selecting points to update.
- Chapter 4 presents efficiency improvements for several POMDP value function representations. Techniques include α vector masking, passive bounded pruning, and hybrid tabular representations.
- Chapter 5 is a theoretical analysis of the max-planes representation for POMDP value functions. We improve previous bounds on the number of α vectors needed to approximate the optimal value function V^* within ϵ.
- Chapter 6 presents the heuristic search algorithms HSVI and FRTDP, which use lower and upper bounds on the optimal value function to guide both action selection and outcome selection during search. Both algorithms provide significantly improved experimental performance on benchmark problems.
- Chapter 7 presents the CIVA algorithm for compressing factored POMDPs by temporarily abstracting away state variables in contexts where they are provably irrelevant. With appropriate problem structure, CIVA produces an abstract model that when flattened is exponentially smaller than the flattened version of the original model.
- Chapter 8 presents results of two experiments in robotic exploration. The first experiment, conducted in the Atacama Desert of Chile, showed that enabling a science rover to autonomously react to signs of life can significantly improve its exploration efficiency. The second experiment, conducted in a controlled outdoor environment, showed that by using advanced POMDP

planning techniques we can tractably generate exploration policies that outperform simple manually generated policies.

- Chapter 9 presents our conclusions.

The following sections describe the main results of Chapters 3-8 in more detail.

1.3 Focused Value Iteration (Chapter 3)

Value iteration approaches for MDPs and POMDPs generate policies by first approximating the optimal value function V^*. We introduce a theoretical framework called "focused value iteration" that encompasses a range of value iteration algorithms that keep two-sided lower and upper bounds on V^*. These algorithms start with weak initial bounds and gradually improve the bounds by making a series of point-based updates.

The focused value iteration framework cleanly separates the key issues of value function representation and heuristic search for selecting points to update. This separation makes it obvious that heuristic search algorithms for fully observable MDPs, such as RTDP and FRTDP, can be applied to POMDPs provided an appropriate value function representation is chosen (Geffner and Bonet, 1998).

The correctness of focused value iteration is based on the uniform improvability of the bounds (Zhang and Zhang, 2001). We define the notion of a "conservative incremental representation", which is a value function representation and associated rules for initialization and point-based updates that together ensure the value function is always uniformly improvable. We prove that when focused value iteration uses a conservative incremental representation: (1) individual point-based updates can not make the bounds looser, and (2) the output policy satisfies a key correctness property that bounds the regret. These properties apply across both MDPs and POMDPs and are independent of how the heuristic search is conducted.

1.4 POMDP Value Function Representation (Chapter 4)

V^* is usually defined as a function mapping beliefs to values, where beliefs are probability distributions over possible states. A naive tabular representation of V^* would not be finite, since V^* is defined over a high-dimensional continuous space.

However, V^* is convex, which means it can be approximated arbitrarily well with a number of finite representations. For example, the max-planes representation approximates V^* using the maximum of a set of hyperplanes, also called α

vectors (Sondik, 1971),[1] and the sawtooth representation uses an efficient approximate projection onto the convex hull of a set of belief/value pairs (Hauskrecht, 2000).

We show that with appropriate initialization and update rules, the max-planes representation is a conservative incremental representation for the lower bound and the exact convex hull and sawtooth representations are conservative for the upper bound. (Neither representation is conservative for both bounds.)

We present an improvement to max-planes called "α vector masking" that leverages sparsity by limiting the applicability of each α vector to a subspace of the belief simplex. With appropriate problem structure, as problem sparsity increases, the masking provides linear reduction in storage space and quadratic reduction in update time. A similar technique is useful for the sawtooth representation, but with weaker improvements.

We present a novel method for pruning dominated α vectors called "passive bounded pruning". Use of passive pruning reduces storage requirements nearly as effectively as the bounded pruning used by Pineau et al. (2006), but has much lower overhead because it reuses calculations performed during point-based updates. Experiments show that relative to the pairwise pruning technique, passive bounded pruning has similar running time but significantly improves space performance.

We also present hybrid tabular representations, in which an underlying representation (for example, max-planes or sawtooth) is combined with a tabular representation that stores values only for a specific finite set of beliefs. The hybrid tabular representation is conservative if the underlying representation is conservative. In some cases the hybrid tabular representation provides better overall planning time performance than the underlying representation alone, though not as much as masking does.

1.5 Max-Planes Approximation Bounds (Chapter 5)

Many existing algorithms use the max-planes representation of the POMDP optimal value function V^*, so it is natural to ask how many α vectors are required to approximate V^* within ϵ.

Pineau et al. (2006) developed one well-known bound. A finite set $\tilde{\mathcal{B}}$ is said to cover the belief simplex $\mathcal{B}$ with sample spacing δ if every belief in $\mathcal{B}$ is within distance δ of a point in $\tilde{\mathcal{B}}$. Pineau et al. selected such a set and developed an approximation to V^* that associated one α vector with each point in $\tilde{\mathcal{B}}$. They

[1] We are interested in approximations because the number of α vectors needed to represent V^* exactly is often intractable (for finite-horizon problems) or infinite (for infinite-horizon problems).

showed that the max-norm of the approximation error was linear in the 1-norm sample spacing.

We present two novel bounds along the same lines. The first generalizes the result of Pineau et al. to include a certain type of non-uniform sample spacing. We showed that for discounted POMDPs, tight spacing is needed only in the neighborhood of beliefs that can be reached from the initial belief b_0 within a few time steps. Specifically, we show that a weighted max-norm of the approximation error is linear in the sample spacing according to a weighted 1-norm. In turn, this implies that executing a one-step lookahead policy based on the approximate value function has bounded regret with respect to optimal policies.

The second bound relies on a transformation of the belief simplex that spreads out the curvature of V^*. We use uniform sampling in the transformed space, which corresponds to concentrating samples in high-curvature areas of the original space. We show that the max-norm of the approximation error is *quadratic* in the 2-norm sample spacing. Roughly speaking, this reduces the number of α vectors needed to achieve a given approximation error from n to $\sqrt{n}$.

1.6 Heuristic Search (Chapter 6)

In many POMDPs, the vast majority of the belief space is irrelevant to reasoning about good policies. Some beliefs are unreachable, others are reachable only if provably sub-optimal actions are selected, and yet others are extremely unlikely to be reached, so they can be neglected if one is willing to accept some approximation error.

POMDP heuristic search algorithms leverage this structure by focusing value function updates on the most relevant parts of the belief space. Some of these algorithms build a set $\tilde{\mathcal{B}}$ of relevant beliefs and rely on batch updates that update either all of $\tilde{\mathcal{B}}$ (Pineau et al., 2006) or a random subset of $\tilde{\mathcal{B}}$ (Spaan and Vlassis, 2005). Others, like RTDP, interleave relevant belief selection and updates asynchronously (Barto et al., 1995).[2]

We present two novel heuristic search algorithms, Heuristic Search Value Iteration (HSVI) and Focused Real-Time Dynamic Programming (FRTDP). HSVI can be viewed as a variant of RTDP that gains important benefits from fitting into the focused value iteration framework. Whereas RTDP maintains only an upper bound, focused value iteration algorithms maintain two-sided bounds on V^*, providing a way to monitor progress so the search can be halted when a desired regret

[2]RTDP is best known as a search strategy for fully observable MDPs, but with an appropriate value function representation it can be applied to POMDPs, as Geffner and Bonet (1998) showed with their RTDP-BEL algorithm.

bound is reached. HSVI also uses the two-sided bounds to direct search more effectively, preferring to visit the states that contribute most to the uncertainty at the initial state s_0. By the same token, updating those states has the greatest potential to reduce the regret bound.

FRTDP, like HSVI, prefers to visit states that contribute to uncertainty at s_0. However, FRTDP is less myopic. It uses cached priority information to avoid fruitlessly revisiting states that resist improvement. FRTDP also has an adaptive maximum depth (AMD) termination criterion to abort trials that run too long. The maximum depth increases adaptively from trial to trial as the search progresses. This technique, proposed by Barto et al. (1995) but never implemented, helps to cut off fruitless trials in "one-way door" problems, where irreversible early decisions can make it much more difficult to reach a goal.

Within the modular focused value iteration framework, both HSVI and FRTDP can be applied to both MDPs and POMDPs, but the two algorithms have different convergence properties. We show that a weakened form of FRTDP called FRTDP-AMD is guaranteed to converge over a class of finite "RTDP-solvable" MDPs. HSVI is guaranteed to converge when applied to discounted finite-branching MDPs, a class that includes discounted POMDPs represented as belief-MDPs. We have not been able to show that either algorithm converges over both problem classes. We have found simple RTDP-solvable MDPs that cause HSVI to enter an infinite loop. FRTDP appears to be robust over both classes in practice.

We compared the experimental performance of HSVI and FRTDP to RTDP and several state-of-the-art search strategies. The benchmark problems included four RTDP-solvable MDPs from the racetrack domain of Barto et al., along with three large POMDPs. Compared to the other tested strategies, FRTDP requires the least running time to achieve a specified regret bound across all of the racetrack problems and two of the three POMDPs. HSVI failed to converge over most of the racetrack problems and came in second to FRTDP over the POMDPs. Relative to the competing approaches, both algorithms provided more than 10x speedup on some problems.

1.7 State Abstraction (Chapter 7)

The state of a POMDP can often be factored into a tuple of n state variables. The corresponding unfactored or "flat" model, with size exponential in n, may be intractably large. This issue is important because most existing POMDP solvers operate on a flat model representation, with only a few exceptions (Hansen and Feng, 2000; Poupart and Boutilier, 2004).

However, in some problems there are efficient ways to identify irrelevant vari-

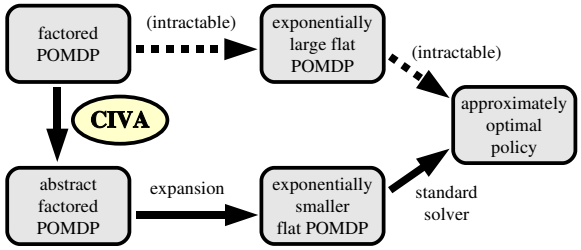


Figure 1.1: CIVA Process Diagram.

ables that cannot affect the solution. In that case, the irrelevant variables can be abstracted away, exponentially shrinking the state space in the flat model (Boutilier and Dearden, 1994). If the overall task can be hierarchically decomposed into subtasks, one can take a finer-grained approach and temporarily abstract away variables that are relevant overall but irrelevant within a particular subtask (Pineau et al., 2003c). When interleaving planning and execution, the amount of abstraction may also vary at different planning horizons (Baum and Nicholson, 1998).

We developed an alternative method called "conditionally irrelevant variable abstraction" (CIVA) for losslessly reducing the size of the factored model. A state variable is said to be *conditionally irrelevant* for a given partial assignment to other state variables if certain conditions are satisfied that guarantee it can be temporarily abstracted away without affecting policy optimality. CIVA considers only factored state, although factored actions and observations can also be useful (Guestrin et al., 2001; Feng and Hansen, 2001). Figure 1.1 shows how CIVA fits into the overall planning process. After CIVA is applied to reduce the factored model, it is flattened to unfactored form and then passed to a POMDP solver.

We applied CIVA to previously intractable POMDPs from a robotic exploration domain. We were able to compress the size of the unfactored state space from more than 10^{24} states to less than 10^4, putting the compressed POMDP within reach of our focused value iteration techniques.

1.8 Science Autonomy (Chapter 8)

"Science autonomy" refers to exploration robotics technologies involving onboard science analysis of collected data (Castaño et al., 2003b). We demonstrated the relevance of science autonomy and POMDP planning to robotic exploration in the context of the Limits of Life in the Atacama project (Figure 1.2), a three-year effort to study techniques for robotic exploration and map the distribution of extremophile life in the Atacama Desert of Chile (Cabrol et al., 2007). In order to

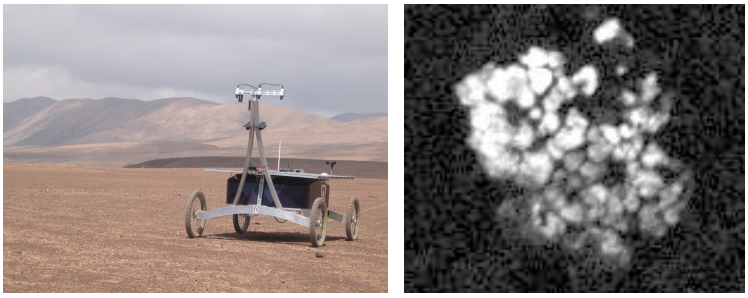


Figure 1.2: (left) The Zoë rover in the Atacama Desert of Chile, (right) A lichen detected by Zoë's fluorescence imager.

simulate Mars-like operational constraints, during Atacama field operations a team of geologists and biologists commanded the Zoë rover once per day from Pittsburgh (Wettergreen et al., 2005).

We performed two main science autonomy experiments. While operating in the Atacama, we enabled Zoë to react to preliminary signs of life by taking more detailed followup measurements, and observed that this simple change improved efficiency. This experiment provided us with valuable experience integrating science autonomy technology with onboard software systems and with the command cycle of remote science operations. It was the first demonstration of a science rover autonomously reacting to the presence of life in the field.

Later, we designed a planning domain called *LifeSurvey* based on the science goals of the Atacama expedition. In the *LifeSurvey* domain, the rover must plan a path to maximize confirmed detections of life based on prior satellite data and short-range noisy sensing. Solving the resulting POMDP models was extremely challenging, as they had up to 10^{24} states in the uncompressed flat representation.

We showed that by combining our techniques for value function representation, heuristic search, and state abstraction, we could approximately solve these POMDPs in less than 20 minutes, generating policies whose regret was bounded to within 20% of the value of the optimal policy. The POMDP policies significantly outperformed simple manually generated strategies, both in simulation and in tests onboard Zoë in a controlled outdoor environment.

1.9 Software Contributions

Most of the algorithms and POMDP and MDP models used in this work are freely available as part of the ZMDP software package, which can be downloaded from my web page at `http://www.cs.cmu.edu/~trey/zmdp/`.

ZMDP is written in C++ and runs on the Linux and Mac OS X platforms. It reads POMDP and MDP models specified in Tony Cassandra's model format, providing a number of options for solving problems and plotting anytime algorithm performance. It has been used to teach about POMDP solution algorithms in one class (CMU 16-830: Planning, Execution, and Learning), and has been downloaded more than 200 times. Please do not hesitate to try it out, and contact me if you have any problems.

1.10 Summary

We present several techniques for improving the scalability of MDP and POMDP planning, in areas ranging from value function representation to heuristic search to state abstraction. We used these techniques to solve extremely challenging POMDPs related to robotic exploration, generating approximately optimal policies with strong regret bounds. Our experiments with a rover in the Atacama Desert of Chile and in controlled outdoor environments provide early validation both for the overall concept of science autonomy and for POMDP exploration planning in particular.

Chapter 2

Probabilistic Planning Background

"Probabilistic planning" describes a set of techniques that an intelligent agent can use to choose actions in the face of uncertainty about its environment and the results of its actions. "Probabilistic" means that the techniques model uncertainty using probability theory and select actions in order to maximize expected utility. "Planning" means the techniques can reason about long-term goals that require multiple actions to accomplish, often taking advantage of feedback from the environment to reduce uncertainty and help select actions on the fly.

2.1 Deterministic Planning

Some background concepts are useful both for deterministic and probabilistic planning problems; we begin our review with a simple deterministic example problem. Consider an indoor navigation task in which a robot must navigate to a goal position at the end of a hallway without colliding with the walls or other obstacles. To start with, assume that the robot has a complete map of the hallway and that it knows at all times both its own position and the position of the goal. Furthermore, assume that to a first approximation we can model the robot's position as taking on discrete values in a map grid.

Figure 2.1 shows an example map for such a task. The planning problem is to choose a sequence of one-step motion actions that will cause the robot at the west side of the map to reach the goal at the east side without colliding with walls or obstacles. One such sequence is `east, east, north, east, east, south`. This is a classical discrete deterministic planning problem.

In general, this class of problem has world states drawn from a finite set $\mathcal{S}$ and

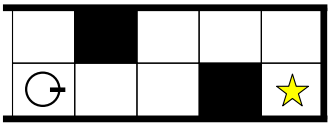


Figure 2.1: A simple indoor navigation problem.

actions available to the agent drawn from a finite set $\mathcal{A}$. State changes occur in discrete steps according to a transition function $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$. The world starts in some known state $s_0 \in \mathcal{S}$.

Under a deterministic world model it is natural to represent the agent's policy as a sequence of actions $\pi = a_0, a_1, a_2, \ldots$. The corresponding world states the agent will reach can be calculated using the transition function: $s_1 = T(s_0, a_0)$, $s_2 = T(s_1, a_1)$, etc.

Sometimes there is a prespecified maximum policy length; this is called a *finite-horizon* problem, and the maximum number of actions is called the *horizon length*, denoted h. *Infinite-horizon* problems have no prespecified maximum sequence length, in which case we write $h = \infty$. There may also be a set $\mathcal{G} \subseteq \mathcal{S}$ of absorbing *goal states*, also called *terminal states*, such that the action sequence ends when a terminal state is reached, regardless of the horizon.

There are many ways to formulate the planning problem. One might wish to find any action sequence that achieves a goal state, or the shortest such action sequence, or the lowest cost sequence (if actions have different costs). More generally, any of these preferences can be captured by specifying a utility function U that maps the sequence of actions and corresponding states to a numeric score, such that larger scores indicate preferred plans.

$$\text{preference level} = U(h, \pi, s_0) = U(s_0, a_0, s_1, a_1, \ldots, s_{h-1}, a_{h-1}) \tag{2.1}$$

Then "optimal control" means selecting a policy π that maximizes $U(h, \pi, s_0)$.

We focus on a restricted class of problems for which the utility function can be expressed as a weighted sum of immediate reward values. Immediate rewards are specified by a function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and

$$U(h, \pi, s_0) := \sum_{t=0}^{h-1} \gamma^t R(s_t, a_t). \tag{2.2}$$

where $\gamma \in (0, 1]$ is called the *discount factor*. Multiplying the reward at step t by γ^t serves to place more weight on the earlier rewards in the sequence. If

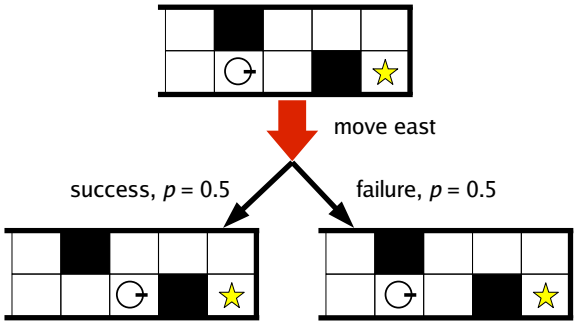


Figure 2.2: Navigation problem with uncertainty in state transitions.

$\gamma < 1$ we say the problem is *discounted*; if $\gamma = 1$ all time steps have equal weight and the problem is *undiscounted*. Discounting is sometimes motivated by real-world effects such as interest rates, but most often it is used as a convenient way of ensuring that the reward sum converges in infinite-horizon problems.

The immediate reward framework is expressive enough to capture problems that have both multiple independent goals and costs associated with each action (expressed as negative rewards).

2.2 Uncertainty in State Transitions

A robot's actions frequently have unexpected outcomes. For example, a robot attempting to move using dead reckoning will often suffer from position errors due to slippage. We can add an (exaggerated) version of this error to the original navigation domain by assuming that each motion action has only a 50% chance of achieving its desired effect, and leaves the robot in the same cell the other 50% of the time. If the robot's move is blocked by a wall or other obstacle, it fails 100% of the time.

Figure 2.2 shows this kind of uncertainty graphically. The result of applying the `east` action can no longer be predicted with certainty; instead, there are two possible outcomes with associated probabilities. However, for the time being we assume that after the action is completed the robot learns with certainty which outcome actually occurred. (Perhaps it periodically receives accurate position information from radio beacons in the hallway.)

In general, this model requires a new type of transition function $T : \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\mathcal{S})$ that maps a state/action pair to a probability distribution of possible next states. $T(s, a, s')$ denotes the probability of transitioning from s to s' when action a is applied. Note that when T is specified in this form the system is *Markovian*,

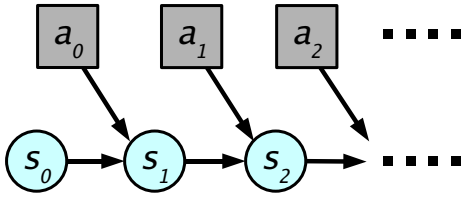


Figure 2.3: Graphical model for an MDP.

meaning that its future behavior is conditionally independent of the history of states and actions given the current state s_t. For this reason the model is called a *Markov decision process* (MDP).

Figure 2.3 shows the graphical model for an MDP. Square nodes in a graphical model represent variables that are controlled by the agent. Circular nodes represent uncontrolled dependent variables. Directed edges in the graph are used to indicate dependence relationships between variables. Note the chain structure of the MDP graphical model, which derives from the Markovian structure.

According to the decision-theoretic definition of rationality, in the presence of uncertainty the agent should choose the policy that maximizes *expected* utility

$$\text{preference level} = E[U(h, \pi, s_0)] = E_{\pi,s_0}\left[\sum_{t=0}^{h-1} \gamma^t R(s_t, a_t)\right]. \tag{2.3}$$

2.3 Policies and Value Functions

Optimal MDP policies can no longer in general be represented as a sequence of actions. A robust policy needs to take into account the possibility of unexpected action outcomes and react appropriately.

Any deterministic policy can be expressed as a function mapping the agent's available information to an action

$$a_t = \pi(h, s_0, a_0, s_1, a_1, \ldots, s_t). \tag{2.4}$$

But it turns out that, because of the Markovian structure, the agent gains no advantage from conditioning on history—it can restrict its reasoning to policies that condition only on the remaining number of steps to go and the current state

$$a_t = \pi(h, s_t). \tag{2.5}$$

In infinite-horizon problems the remaining number of steps to go is always the

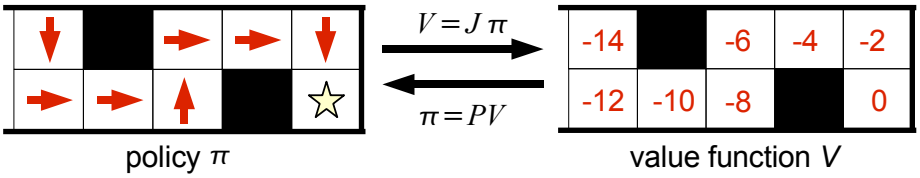


Figure 2.4: Policy and corresponding value function for the navigation problem.

same (infinite) and the form of the policy simplifies to

$$a_t = \pi(s_t). \tag{2.6}$$

This form is called a *flat policy* or *stationary policy* or *universal plan* because it specifies an action to take for any state, without any additional structure or time dependence.[1]

Figure 2.4 (left) graphically represents an optimal policy π for the MDP form of the navigation problem. Each non-goal cell of the map contains an arrow indicating the action the robot should take in that state (that is, the direction it should move).

Figure 2.4 (right) shows the expected utility of different starting states when executing the optimal policy on the left, under the assumption that the robot incurs a cost of -1 for every move and the problem is undiscounted. Because every move has a 50% chance of failure, the robot requires two moves on average to progress one cell forward. Thus the expected total cost to the goal from any given cell works out to be twice its distance from the goal.

The left side of Figure 2.4 represents a policy π that maps states to actions, and the right side represents the corresponding *value function* V that maps states to expected utility values when executing π. We can formalize this relationship in general by defining an operator J_h that maps any policy π to the corresponding horizon h value function, such that if $V = J_h\pi$,

$$V(s) = J_h\pi(s) := E[U(h, \pi, s_0 = s)]. \tag{2.7}$$

In infinite horizon problems, we customarily omit the subscripts h and write $V = J\pi$.

[1]The exponential form of discount weights in (2.2) ensures that the resulting infinite-horizon MDPs are *stationary*, meaning that the formula for calculating future rewards is the same at any time step up to multiplication by the factor γ^t, which is independent of future actions. This implies that the same policies are optimal at any time step. Other formulations of the utility function may result in non-stationary infinite-horizon MDPs; we do not consider that case.

There is also a natural way to turn a value function V back into a corresponding policy. Suppose the robot is in a cell s. Any action it takes will cause it to transition with probability 50% to one of the neighbors of s. But since V specifies the value of each neighbor, it is easy to see that the robot should try to move to the neighboring cell with the best (least negative) value. This method for calculating the policy is called *one-step lookahead*.

For arbitrary MDPs, the appropriate generalization of this idea is the one-step lookahead operator P, which is defined to map any value function V to a policy π, such that if $\pi = PV$,

$$\pi(s) = PV(s) := \arg\max_a \left[R(s,a) + \gamma \sum_{s'} T(s,a,s')V(s') \right]. \tag{2.8}$$

The expected reward from taking an action breaks up into the immediate reward and the expected long-term reward from possible next states, weighted by their likelihood. The agent should execute the action that maximizes this expected reward.

Analogous to (2.8), the *Bellman update* operator H is defined to be a self map over the space of value functions such that if $V' = HV$,

$$V'(s) = HV(s) := \max_a \left[R(s,a) + \gamma \sum_{s'} T(s,a,s')V(s') \right]. \tag{2.9}$$

Intuitively, the Bellman update corrects the value of each state so that it is consistent with the values of its possible successor states assuming the best action is chosen. Bellman (1957) introduced the Bellman update and proved several results that are fundamental to MDP planning:

1. Let $\mathcal{P}$ be the set of all policies. There is at least one policy $\pi^* \in \mathcal{P}$ which globally dominates all other policies in the sense that for any finite or infinite horizon h and $\pi \in \mathcal{P}$, $J_h\pi^* \geq J_h\pi$.[2]

2. Let π^* be an optimal policy. At any finite or infinite horizon h, there is a (unique) optimal value function $V_h^* := J_h\pi^*$. There may be several optimal policies; at any given horizon h they all share the same optimal value function V_h^*. When $h = \infty$ we customarily omit the subscripts h and write $V^* = J\pi^*$.

3. An optimal policy can be recovered from the optimal value function using

[2] Our convention is that equalities and inequalities between functions are understood to be pointwise. For example, $V \leq V'$ means that $V(s) \leq V'(s)$ for all s in the domain of V.

the one-step lookahead operator P according to $\pi^*(h, s) = PV^*_{h-1}$. In the infinite-horizon case we customarily omit h and write $\pi^*(s) = PV^*$. If there are multiple optimal policies, they correspond to different tie-breaking conventions for the $\arg\max$ operator in (2.8).

4. The Bellman update H satisfies the recurrence $V^*_h = HV^*_{h-1}$. Thus, together with the identity $V^*_0 = 0$, it can be used to generate the optimal value function at any finite horizon h.

5. In discounted infinite-horizon problems, the sequence $V^*_0, V^*_1, V^*_2, \ldots$ uniformly converges to V^*_∞. Thus by applying H enough times the optimal value function V^*_∞ can be approximated to any desired degree of accuracy.

2.4 Value Iteration

Value iteration is an algorithm that uses repeated application of the Bellman update to solve an MDP. In finite-horizon problems it generates the V^*_h functions for a range of finite values of h and selects optimal run-time actions using one-step lookahead according to $a = \pi^*(h, s) = PV^*_{h-1}(s)$.

In infinite-horizon problems value iteration generates an approximation $V \approx V^*_\infty$ and selects near-optimal run-time actions using one-step lookahead according to $a = \pi(s) = PV(s)$. We will focus our attention on the infinite-horizon case.

The first step in implementing value iteration is to choose a data structure for representing the value functions V^*_h and an algorithm that calculates the Bellman update for that representation. Algorithm 2.1 introduces the UPDATESIGNATURE type signature that specifies the types and functions that an update implementation must define. A complete implementation needs a data structure for representing the value function, a way to calculate the the value $V(s)$ for any state s, a way to generate the initial value function V^*_0, and a way to calculate the result HV of a Bellman update. As we will see later, it is also useful to have a way to calculate the residual or maximum difference between any two value functions.

Algorithm 2.2 defines TABULAR, which is a particularly straightforward update implementation conforming to UPDATESIGNATURE. The TABULAR implementation represents a value function as a finite array of real values, indexed by state. Evaluation of $V(s)$ for a state s is simply an array lookup. The Bellman update is implemented in the obvious way based on (2.9). Later we will see more interesting update implementations for problems where the TABULAR implementation is impractical.

Finally, Algorithm 2.3 defines value iteration. Note that it is implemented as a generic algorithm that can be used with any update implementation that conforms

Algorithm 2.1 UPDATESIGNATURE, a type signature for Bellman update implementations.

1: **type** $\mathcal{V}$ [the value function representation]
2: **function** evaluate: $\mathcal{V} \times \mathcal{S} \to \mathbb{R}$
3: **function** initialValueFunction: $\emptyset \to \mathcal{V}$
4: **function** globalUpdate: $\mathcal{V} \to \mathcal{V}$
5: **function** residual: $\mathcal{V} \times \mathcal{V} \to \mathbb{R}$

Algorithm 2.2 TABULAR, a Bellman update implementation.

1: **type** TABULAR.$\mathcal{V}$ = finite array of real values, indexed by state
2:
3: **function** TABULAR.evaluate(V, s) **:**
4: **return** $V(s)$
5:
6: **function** TABULAR.initialValueFunction() **:**
7: **for** $s \in \mathcal{S}$ **:**
8: $V_0^*(s) \leftarrow 0$
9: **return** V_0^*
10:
11: **function** TABULAR.globalUpdate(V) **:**
12: **for** $s \in \mathcal{S}$ **:**
13: $V'(s) \leftarrow \max_a \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s')V(s')\right]$
14: **return** V'
15:
16: **function** TABULAR.residual(V, V') **:**
17: **return** $\max_{s \in \mathcal{S}} |V(s) - V'(s)|$

to UPDATESIGNATURE. In order to instantiate value iteration, one substitutes an actual implementation wherever the generic name <UPDATER> appears.

The logic concerning the δ and ϵ variables in the value iteration algorithm serves to ensure that the resulting policy π has small regret, or expected loss relative to an optimal policy, which is defined to be

$$\text{regret}(\pi) := J\pi^*(s_0) - J\pi(s_0). \tag{2.10}$$

Williams and Baird (1993) showed how to bound the regret of value iteration for discounted infinite-horizon problems with the following result. If $V' = HV$

Algorithm 2.3 Value iteration (Bellman, 1957).

uses implementation <UPDATER> **conforming to** UPDATESIGNATURE

function valueIteration(δ) **:**
  [returns a value function V such that $\mathrm{regret}(PV) \leq \delta$]
  $V \leftarrow$ <UPDATER>.initialValueFunction()
  loop:
    $V' \leftarrow$ <UPDATER>.globalUpdate(V)
    $\epsilon \leftarrow$ <UPDATER>.residual(V, V')
    $V \leftarrow V'$
    if $2\epsilon\gamma/(1-\gamma) \leq \delta$ **:**
      return V

function chooseAction(V, s) **:**
  [used to select actions during policy execution]
  return $\arg\max_a [R(s,a) + \gamma \sum_{s'} T(s,a,s')$ <UPDATER>.evaluate(V, s')$]$

and $\epsilon = \|V - V'\|_\infty$, then [3]

$$\left\|J\pi^* - JPV'\right\|_\infty \leq \frac{2\epsilon\gamma}{1-\gamma}, \tag{2.13}$$

In other words, if the Bellman update that generated V' has a small residual ϵ, then the resulting policy PV' has small regret. Intuitively, this is because the sequence $V_0^*, V_1^*, V_2^*, \ldots$ converges to V_∞^* taking smaller steps at each iteration of H, and because near-optimal value functions result in near-optimal policies.

[3] The notation $\|\cdot\|_\infty$ refers to the $\mathcal{L}^p$ norms used to measure distance between functions. For any $p \in \mathbb{N}^+$, discrete domain X, and functions $f, f' : X \to \mathbb{R}$,

$$\left\|f - f'\right\|_p := \left[\sum_{x \in X} |f(x) - f'(x)|^p\right]^{1/p}, \tag{2.11}$$

and $p = \infty$ gives the so-called *max-norm*,

$$\left\|f - f'\right\|_\infty := \sup_{x \in X} |f(x) - f'(x)|. \tag{2.12}$$

This notation naturally extends to continuous spaces X, replacing the discrete sum with an integral.

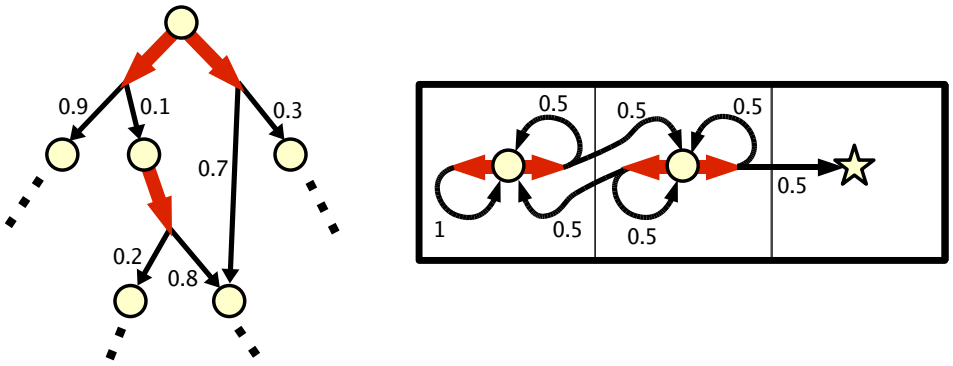


Figure 2.5: Search graphs: (left) A typical example. (right) The search graph for a simplified version of the navigation problem.

2.5 Search Graphs and Policy Graphs

The $T(s, a, s')$ transition function notation we have used so far encourages one to think of an MDP model as a table of transition probabilities. However, it is often more natural to think of the model as a search graph in which states are nodes and transitions are edges. In a typical MDP each state has only a few neighbors (that is, possible successor states) so that the resulting graph is sparse, and planning algorithms can take advantage of the sparsity.

An MDP search graph is an AND/OR graph. Each state of the MDP has a node in the graph, and each state/action transition of the MDP has a corresponding k-connector in the graph, connecting a state s to the set of possible successor states $\mathcal{N}(s, a) \subseteq \mathcal{S}$. Each outgoing edge of the k-connector to a successor state $s' \in \mathcal{N}(s, a)$ is followed with probability $T(s, a, s')$ as before.

Figure 2.5 (left) shows a typical example search graph. Nodes are states, thick arrows are k-connectors representing actions available to the agent, and thin arrows represent outgoing edges of the k-connectors. Note that there may be multiple paths through the graph that reach the same state. Figure 2.5 (right) shows the search graph for a simplified version of the navigation problem in which the robot can only move `east` or `west`. Note the presence of self-transitions and loops.

A *policy graph* is an alternative representation for a policy. Nodes in the policy graph represent points where the agent must select an action. The agent begins execution in a specified start node of the graph. At each time step it executes the action specified by the current node, then transitions to a new node based on the outcome of the action.

Figure 2.6 shows a policy graph π for the navigation problem with $h = 3$ steps

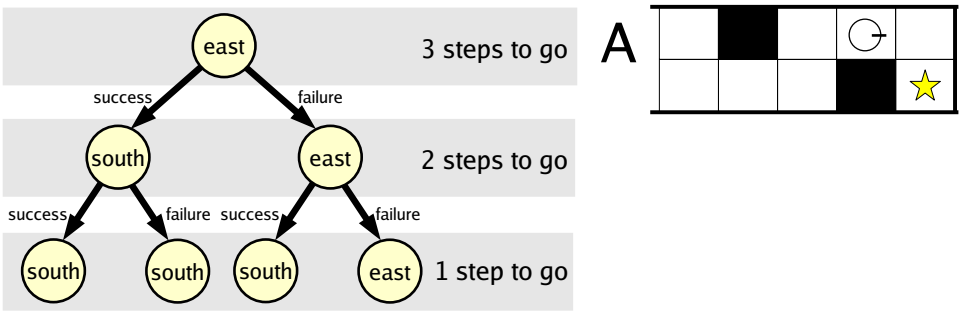


Figure 2.6: A policy tree for the navigation problem with $h = 3$ and an example state A.

to go and example state A. π is designed to be a robust policy for reaching the goal from state A. In this policy, when a move fails, the robot reacts by attempting the same move again.

A policy graph can be formally defined as follows. Let X be the set of nodes. For any node $x \in \mathcal{X}$, the action to take is denoted $A(x)$ and the children of x are denoted $C(x, i)$. If there are k possible outcomes, i ranges from 1 to k. Execution starts in node $x_0 \in \mathcal{X}$.

There is a close relationship between the search graph and the policy graph for an MDP. During execution of the policy, the dynamic system steps through states s of the search graph and the agent steps through nodes x of the policy graph in synchrony. In order for the policy graph to make sense when applying action $A(x)$ in state s, there must be a mapping from the possible outcomes $\mathcal{N}(s, a)$ in the search graph to outcomes $C(x, \cdot)$ in the policy graph. In particular, if the policy graph specifies outcomes using labels like `success` and `failure`, the MDP model needs to use the same labels.

The policy graph is often taken to be a subgraph of the search graph in which one action from each state node is selected for execution and the remaining actions are discarded. When such a subgraph relationship holds, there is a one-to-one correspondence between states s and nodes x. However, in general there need not be such a relationship. During execution the same policy node may be reached with many different world states, and vice versa.

The policy graph representation can be particularly compact for an MDP with a short finite horizon h. Any deterministic policy for the MDP can be represented as a depth h tree. With k-way branching at each step, the number of nodes at any

level t of the tree is k^t, so the total number of nodes in the tree is

$$\text{nodeCount}(k, h) = \sum_{t=0}^{h} k^t = \frac{k^{h+1} - 1}{k - 1}, \tag{2.14}$$

which is exponential in h.

Let $\mathcal{T}_h$ be the set of all policy trees of depth h. In order to specify such a tree, one must select an action for each node from $|\mathcal{A}|$ possibilities, so the number of possible trees is

$$|\mathcal{T}_h| = |\mathcal{A}|^{\text{nodeCount}(k,h)} = |\mathcal{A}|^{\frac{k^{h+1}-1}{k-1}}, \tag{2.15}$$

which is doubly exponential in h. Although these quantities grow very rapidly as h increases, *they do not depend on the size of the state set*. In contrast, specifying a flat policy representation means selecting an action for each state, so the number of possible flat policies is

$$|\mathcal{P}| = |\mathcal{A}|^{|\mathcal{S}|}. \tag{2.16}$$

which depends on $|\mathcal{S}|$ rather than nodeCount(k, h). Thus the relative ease of working with the different policy representations depends on the relative size the state set and the node count of policy trees. The policy graph representation will be particularly advantageous later when we discuss MDPs with continuous state spaces.

Although our policy graph discussion has so far focused on trees, a general graph can often represent the same policy more compactly. Nonetheless, in searching for an optimal policy, it suffices to consider only trees, since a general graph can be converted to a tree by selecting one node as the root, then "unrolling" the graph by cloning nodes that can be reached via multiple paths. Note that in infinite-horizon problems, a finite graph with loops unrolls into an infinite tree.

2.6 Partial Observability

In the last section we rather unrealistically assumed that the navigating robot received perfectly accurate position information after every time step. In this section, we relax that assumption. In particular, suppose the robot cannot tell how far along the hallway it has traveled. Instead, it has a noisy obstacle sensor that nominally returns an `obstacle` reading when the map cell to the east of the robot is blocked and a `clear` reading otherwise, but gives an incorrect reading 10% of the time.

Figure 2.7 shows the resulting uncertainty graphically. As before, the `east` action can cause a transition to either of two states, but now we have the additional

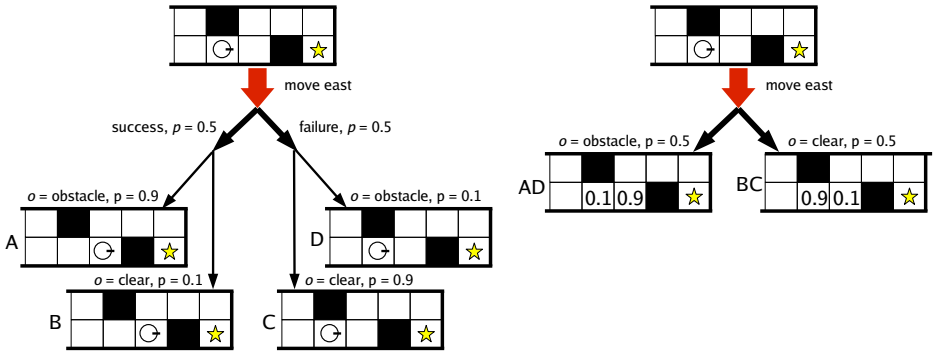


Figure 2.7: Navigation problem with uncertainty in state transitions and partial observability: (left) Fully expanded view of state transition. (right) Information available to the robot.

complication of the noisy observation. Taken together there are four possible results, labeled condition A (success, `obstacle` observation), B (success, `clear` observation), C (failure, `clear` observation), and D (failure, `obstacle` observation).

Unfortunately, because the robot receives only the observation, it cannot distinguish between conditions A and D, nor between conditions B and C. As a result, its available information is better represented by the transition diagram on the right. Conditions A and D have been combined into condition AD, and B and C have been combined into BC. Looking more closely at condition AD, we see that the robot cannot infer its position with certainty, since condition A and condition D have the robot in different positions. Instead, the likelihoods of possible positions of the rover are marked on the map; the values correspond to the relative likelihood of condition A and condition D.

This type of model is called a *partially observable Markov decision process* (POMDP). In general, a POMDP is an MDP extended to include an observation model. Rather than receiving complete state information s_t after each step of execution, in a POMDP model the agent is assumed to receive a noisy observation o_t.

POMDPs model observations probabilistically. The set of possible observations is a discrete set $\mathcal{O}$, and each observation carries information about the preceding action and current state according to the noisy observation function $O : \mathcal{A} \times \mathcal{S} \rightarrow \Pi(\mathcal{O})$, defined such that

$$O(a, s', o) := \Pr(o_{t+1} = o \mid a_t = a, s_{t+1} = s'). \tag{2.17}$$

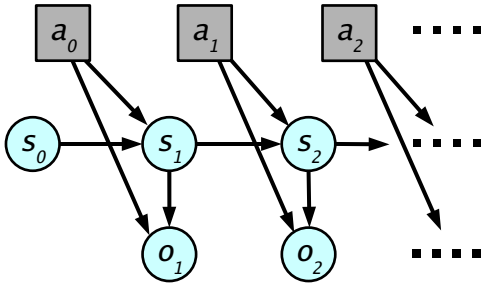


Figure 2.8: Graphical model for a POMDP.

The agent is also assumed to have a probability distribution $b_0 \in \Pi(\mathcal{S})$ describing the initial state of the system, such that $b_0(s) = \Pr(s_0 = s)$. In general, we describe a probability distribution over $\mathcal{S}$ as a *belief*, and denote the space of beliefs with $\mathcal{B} = \Pi(\mathcal{S})$. Beliefs can be thought of as length-$|\mathcal{S}|$ vectors, and because the entries of a belief vector must sum to 1, $\mathcal{B}$ is a simplex of dimension $|\mathcal{S}| - 1$ embedded in $\mathbb{R}^{|\mathcal{S}|}$.

Figure 2.8 shows the graphical model for a POMDP. The chain structure of the state sequence from the MDP is retained, but the agent no longer has direct access to the state information at each time step. Instead, it can infer only uncertain state information from the history of observations.

At each time step, the agent can use Bayesian reasoning to generate an updated belief that takes into account a prior (using the previous belief and the probabilistic transition model) and the most recent observation (using the noisy observation model). Let b^{ao} denote the agent's updated belief at the next time step after taking action a and receiving observation o. That is,

$$b^{ao}(s') := \Pr(s_{t+1} = s' \mid b_t = b, a_t = a, o_{t+1} = o). \tag{2.18}$$

The updated belief can be calculated as follows:

$$\begin{aligned} b^{ao}(s') &= \Pr(s' \mid b, a, o) && (2.19)\\ &= \frac{\Pr(s', o \mid b, a)}{\Pr(o \mid b, a)} && (2.20)\\ &= \frac{\Pr(o \mid a, s') \sum_s \Pr(s' \mid s, a) \Pr(s \mid b)}{\sum_{\bar{s}} \Pr(o \mid a, \bar{s}) \sum_s \Pr(\bar{s} \mid s, a) \Pr(s \mid b)} && (2.21)\\ &= \frac{O(a, s', o) \sum_s T(s, a, s') b(s)}{\sum_{\bar{s}} O(a, \bar{s}, o) \sum_s T(s, a, \bar{s}) b(s)} && (2.22) \end{aligned}$$

When planning in an MDP model, the agent's policy could be conditioned on a history of states and actions. In a POMDP model, the agent has knowledge only of actions and observations, so a deterministic policy with horizon h has the form

$$a_t = \pi(h, a_0, o_1, a_1, o_2, a_2, \ldots, a_{t-1}, o_t). \tag{2.23}$$

With an MDP, the history could be safely discarded given the current state because the system was Markovian. With a POMDP, the agent does not have access to the current state, but it can use the current *belief* as a sufficient statistic for the history. Thus, given the current belief, the agent gains no advantage from conditioning on history, and it can restrict its reasoning to policies in the form

$$a_t = \pi(h, b_t). \tag{2.24}$$

This change of variables suggests transforming the POMDP into a *belief MDP*, as follows:

1. The POMDP belief simplex $\mathcal{B}$ plays the role of the MDP state set.

2. The action set, horizon, and discount factor of the POMDP are used unchanged for the belief MDP.

3. The transition function $\tilde{T}$ of the belief MDP gives the probability of transitioning from one belief to another according to the observation model and belief update rule

$$\begin{aligned} \tilde{T}(b, a, b') &:= \Pr(b' \mid b, a) && (2.25)\\ &= \sum_o \Pr(b' \mid b, a, o) \Pr(o \mid b, a) && (2.26)\\ &= \sum_o \delta(b', b^{ao}) \sum_{s,s'} O(a, s', o) T(s, a, s') b(s), && (2.27) \end{aligned}$$

   where $\delta(x, y) = 1$ if $x = y$ and 0 otherwise.

4. The reward function $\tilde{R}$ gives the expected immediate reward from applying an action in a given belief

$$\begin{aligned} \tilde{R}(b, a) &:= E_{b_t=b}[R(s_t, a)] && (2.28)\\ &= \sum_s R(s, a) b(s). && (2.29) \end{aligned}$$

Note that the state set of the belief MDP is the POMDP belief simplex, which

is uncountably infinite. Although our earlier discussion of MDPs assumed that the state space was finite, the same theoretical results go through with an infinite state set. For instance, *assuming the Bellman update computations can be realized*, we can apply value iteration to the belief MDP and generate a policy with the same regret bounds presented in §2.4.

2.7 Belief MDP Value Function Structure

The value functions of belief MDP policies have a special structure that makes it possible to generalize value iteration. For any finite horizon h, the optimal value function V_h^* is piecewise-linear and convex (PWLC).

To see where this structure comes from, imagine selecting a particular policy tree π and evaluating its expected reward starting from different initial beliefs b_0. Since π is a policy tree, the actions it selects during execution are completely determined by the agent's observations. These observations and the states reached by the system in turn depend on the initial state s_0, but they are *not* explicitly conditioned on the initial belief b_0. Once the policy π and an initial state s_0 are specified, the trajectory of the system (states, actions, and observations) is a random process that is independent of b_0. As shown below, it is because of this conditional independence that the value function $J_h\pi$ is linear in the initial belief.

Formally, let π be any policy whose actions are not conditioned on the initial belief (such as a policy tree) and let α be a length-$|\mathcal{S}|$ vector such that $\alpha(s)$ is the expected value of following π starting from state s:

$$\alpha(s) = E_{\pi, s_0=s}\left[\sum_{t=0}^{h-1} \gamma^t R(s_t, a_t)\right]. \tag{2.30}$$

Then the value of executing π starting from a belief b is

$$J_h\pi(b) = E_{\pi,b_0=b}\left[\sum_{t=0}^{h-1}\gamma^t R(s_t,a_t)\right] \tag{2.31}$$

$$= E_{\pi,s_0\sim b}\left[\sum_{t=0}^{h-1}\gamma^t R(s_t,a_t)\right] \tag{2.32}$$

$$= \sum_s b(s) E_{\pi,s_0=s}\left[\sum_{t=0}^{h-1}\gamma^t R(s_t,a_t)\right] \tag{2.33}$$

$$= \sum_s b(s)\alpha(s) \tag{2.34}$$

$$= \alpha^T b, \tag{2.35}$$

where (2.32) follows from the fact that the visited states s_t and selected actions a_t are conditionally independent of b_0 given π and s_0.

Abusing notation, we also write

$$\alpha(b) := \alpha^T b, \tag{2.36}$$

When the max operator is applied to a set of α vectors, each vector is interpreted in this second sense, as a function over the belief simplex. In other words, if

$$V = \max\{\alpha_1,\ldots,\alpha_n\}, \tag{2.37}$$

then

$$V(b) := \max_{\alpha_i} \alpha_i^T b. \tag{2.38}$$

It helps to look at policy trees in the context of a concrete example problem. We use the well-known TIGER POMDP (Cassandra et al., 1994). In the TIGER problem, you stand before two doors. Behind one door is a tiger and behind the other is a pot of gold, but you do not know which is which. Thus there are two equally likely states—the tiger is located either behind the left or right door (the `tiger-left` or `tiger-right` state). You may try to learn more using the `listen` action, which provides information about which door the tiger is lurking behind, either the `noise-left` or `noise-right` observation—the observation has an 85% chance of accurately indicating the location of the tiger. Alternately, you may choose to open one of the two doors using the `open-left` or `open-right` actions, at which point the game ends and you will either happily

	$s =$`tiger-left`	$s =$`tiger-right`
$b_0(s)$	0.5	0.5
$R(s, \texttt{open-left})$	-100	10
$R(s, \texttt{open-right})$	10	-100
$R(s, \texttt{listen})$	-1	-1
$O(\texttt{listen}, s, \texttt{noise-left})$	0.85	0.15
$O(\texttt{listen}, s, \texttt{noise-right})$	0.15	0.85

Table 2.1: TIGER problem parameters

receive the gold (reward +10) or be eaten (reward -100). The parameters of the POMDP are summarized in Table 2.1; the discount factor $\gamma = 0.95$.

Figure 2.9 shows some example two-step policy trees for the TIGER POMDP and the corresponding α vectors. The belief b varies along the x axis of the plot, from 100% certainty of `tiger-left` at the extreme left to 100% certainty of `tiger-right` at the extreme right. Each line in the plot relates to one of the policy trees as follows:

- *Policy A*: You select `listen`, then select `open-right` if `noise-left` is heard (and otherwise listen again). This is a good policy in the `tiger-left` state, since the most likely course of events is that you will hear `noise-left` and then perform the `open-right` action, receiving the pot of gold. Policy A performs poorly in the `tiger-right` state—it usually causes you to receive a small penalty for listening twice, and in the worst case you get a false `noise-left` reading and are eaten by the tiger. This is reflected in the policy A value function, which is at its highest in the `tiger-left` state and slopes down as the probability of being in the `tiger-right` state increases.

- *Policy B*: This policy is symmetrical with policy A—it works best when there is a high probability of being in the `tiger-right` state.

- *Policy C*: You listen, then open whichever door you hear a noise behind. This policy is clearly a bad idea regardless of what the you believe, since it preferentially opens the door where the tiger is located! This is reflected in its uniformly poor value function.

Let $\mathcal{T}_h = \{\pi_1, \ldots, \pi_n\}$ be the set of all policy trees of depth h as before, and let $\Gamma_h = \{\alpha_1, \ldots, \alpha_n\}$ be the corresponding set of α vectors. The optimal value for

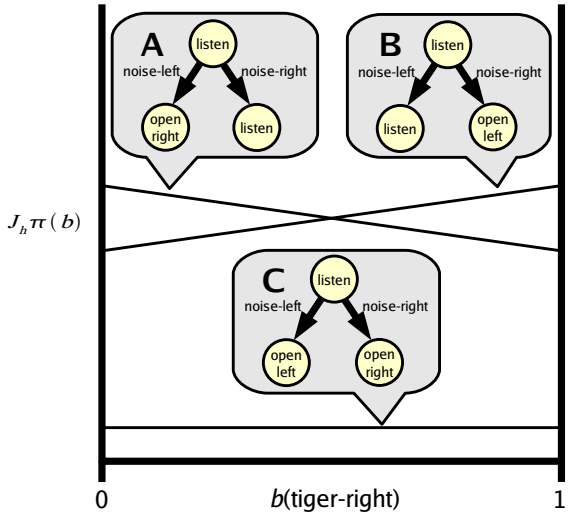


Figure 2.9: Example policy trees and corresponding α vectors for the TIGER POMDP.

any particular belief b is the largest value assigned to b by any policy tree, meaning

$$V_h^*(b) = \max_{\pi \in \mathcal{T}_h} J_h \pi(b) \tag{2.39}$$

$$= \max \Gamma_h \tag{2.40}$$

$$:= \max_{\alpha \in \Gamma_h} \alpha^T b \tag{2.41}$$

In other words, for any finite h, the function V_h^* can be written as the maximum of a finite number of linear functions. We call this the *max-planes representation*. The existence of this representation means that V_h^* is a PWLC function.

Figure 2.10 shows optimal value functions V_h^* for a typical two-state POMDP, for several values of h. For each value of h, the thin lines are individual α vectors, each corresponding to a depth h policy tree, and V_h^* is the upper surface of all these α vectors, represented with a thicker line. The number of α vectors needed to represent V_h^* tends to increase with h, and in the limit V_∞^* may contain a countably infinite number of α vectors, in which case it is still convex but no longer piecewise-linear.

Note that the value function for policy C in Figure 2.9 is everywhere dominated by policies A and B. It turns out that in a typical POMDP most policy trees from $\mathcal{T}_h$ are, like policy C, sub-optimal for every belief. This means that the corresponding α vectors are not part of the upper surface of V_h^*. These dominated α vectors can

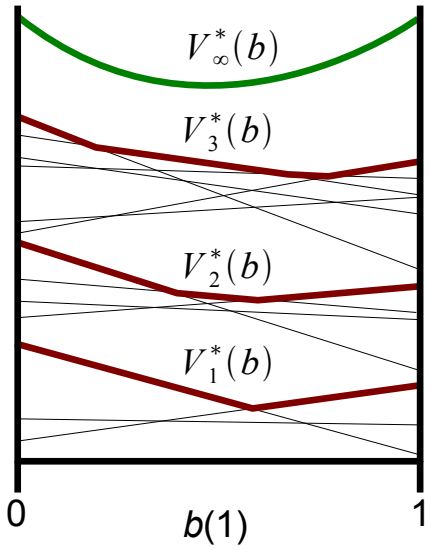


Figure 2.10: Optimal value functions at different time horizons for a typical two-state POMDP.

be pruned from the set Γ_h without affecting the value function. Pruning dominated vectors can exponentially reduce the size of the value function representation; thus it plays an important role in practical POMDP value iteration algorithms.

So far we have graphed value functions for POMDPs with just two states. POMDPs with more states have the same value function structure, but since the value function is defined over a high-dimensional space of beliefs it is naturally harder to visualize. Figure 2.11 provides some geometrical intuition by showing the upper surface V^*_h for an example three-state POMDP. The hatched area below the surface represents the domain over which V^*_h is defined.

2.8 Relating Max-Planes Structure to the Bellman Update

Since all of the V^*_h functions have max-planes structure, it should come as no surprise that the Bellman update that generates them preserves this type of structure, as formalized in the following result.[4]

[4]It is well known that the V^*_h functions produced by value iteration are PWLC. Our analysis is unusual in that it explicitly identifies certain linear structure within H and assumes only that the value function passed to H is convex. It is possible that this result is novel.

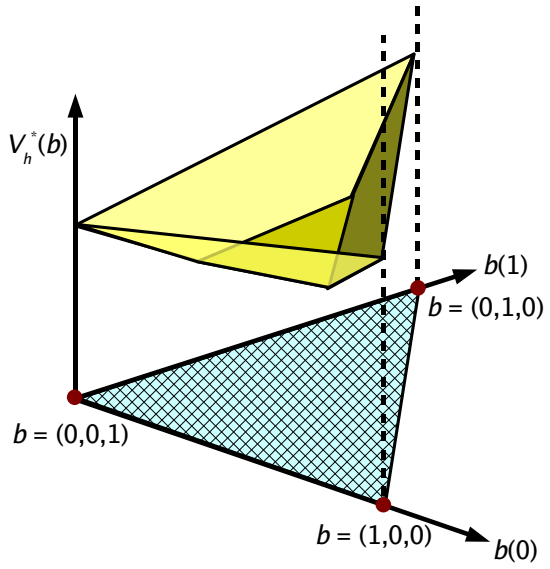


Figure 2.11: V^* for an example three-state POMDP.

Theorem 2.1. *The Bellman update H preserves the max-planes structure present in its input. Specifically, if V is convex (or PWLC) then HV is convex (or PWLC).*

Proof. Let V be a convex (PWLC) value function. From the theory of convex functions, we know that V is convex if and only if it can be represented as the supremum of a set Γ of α vectors:

$$V(b) = \sup_{\alpha \in \Gamma} \alpha^T b. \tag{2.42}$$

Recall the definition of H for belief MDPs,

$$HV(b) = \max_a \big[\tilde{R}(b,a) + \gamma \underbrace{\sum_{b'} \tilde{T}(b,a,b') V(b')}_{=: X_a(b)} \big]. \tag{2.43}$$

Some basic properties from the theory of convex functions are:

(P1) If $\mathcal{F}$ is a finite set of functions that are all convex (PWLC), then the pointwise max function $b \mapsto \max_{V \in \mathcal{F}} V(b)$ is convex (PWLC).

(P2) If $V_1, \ldots, V_n$ are all convex (PWLC), and $c_1, \ldots, c_n \geq 0$ are real scalars, then $c_1 V_1 + \cdots + c_n V_n$ is convex (PWLC).

(P3) If V is convex (PWLC) and K is linear, then the function composition $V \circ K$ is convex (PWLC).

From (2.43), applying (P1) and (P2) we see that in order to prove HV is convex (PWLC), it suffices to show that the functions $\tilde{R}(\cdot, a)$ and X_a are convex (PWLC).[5] We can immediately verify from (2.29) that $\tilde{R}(\cdot, a)$ is linear and thus PWLC. It remains only to show that X_a is convex (PWLC).

Expanding,

$$X_a(b) = \sum_{b'} \tilde{T}(b, a, b')V(b') \tag{2.44}$$

$$= \sum_{b'} \Pr(b' \mid b, a) \sup_{\alpha \in \Gamma} \alpha^T b' \tag{2.45}$$

$$= \sum_{o} \Pr(o \mid b, a) \sup_{\alpha \in \Gamma} \alpha^T b^{ao} \tag{2.46}$$

$$= \sum_{o} \Pr(o \mid b, a) \sup_{\alpha \in \Gamma} \sum_{s'} \alpha(s') b^{ao}(s') \tag{2.47}$$

$$= \sum_{o} \Pr(o \mid b, a) \sup_{\alpha \in \Gamma} \sum_{s'} \alpha(s') \frac{\Pr(s', o \mid b, a)}{\Pr(o \mid b, a)} \tag{2.48}$$

$$= \sum_{o} \sup_{\alpha \in \Gamma} \sum_{s'} \alpha(s') \Pr(s', o \mid b, a) \tag{2.49}$$

$$= \sum_{o} \sup_{\alpha \in \Gamma} \sum_{s'} \alpha(s') \sum_{s} O(a, s', o) T(s, a, s') b(s) \tag{2.50}$$

Define the linear mapping $M^{ao} : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$ such that

$$(M^{ao} b)(s') := \Pr(s', o \mid b, a) = \sum_{s} O(a, s', o) T(s, a, s') b(s). \tag{2.51}$$

[5] Here we use $\tilde{R}(\cdot, a)$ to indicate the mapping $b \mapsto \tilde{R}(b, a)$. In general, the "$\cdot$" notation is useful for constructing a single-argument function by fixing the value of all but one argument of a multiple-argument function.

Substituting (2.51) into (2.50),

$$X_a(b) = \sum_o \sup_{\alpha \in \Gamma} \sum_{s'} \alpha(s')(M^{ao}b)(s') \tag{2.52}$$

$$= \sum_o \sup_{\alpha \in \Gamma} \alpha^T M^{ao} b \tag{2.53}$$

$$= \sum_o (V \circ M^{ao})(b). \tag{2.54}$$

Thus, since V is convex (PWLC) and each mapping M^{ao} is linear, applying (P2) and (P3), we see that X_a is convex (PWLC). □

We can get further insight into the relationship between the α vectors of V and the α vectors of HV. Let $k = |\mathcal{O}|$. Combining (2.43) and (2.53),

$$HV(b) = \max_{a \in \mathcal{A}} \left[\tilde{R}(b,a) + \gamma \sum_o \sup_{\alpha \in \Gamma} \alpha^T M^{ao} b \right] \tag{2.55}$$

$$= \max_{a \in \mathcal{A}} \sup_{(\alpha_1,\ldots,\alpha_k) \in \Gamma^k} \left[\tilde{R}(b,a) + \gamma \sum_o \alpha_o^T M^{ao} b \right] \tag{2.56}$$

$$= \max_{a \in \mathcal{A}} \sup_{(\alpha_1,\ldots,\alpha_k) \in \Gamma^k} \left[\tilde{R}(b,a) + \gamma \sum_o (M^{aoT} \alpha_o)^T b \right]. \tag{2.57}$$

Define

$$\Gamma' := \left\{ \tilde{R}(\cdot,a) + \gamma \sum_o M^{aoT} \alpha_o \;\middle|\; a \in \mathcal{A}, (\alpha_1,\ldots,\alpha_k) \in \Gamma^k \right\}. \tag{2.58}$$

Combining (2.57) and (2.58), we find that

$$HV(b) = \sup_{\alpha \in \Gamma'} \alpha^T b, \tag{2.59}$$

or in other words, Γ' is a max-planes representation of HV.

Equation (2.58) shows how after a Bellman update, each α vector of the resulting max-planes representation Γ' formed from an affine combination of k transformed α vectors from Γ that have passed through the mappings M^{aoT} for varying values of o.

It turns out, in fact, that the policy evaluation operator J_h is a homomorphism from $\mathcal{T}_h$ to Γ_h, such that if $\pi \in \mathcal{T}_h$ and $\alpha = J_h \pi$, then the k transformed vectors that combine to form α correspond to the k depth $h-1$ policy subtrees that combine to form π.

Figure 2.12 graphically shows this relationship. On the left we see three α

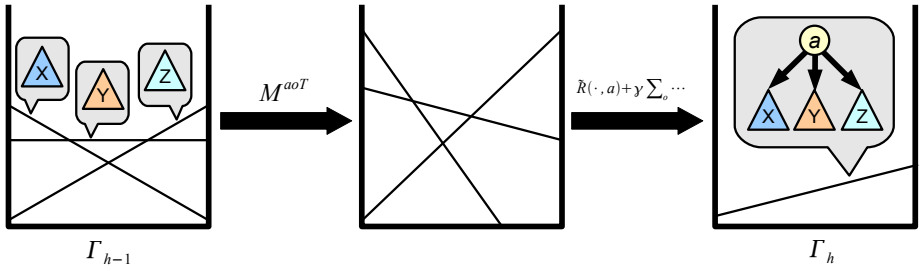


Figure 2.12: The relationship between composing policy subtrees and taking an affine combination of transformed α vectors.

vectors from Γ_{h-1}, along with their corresponding policy trees, labeled X, Y, and Z. In the middle, the α vectors have been transformed by the mappings M^{aoT} for varying values of o. On the right, we have taken a linear combination of the vectors and added the immediate reward $\tilde{R}(\cdot, a)$ to form one of the α vectors in Γ_h. This α vector is the value function of the tree formed by selecting action a at the root and transitioning to one of the subtrees X, Y, or Z based on the resulting observation.

2.9 Prior Research on POMDPs

There is a rich literature of prior work on POMDPs. We briefly review research in several areas. A separate discussion of prior work on science autonomy and exploration planning is found in §8.1.

2.9.1 Foundations

Drake (1962) and Astrom (1965) formulated the POMDP planning model. Sondik (1971, 1978) developed the first POMDP solution algorithms. Papadimitriou and Tsitsiklis (1987) provided early computational complexity results for MDPs and POMDPs. More recently, Littman (1996) and Cassandra (1998a) analyzed the complexity of various classes of POMDPs.

2.9.2 Value Iteration

Many early POMDP solution algorithms performed exact value iteration using piecewise linear convex value function representations (Sondik, 1971; Monahan, 1982; Cheng, 1988; White, 1991). These algorithms had high computational com-

plexity both in theory and in practice; they were largely impractical given the computing hardware available at the time.

The process of computing the Bellman update H used by value iteration can be broken up into (1) choosing a set of important beliefs, and (2) at each selected belief, performing a local "point-based" update that calculates the optimal α vector for that belief from HV (Cheng, 1988). The Witness algorithm (Littman, 1996) uses this idea to compute an exact Bellman update using a series of local updates at points selected using linear programs. Witness is more tractable than earlier exact VI techniques because it generates far fewer dominated α vectors.

The Incremental Pruning algorithm (Cassandra et al., 1997) achieved similar performance improvement. Incremental Pruning breaks up calculation of the Bellman update into a series of batch operations and interleaves these operations with pruning steps so that dominated α vectors are pruned earlier in the calculation.

Larger POMDPs require approximation techniques discussed below, including more compact approximate value function representations and point-based value iteration techniques, which break up the global Bellman update into smaller focused updates.

2.9.3 Point-Based Value Iteration

The Witness algorithm computes an exact Bellman update using a series of point-based updates at carefully selected points. On the other hand, if we are willing to accept some approximation error in the Bellman update, we can get away with updating fewer beliefs and selecting them less carefully. This is the key idea behind point-based value iteration algorithms.

The first algorithms in this class, Cheng's Point-Based Dynamic Programming (Cheng, 1988) and Zhang's Point-Based Value Iteration (Zhang and Zhang, 2001), maintain their value function in the form of a set of α vectors and, associated with each vector, a "witness" belief where the vector dominates all other vectors. They interleave many (cheap) point-based updates at the witness points with occasional (very expensive) exact Bellman updates that generate new α vectors and witness points. The algorithms differ mainly in their process for selecting witness points.

Batch update methods keep separate sets of pre-update α vectors and post-update α vectors, discarding the pre-update vectors after the batch is complete. Asynchronous update methods like the Incremental Linear-Function Approach (Hauskrecht, 2000) keep a single set of α vectors and add new vectors to the set as they are generated, so that they can immediately benefit subsequent point-based updates. Because old vectors are not automatically discarded, with asynchronous updates it becomes more important to carefully prune dominated vectors, so as to keep the representation compact.

Pineau's Point-Based Value Iteration (PBVI) (Pineau et al., 2003b, 2006) is perhaps the most widely used point-based algorithm. PBVI selects a finite set $\tilde{\mathcal{B}}$ of beliefs and performs point-based updates on all points of $\tilde{\mathcal{B}}$ in synchronous batches until a convergence condition is reached, then uses a heuristic to expand $\tilde{\mathcal{B}}$ and continues the process. They show that each batch update approximates the Bellman update and repeated batch updates converge to an approximation of V^* whose error is related to the sample spacing of $\tilde{\mathcal{B}}$. In extensions of this work, they improved the heuristic for selecting new beliefs (Pineau and Gordon, 2005) and improved update efficiency by storing witness points in a metric tree (Pineau et al., 2003a).

The PERSEUS algorithm (Spaan and Vlassis, 2005) uses batch updates that are slightly weaker but considerably faster than those of PBVI. During each update, PERSEUS backs up randomly sampled points from $\tilde{\mathcal{B}}$, stopping when all points of $\tilde{\mathcal{B}}$ have higher values than they did according to the previous value function. Since a single α vector often improves the value of several points, the batch usually requires many fewer than $|\tilde{\mathcal{B}}|$ point-based updates.

2.9.4 Value Function Representation

There has been considerable prior work on POMDP value function representation; Hauskrecht (2000) provides a good survey. We mention only a few papers here.

The use of the max-planes representation, also called the piecewise linear convex (PWLC) representation, goes back to some of the earliest work on POMDPs (Sondik, 1971). Max-planes representations constructed during value iteration usually contain many dominated α vectors that must be pruned for efficiency; many researchers have considered different pruning approaches (Sondik, 1971; Monahan, 1982; White, 1991; Littman, 1996; Cassandra et al., 1997).

Pineau et al. (2006) showed that if a max-planes approximation is formed by taking the gradient at beliefs in a sample set, the max-norm error of the approximation is linear in the sample spacing according to the 1-norm. In Chapter 5 we generalize this result using weighted norm machinery. Bertsekas and Tsitsiklis (1996) includes a useful discussion of weighted norms. Munos (2004) found some related results applying weighted max-norm machinery to value iteration.

Approximating V^* using the max-planes representation is an instance of the general problem of approximating a convex body with a circumscribed polytope. This is the subject of a rich parallel literature in operations research and mathematics journals. Rote (1992) gives an overview and a detailed quantitative analysis for plane figures. Gruber (1993) developed a number of asymptotic bounds for approximating convex bodies in arbitrary dimensions.

Whereas the usual max-planes representation takes the maximum of its individ-

ual α vectors, the SPOVA algorithm (Parr and Russell, 1995) combines α vectors using a soft-max function. This makes the mean Bellman residual over any set of sample beliefs a differentiable function of the α vectors, so the residual can be minimized (generating an approximation of V^*) through gradient descent.

Hauskrecht (2000) proposed an alternative PWLC representation, computing values by projecting onto the convex hull of a finite set of belief/value pairs. The exact convex hull is of course PWLC; an approximate convex hull called the "sawtooth" representation is not convex, but projections can be computed more efficiently.

Several researchers have used grid-based value function approximations, in which the belief simplex is broken up into local cells, and the approximation has some simple behavior within each cell. For example, it might be constant, or it might linearly interpolate the values at the vertices of the cell. Lovejoy (1991) developed an early fixed-resolution grid scheme; since then a variety of variable-resolution schemes have been proposed (Brafman, 1997; Zhou and Hansen, 2001).

Wang et al. (2006) represented a convex upper bound approximation to V^* using the maximum of a bounded number of quadratic surfaces rather than α vectors; the use of quadratics allowed additional freedom and a more compact representation, but also made approximate Bellman updates much more time-consuming.

2.9.5 Heuristic Search

Heuristic search played a role in many of the POMDP solution algorithms already discussed; this section is specifically concerned with heuristic search techniques for selecting POMDP beliefs to update.

The first example of this technique is the RTDP-BEL algorithm (Geffner and Bonet, 1998), based on the Real-Time Dynamic Programming (RTDP) algorithm for MDPs (Barto et al., 1995). RTDP selects states to update using a series of trials. Each trial starts at the initial state s_0 and explores forward, selecting actions greedily and simulating the transition dynamics to draw successor states stochastically until a goal state is reached. RTDP can be seen as a generalization of earlier search techniques like A^* (Hart et al., 1968) and Learning Real-Time A^* (Korf, 1990). RTDP-BEL is a straightforward extension of RTDP to POMDPs represented as belief-MDPs, using a grid-based value function approximation.

The same authors later developed improved algorithms to address some of RTDP's limitations. Labeled RTDP labels states once it knows from tracking residuals that they have approximately optimal values; labeled states can be skipped in later trials (Bonet and Geffner, 2003b). The Heuristic Dynamic Programming algorithm also labels finished states, but often more efficiently, and it fits better into a broad framework of find-and-revise algorithms that applies across many types of

problems (Bonet and Geffner, 2003a). Bonet and Geffner applied these algorithms only to MDPs, but they can be generalized to POMDPs in the same fashion as RTDP.

The Bounded RTDP (BRTDP) algorithm (McMahan et al., 2005) is another MDP heuristic search algorithm based on RTDP. Much like our HSVI algorithm (Smith and Simmons, 2004), BRTDP builds two value functions that bound V^* above and below, and prioritizes updates to states where the interval between the bounds is large. We regret not adding BRTDP to our performance comparison, due to a misunderstanding on our part that led us to think it would be difficult to apply to POMDPs. This comparison should be performed in future work.

More recently, Shani et al. have modified a number of point-based value iteration algorithms to use prioritized asynchronous updates similar to our HSVI algorithm (Smith and Simmons, 2004), and have developed several novel heuristics for selecting good belief points (Shani et al., 2006, 2007; Virin et al., 2007).

2.9.6 Structured Approaches

Most POMDP solution approaches that depend on the model representation assume that (1) the representation is chosen in advance by a domain expert, and (2) special structure is not specified *a priori*, but must be discovered from the transition dynamics. However, there is a growing body of solution techniques that rely on explicit additional structure to increase planning efficiency.

Theocharous (2002) developed a hierarchical POMDP (HPOMDP) formulation based on hierarchical hidden Markov models (Fine et al., 1998), and related to similar approaches for MDPs (Dietterich, 2000; Andre and Russell, 2002). The HPOMDP structure enables a bottom-up planning process in which macro actions are built to solve subtasks (such as navigating from one end of a hallway to the other) and the overall policy decides which macros to apply based on high-level state information. HPOMDP planning was successfully applied to a robot navigation task.

The Policy-Contingent Abstraction (PolCA+) algorithm for POMDPs (Pineau, 2004) assumes that a task hierarchy is specified and uses it to automatically aggregate states and observations at various levels of the hierarchy. A hierarchical policy is built bottom up. Pineau notes that, in the absence of perfect state information, it is unreliable to base subtask termination on detection that the system has entered a terminal state. To mitigate this problem, PolCA+ does not use explicit subtask termination—instead, high-level elements reconsider which subtask should be active at every time step.

Hansen and Zhou (2003) propose solving hierarchical POMDPs by generating policies in the form of hierarchical finite-state controllers (FSCs). The problem

of subtask termination is handled by including a terminal node in each FSC. Their approach comes with stronger policy optimality guarantees. Recently, Charlin et al. (2006) suggested a number of non-convex optimization techniques for generating compact hierarchical FSCs.

Factored models break up state information into distinct state variables. Often parts of the transition dynamics will depend on only a few of the variables, allowing more efficient reasoning. Boutilier and Dearden (1994) and Boutilier and Poole (1996) present a notion of factored MDPs and POMDPs in which some variables are totally irrelevant to the problem and can be abstracted away. Baum and Nicholson (1998) suggest non-uniform abstraction for MDPs that varies the level of detail at different planning horizons.

Some approaches operate directly on a factored POMDP model during the solution process. McAllester and Singh (1999) represent beliefs compactly using lossy Boyen-Koller belief simplification. St. Aubin et al. (2000) developed an Algebraic Decision Diagram representation for MDPs, extended by Hansen and Feng (2000) to apply to POMDPs and use a factored observation model.

Givan et al. (2003) present a unifying theoretical framework and useful notation that encompasses many approaches to state aggregation in MDPs and POMDPs. Feng and Hansen (2004) propose a version of Incremental Pruning that uses temporary state aggregation at each update step to reduce the size of the linear programs used during the pruning process.

Poupart and Boutilier (2003b) present Value-Directed Compression, a linear compression technique that adds only enough dimensions to (exactly or approximately) preserve the reward structure of the original problem.

Roy and Gordon (2003) and Roy et al. (2005) used the Exponential-Family PCA algorithm to perform non-linear compression of POMDP beliefs. This compression strategy sacrifices the PWLC structure of the POMDP value function, but leads to a much more compact belief space in typical problems where beliefs are very sparse.

2.9.7 Policy Gradient Approaches

Policy gradient methods search within a continuously parameterized class of policies, leveraging well-understood gradient search techniques. Policy gradient methods tend to scale well, as their complexity depends primarily on the complexity of the policy class rather than that of the POMDP dynamics. However, their effectiveness relies on expert judgment to select a simple policy class containing good policies (which may not exist, depending on the problem structure), and the methods can often be trapped by local optima.

Williams (1992) developed REINFORCE, an early policy search method that used gradient descent with neural network weights. Baird and Moore (1999) developed the "value and policy search" (VAPS) framework, which unifies a class of gradient search algorithms that combine aspects of policy search and value iteration; algorithms that instantiate VAPS have strong convergence guarantees. Baird and Moore applied VAPS to several types of problems including POMDPs. Baxter and Bartlett (2000) developed the GPOMDP policy gradient method, which improves REINFORCE in that it requires only a single sample path from a policy to calculate an approximate gradient and relies on fewer manually tuned parameters.

Meuleau et al. (1999) showed that when a stochastic policy is used and its parameters are action probabilities and transition probabilities in a policy graph, the exact policy gradient can be computed efficiently. Kearns et al. (2000) developed a technique for approximating a POMDP with a deterministic model generated by sampling a fixed set of "reusable trajectories" in advance. In the deterministic model exact policy gradients can be computed efficiently and the complexity is independent of the POMDP transition dynamics. The PEGASUS algorithm (Ng and Jordan, 2000) simplified the reusable trajectories method and achieved tighter complexity bounds.

Kakade (2001) showed that gradient search methods can be improved both in theory and in practice if, in place of the usual policy gradient, one uses a "natural gradient" that is invariant to certain aspects of the parameterization.

2.9.8 Policy Iteration

Policy iteration methods start with a simple policy and repeatedly refine it using a two step process of (1) evaluating the long-term reward of the policy, which generates a value function, and (2) generating a new policy from the value function (for example, using one-step lookahead). Exact policy iteration is guaranteed to converge to a global optimum, but (like exact value iteration) quickly becomes intractable as the size of the policies and value functions grows.

Sondik (1978) developed the first POMDP policy iteration algorithm. The primary policy representation was a mapping from convex subsets of the belief space to actions, but for efficient evaluation a secondary finite-state controller (FSC) representation was used. However, the conversion between these two representations is so complicated that Sondik's algorithm is not used in practice.

Hansen (1998) developed a more practical approach in which an FSC is the primary policy representation. Hansen's algorithm finds a global optimum, but scales poorly due to fast growth in the size of the FSC.

The Bounded Policy Iteration (BPI) algorithm (Poupart and Boutilier, 2003a, 2004) replaces the deterministic FSC of Hansen's algorithm with a stochastic FSC

of bounded size. BPI monotonically improves the FSC one node at a time, converging to a local optimum. Each refinement step is relatively efficient due to the bounded size of the FSC. They later combined BPI with Value-Directed Compression (Poupart and Boutilier, 2004).

2.9.9 History-Based Approaches

MDP policies map states to actions. In order to optimally handle partial observability, most POMDP algorithms use policies that map belief distributions to actions, relying on Bayesian updates during execution to reduce the history of actions and observations to a single belief distribution. However, for some POMDPs, it is sufficient to condition the action directly on the last few actions and observations, avoiding the complexity of Bayesian updates.

Loch and Singh (1998) used memoryless stochastic policies, in which the action distribution is conditioned only on the most recent observation. They demonstrated that when good memoryless policies exist they can often be found using the Sarsa(λ) reinforcement learning algorithm with eligibility traces. The eligibility traces serve as a weak form of memory during policy learning.

The Utile Suffix Memory (USM) approach (McCallum, 1995b,a) uses variable-length short-term memory. The policy is represented as a decision tree whose root node variable is the most recent (and thus presumably most relevant) observation, with earlier actions and observations in the history at increasing depth in the tree. If the best action can be selected using only recent actions and observations, USM tends to learn a shallow tree that avoids overfitting.

More recently, Brafman and Shani (2004) improved USM performance in the presence of noisy sensors. The Noisy USM method uses a sensor model to generate multiple weighted copies of each sample trajectory, corresponding to different possible interpretations of noisy observations.

2.9.10 Policy Heuristics

Policy heuristic algorithms use simplifying assumptions to efficiently generate policies that perform well for some problems but usually have no regret bounds in general. These methods can be effective, but selecting an appropriate policy heuristic for a problem is something of a black art.

A number of heuristics are based on relaxing the assumption of partial observability, turning the POMDP into a fully observable MDP whose value function can be computed efficiently. The maximum-likelihood heuristic (Nourbakhsh et al., 1995) was the first to be developed. They selected the most likely state from the belief distribution and chose the best action for that state in the fully observable

MDP model. Simmons and Koenig (1995) proposed a voting heuristic. Each state was allowed to cast a weighted vote for its best action in the fully observable MDP model, with weight equal to the state's likelihood in the belief distribution.

Littman et al. (1995) developed QMDP, probably the most widely used heuristic in this class. QMDP chooses the action with the greatest expected value under the assumptions that (1) the agent's belief correctly represents the probability distribution of the current state, and (2) after selecting an action, the state is revealed and the system proceeds according to the fully observable MDP dynamics. QMDP is often used as a baseline against which to measure other policies.

Hauskrecht (2000) showed that some policy heuristics can be interpreted as one-step lookahead with respect to a value function that is the fixed point of a modified Bellman update operator. One such update operator produces the MDP value function used by the QMDP heuristic, which consists of a single α vector. Hauskrecht developed another update operator that produces the Fast Informed Bound (FIB), a value function with $|\mathcal{A}|$ α vectors. He showed that the FIB value function is closer to V^* than the MDP value function but can still be computed efficiently.

Heuristic policies based on the fully observable MDP often underestimate the value of information gathering. The dual-mode controller developed by Cassandra et al. (1996) tries to mitigate this problem. It tracks the entropy of the belief distribution, which is a measure of the agent's uncertainty. When entropy is low the controller uses one of the fully observable MDP heuristics; when entropy is high the controller selects the action that most reduces the expected entropy on the next time step.

2.9.11 Continuous POMDPs

POMDPs derived from real-world applications often have continuous state, action, or observation spaces. Policy gradient approaches tend to generalize naturally to continuous POMDPs, since they are largely independent of how the POMDP transition dynamics are represented.

On the other hand, most value iteration researchers handle continuous variables using fixed-resolution discretization, with only a few exceptions. Thrun (2000) developed MC-POMDP, an early value iteration approach for continuous POMDPs. MC-POMDP represents a belief as a set of samples and uses particle filter techniques to approximately propagate beliefs through the continuous system dynamics. It performs Q-learning, representing the Q functions with nearest neighbor approximations.

Porta et al. (2006) showed that the piecewise linear and convex structure familiar for discrete POMDPs also exists for continuous POMDPs, allowing value

functions to be efficiently represented using "α functions" that take the place of α vectors. They generalized the PBVI algorithm for discrete POMDPs to handle continuous state, approximating beliefs with Gaussian mixture models.

2.9.12 Decentralized POMDPs

Recently, there has been growing interest in multi-agent systems. The partially observable stochastic game (POSG) framework models planning for multiple agents with different information resources; the agents may or not cooperate with each other (Hansen et al., 2004). A decentralized POMDP (DEC-POMDP) is a special type of POSG in which the agents are assumed to be purely cooperative, but still have limited ability to share information (Bernstein et al., 2002).

Unlike single-agent POMDPs, DEC-POMDPs can be used to decide when agents should communicate (Roth et al., 2006). Solving general DEC-POMDPs is known to be intractable, but approximate POMDP solution techniques such as BPI and PERSEUS have been adapted to the DEC-POMDP framework (Bernstein et al., 2005; Spaan et al., 2006).

2.9.13 Model Learning

Several of the planning approaches already discussed have a reinforcement learning flavor such that they can be used either in an offline setting with samples drawn from an *a priori* system model or in an online setting with samples drawn from actual execution.

There are also algorithms that explicitly learn a model and then used model-based planning techniques. Chrisman (1992) used the Baum-Welch algorithm to learn a Hidden Markov Model to resolve perceptual aliasing, in parallel with policy learning. More recently, Shani et al. (2005) made effective use of the Noisy USM algorithm to learn a history-based system model and the PERSEUS algorithm to generate an approximate policy.

Jaulmes et al. (2005) developed the MEDUSA algorithm, which assumes a class of POMDP models that is continuously parameterized, with a Dirichlet prior over the parameters. Sampled behavior of the system can be used to efficiently update the parameter distribution. In order to control the system, MEDUSA keeps a number of models drawn from the current model distribution, along with near-optimal policies for those models. At each time step one of the models is selected stochastically, and the agent acts according to the policy for that model.

The hidden state of a POMDP can lead to problems during learning—for instance, there may be irrelevant degrees of freedom in the model that are unconstrained by observations. To mitigate this problem, Littman et al. (2002) devel-

oped an alternative model formulation called the predictive state representation (PSR). PSRs replace hidden states with core tests that are directly related to action/observation histories. Any POMDP with n states can be represented by a PSR with at most n core tests. Singh et al. (2003) developed the first learning algorithm for PSRs; since then, several others have been proposed.

2.9.14 Applications

Most POMDP research papers mention applications of interest, but POMDP planning technology is still new enough that it has not seen significant integration into everyday decision processes. Cassandra (1998b) provides an overview of potential POMDP applications that, while dated, may still be the best survey available. We focus on a few more recent applications.

Hauskrecht and Fraser (2000) developed a POMDP model of ischemic heart disease and performed a test in which a POMDP planner recommended treatment actions for ten actual patients; the actions were critiqued by a physician but not actually performed. Actions selected by the planner were deemed appropriate in most cases, and the few mistakes that were made suggested correctible deficiencies in the model.

When a robot interacts with a person important aspects of state, such as the person's current goals, are not observable from moment to moment. The Nursebot social robot, designed for companionship in an elder care facility, used POMDP planning to manage human-robot dialogs, and later for motion control while searching for a person (Roy et al., 2000, 2003).

Hoey et al. (2007) developed a system for assisting persons with dementia with handwashing. Their system observes handwashing progress with a camera that covers the sink area. If the subject is having difficulty with the task, the system can respond with increasing levels of intervention, first prompting the subject, then notifying a human caregiver if necessary. The POMDP model includes attitude features such as the subject's overall dementia level and responsiveness to prompting. Their system performed well in simulation and with actors.

Hsiao et al. (2007) developed a POMDP planner for robotic grasping tasks in which the robot is uncertain about the shape and relative pose of the object to be grasped. The robot can improve its state estimate by touching the object at different points; POMDP planning makes effective use of these information-gathering actions.

Ferguson et al. (2004) developed a path planner for large environments with a few "pinch points" that may or may not be blocked. They introduced a special subclass of POMDPs called deterministic problems with hidden state (DPHS), and showed that their path planning problem fell into this class, allowing it to be solved

much more efficiently than a general POMDP.

Chapter 3

Focused Value Iteration

Value iteration is a powerful technique for generating good MDP and POMDP policies, but it is intractable for some problems. In particular, we have seen that for belief-MDPs the number of α vectors in the exact representation of V_h^* can grow doubly exponentially in h.

In this chapter we define a class of *focused value iteration* algorithms that mitigate some of the problems encountered by basic value iteration. Focused value iteration algorithms make use of the following ideas:

1. *Value function guarantees:* By choosing appropriate value function representations, we can maintain upper and lower bounds on the optimal value function V_∞^* and make policy quality guarantees without needing to calculate residuals.

2. *Point-based updates:* In place of the global exact Bellman update operator, we use point-based updates that typically improve the value function only over a small part of the state space. More of these point-based updates are required, but they are much cheaper to compute.

3. *Heuristic search:* We can use value bounds and other information to heuristically guide search, focusing on the best places to apply point-based updates. Significant speedups can be achieved by ignoring irrelevant parts of the state space.

The correctness of focused value iteration is based on the uniform improvability of the bounds (Zhang and Zhang, 2001). We define the notion of a "conservative incremental representation", which is a value function representation and associated rules for initialization and point-based updates that together ensure the value function is always uniformly improvable. We prove that when focused value iteration uses a conservative incremental representation: (1) individual point-based

updates can not make the bounds looser, and (2) the output policy satisfies a key correctness property that bounds the regret. These properties apply across both MDPs and POMDPs and are independent of how the heuristic search is conducted.

Throughout this chapter we assume that:

- The context is a discounted infinite-horizon MDP. We use the convention $V^* = V^*_\infty$.
- The state space may be discrete (a finite MDP) or continuous (the belief-MDP representation of a POMDP). Throughout the discussion, one can replace MDP states s with POMDP beliefs b and the MDP state space $\mathcal{S}$ with the POMDP belief space $\mathcal{B}$.

3.1 Value Function Representations and Update Operators

This section formalizes what we mean by a value function representation and an update operator, and introduces some properties we would like value functions to have.

Definition 3.1. *A* value function representation *is a tuple $\langle \mathcal{V}, \mathit{evaluate} \rangle$, where $\mathcal{V}$ is a set of data structures and $\mathit{evaluate} : \mathcal{V} \times \mathcal{S} \to \mathbb{R}$ is an evaluation rule. For any data structure $V \in \mathcal{V}$ and state s, $\mathit{evaluate}(V, s)$ gives the value at s. When the evaluation rule is obvious in context, we treat the data structure V and the value function it represents as interchangeable, and write $\mathit{evaluate}(V, s)$ as $V(s)$.*

Definition 3.2. *For two functions V, V', we say V* dominates *V', written $V \geq V'$, if for all s, $V(s) \geq V'(s)$. The $\leq$ and $=$ relations are defined in a similar way.*

Definition 3.3. *A* valid lower bound *is a value function V satisfying $V \leq V^*$. A* valid upper bound *satisfies $V \geq V^*$.*

Focused value iteration keeps two-sided bounds on the optimal value function V^*. In order for these data structures to sensibly be called "bounds", they must at a minimum be valid. Of course, we would also like them to approximate V^* as closely as possible.

Definition 3.4. *An* update operator *is a mapping $K : \mathcal{V} \to \mathcal{V}$.*

Definition 3.5. *An update operator K is* isotone *if $V \leq V'$ implies $KV \leq KV'$.*

Definition 3.6. *A* uniformly improvable lower bound *is a value function V satisfying $HV \geq V$. (In other words, applying H pushes the lower bound everywhere towards V^*.) Similarly, a* uniformly improvable upper bound *satisfies $HV \leq V$.*[1]

One can show that uniformly improvable bounds are always valid. We will use uniform improvability to establish that the two-sided bounds on V^* improve monotonically throughout the course of focused value iteration—they remain valid and never become looser as the algorithm progresses.

3.2 Using Uniform Improvability to Bound Regret

To better understand uniform improvability and the Bellman update we need some concepts and results from fixed point theory.

Definition 3.7. *A* Banach space *is a complete normed vector space.*

Example Banach spaces include:

1. For any finite set E, let X be the space of real-valued functions $E \to \mathbb{R}$, measuring the distance between functions using any (discrete) $\mathcal{L}^p$ norm. X is a Banach space. In particular, the space of value functions for a finite discrete MDP is a Banach space, since the value function domain is the finite state space.

2. For any finite dimensionality k and any closed subset $E \subseteq \mathbb{R}^k$, let X be the space of real-valued functions $E \to \mathbb{R}$, using any (continuous) $\mathcal{L}^p$ norm. X is a Banach space. In particular, the space of value functions for a belief-MDP is a Banach space, since the value function domain is the belief simplex, which is a closed subset of $\mathbb{R}^{|\mathcal{S}|}$.

Definition 3.8. *Let $(X, \|\cdot\|)$ be a normed vector space. An operator $K : X \to X$ is a contraction mapping with contraction factor $\xi < 1$ if for any $x, y \in X$*

$$\|Kx - Ky\| \leq \xi \|x - y\| . \tag{3.1}$$

In particular, one can verify that for discounted MDPs, H is a contraction mapping with contraction factor γ over the space of value functions with respect to the max-norm $\|\cdot\|_\infty$. This holds for both the discrete MDP and belief-MDP cases.

Definition 3.9. *A* complete lattice *is a partially ordered set $(L, \leq)$ with well-defined infimum and supremum operations that apply to subsets of L.*

[1] Zhang and Zhang (2001) introduced the concept of uniform improvability and proved some of its key properties. They defined uniform improvability only for lower bounds.

Of particular interest to us, the set of all real-valued value functions with the pointwise $\leq$ operator constitutes a complete lattice, provided that the function range is extended to allow the infimum and supremum functions to take on values of $-\infty$ and $+\infty$.

Definition 3.10. *For any policy π and value function V, define $\pi \otimes V$ to be the expected value of following π for one step of execution, then receiving long-term reward according to V. That is, if $V' = \pi \otimes V$, then*

$$V'(s) = (\pi \otimes V)(s) := R(s, \pi(s)) + \gamma \sum_{s'} T(s, \pi(s), s')V(s'). \tag{3.2}$$

In this context we can also use the notation "a" to refer to the policy that always chooses action a, so that

$$(a \otimes V)(s) := R(s, a) + \gamma \sum_{s'} T(s, a, s')V(s'). \tag{3.3}$$

Note that a more common alternative notation for $(a \otimes V)(s)$ is $Q^V(s, a)$. We find the $\otimes$ notation to be more flexible and convenient.

Lemma 3.11. *For any value function V, we have $HV = PV \otimes V$.*

Proof. Intuitively, this result holds because H corrects the value function using one-step lookahead, and P is the one-step lookahead operator. One can verify it by comparing the definitions of P, H, and $\otimes$, equations (2.8), (2.9), and (3.2). □

Lemma 3.12. *For any policy π, we have $J\pi = \pi \otimes J\pi$.*

Proof. The left-hand side of the equality is the expected long-term reward from executing π at every time step. The right-hand side is the expected reward from executing π at the first time step, then receiving the expected reward of executing π for all remaining time steps. Obviously, these quantities are equal.

One can reach the same result algebraically by comparing equations (2.3) and (3.2). □

Theorem 3.13 (Banach fixed point theorem (Granas and Dugundji, 2003))**.** *Let X be a Banach space and let K be a contraction mapping on X. Then K has a unique fixed point $x^* \in X$.*

Theorem 3.14 (Tarski's fixed point theorem (Tarski, 1955))**.** *Let $(L, \leq)$ be a complete lattice and let $K : L \to L$ be isotone. Then the set of fixed points of K in L is also a complete lattice. As a result, K has a least fixed point u and a greatest fixed point v. Furthermore, for all $x \in L$, $x \leq Kx$ implies $x \leq v$.*

We are now ready to present some basic theory on bounding regret using uniform improvability.

Theorem 3.15. *A uniformly improvable value function is a valid bound. If V^L is a value function satisfying $V^L \leq HV^L$, then $V^L \leq V^*$. Similarly, $HV^U \leq V^U$ implies $V^* \leq V^U$.*

Proof. We have

(P1) H is isotone.

(P2) $V^L \leq HV^L$.

(P3) H has a unique fixed point because it is a contraction mapping under the assumption $\gamma < 1$.

(P4) V^* is the greatest fixed point of H, since it is the unique fixed point.

Applying Tarski's fixed point theorem, (P1), (P2), and (P4) imply that $V^L \leq V^*$. There is a symmetrical argument for V^U. □

Theorem 3.16. *If V^L is a uniformly improvable lower bound, then $V^L \leq JPV^L$. In other words, the policy PV^L induced by one-step lookahead with V^L achieves at least the expected value specified by V^L.*

Remark 3.17. *Compare with Hauskrecht (2000) (Theorems 10 and 14), which establishes a similar result for specific classes of belief-MDP value functions, such as those derived from finite state controllers and from point-based updates to the max-planes representation.*

Proof. Let V^L be a uniformly improvable lower bound. Let K be the mapping $X \mapsto PV^L \otimes X$. We have:

(P1) $HV^L = PV^L \otimes V^L = KV^L$, from Lemma 3.11 and the definition of K.

(P2) $V^L \leq KV^L$, from (P1) and the uniform improvability of V^L.

(P3) K is isotone (this is straightforward to show).

(P4) K has a unique fixed point because it is a contraction mapping under the assumption $\gamma < 1$.

(P5) $JPV^L = PV^L \otimes JPV^L = KJPV^L$, from Lemma 3.12 and the definition of K.

(P6) JPV^L is the greatest fixed point of K since it is the unique fixed point, which follows from (P4) and (P5).

Applying Tarski's fixed point theorem, (P2), (P3), and (P6) imply that $V^L \leq JPV^L$. □

Theorem 3.18. *If $V^L \leq V^* \leq V^U$ and V^L is uniformly improvable, then the regret from executing PV^L satisfies*

$$\textit{regret}(PV^L) \leq V^U(s_0) - V^L(s_0). \tag{3.4}$$

Proof.

$$\begin{aligned} \text{regret}(PV^L) &\leq V^*(s_0) - JPV^L(s_0) && \text{[definition of regret]} && (3.5)\\ &\leq V^U(s_0) - JPV^L(s_0) && [V^* \leq V^U] && (3.6)\\ &\leq V^U(s_0) - V^L(s_0). && \text{[Theorem 3.16]} && (3.7) \end{aligned}$$

□

These theorems show that if we can generate uniformly improvable bounds $V^L \leq V^* \leq V^U$ with a tight bounds interval at the initial state s_0, the policy PV^L is guaranteed to have small regret.

We are not aware of any statements of Theorems 3.15, 3.16, and 3.18 predating our work in Smith and Simmons (2004), although these results are relatively straightforward applications of Tarski's fixed point theorem.

3.3 Generating Uniformly Improvable Bounds

There are a number of existing techniques for generating valid bounds on V^*; Hauskrecht (2000) provides a detailed discussion and specific algorithms for POMDPs, with proofs of validity. This section generalizes some of those results, showing that:

- Several existing algorithms conform to just two algorithm schemas, simplifying analysis.
- The resulting bounds are not just valid, but also uniformly improvable (making them suitable for focused value iteration).
- The same concepts apply across both finite MDPs and POMDPs.

The specific bound initialization techniques we use for POMDPs are discussed in Chapter 4.

Theorem 3.19. *For any policy π, the value function $J\pi$ is uniformly improvable (and hence valid).*

Proof. Let a policy π be given. We need to show that $HJ\pi \geq J\pi$. We have

$$HJ\pi = PJ\pi \otimes J\pi \qquad \text{[definition of } H\text{]} \tag{3.8}$$

$$\geq \pi \otimes J\pi \qquad \text{[in general, } PV = \arg\max_{\pi}(\pi \otimes V)\text{]} \tag{3.9}$$

$$= J\pi. \qquad \text{[Lemma 3.12]} \tag{3.10}$$

□

Theorem 3.19 suggests a simple algorithm schema called *policy search lower bound initialization*: perform some form of policy search and evaluate the best policy you find. The resulting value function is a uniformly improvable lower bound. The blind policy and UMDP methods for generating POMDP lower bounds conform to this schema (Hauskrecht, 1997, 2000).

Theorem 3.20. *Let K be an update operator such that for any value function V, $KV \geq HV$. If V^U is a fixed point of K, then V^U is a uniformly improvable upper bound (and hence a valid upper bound). A symmetrical result holds for lower bounds.*

Proof. Choose K and V^U according to the conditions of the theorem. We need to show that $HV^U \leq V^U$. We have

$$HV^U \leq KV^U \qquad \text{[definition of } K\text{]} \tag{3.11}$$

$$= V^U. \qquad \text{[}V^U \text{ is a fixed point of } K\text{]} \tag{3.12}$$

□

This result underlies a corresponding schema for the upper bound called *optimistic value iteration upper bound initialization*: find an update operator K whose result is always larger than H and calculate its fixed point. The "optimistic" operator K is often derived by relaxing the problem in some way. The MDP, QMDP, and Fast Informed Bound methods for generating POMDP upper bounds conform to this schema (Hauskrecht, 2000).

3.4 Point-Based Updates

Focused value iteration is based on the insight that calculating the exact global Bellman update H is often intractable. This motivates our interest in related update operators that are cheaper to compute.

Definition 3.21. *A* point-based update operator *is a mapping $K : \mathcal{S} \to \mathcal{V} \to \mathcal{V}$. For any state s and value function V, $K_s V$ is the updated value function that results from applying K to V at state s.*[2]

We are interested in point-based update operators that can be used to locally improve the value function. Updating a value function V at a state s typically modifies V so that it is a better approximation to V^* at s and possibly in a neighborhood around s, while leaving V unchanged over most of the state space.

Definition 3.22. *The* single-point Bellman update *is the point-based update operator H^{SP} such that*

$$H_s^{SP} V(s') = \begin{cases} HV(s') & s = s' \\ V(s') & \textit{otherwise}. \end{cases} \tag{3.13}$$

The single-point Bellman update H^{SP} is perhaps the simplest example of a useful point-based update operator. Rather than applying the Bellman update H to the whole value function, it modifies the value only at a single state s. In Chapter 4, H^{SP} is used as the update operator in implementations of focused value iteration for finite MDPs and POMDPs. In this section, we use H^{SP} as a baseline to compare with other point-based update operators.

Definition 3.23. *If K and L are two update operators, we say that K* dominates *L* for lower bounds *if for all uniformly improvable lower bounds V, $KV \geq LV$. (In other words, between K and L, K pushes the result further in the direction of V^*.) Similarly, K* dominates *L* for upper bounds *if for all uniformly improvable upper bounds V, $KV \leq LV$.*

Intuitively, between two update operators K and L, we prefer the one that provides more improvement each time it is applied. The dominance relation between operators is one way to formalize that idea. As long as the result is valid, we prefer larger values for lower bounds and smaller values for upper bounds, in both cases

[2] In the POMDP literature, the term "point-based update" often refers to a specific type of update to the max-planes representation (Pineau et al., 2006; Spaan and Vlassis, 2005). Our definition is compatible but more general.

bringing the result closer to V^*. Note that the dominance relationship is quantified over uniformly improvable value functions only; this weakens the conditions needed to apply later theorems.

Definition 3.24. *An update operator K is* conservative *for lower (upper) bounds if:*

1. *K is dominated by H for lower (upper) bounds.*
2. *K dominates the identity operator for lower (upper) bounds.*

A point-based update operator K is conservative if K_s is conservative for every state s.

A conservative update operator is one that neither "overshoots" the global Bellman update H nor weakens the bound. We will show later that conservative update operators preserve uniform improvability (and hence validity) of the bounds.

Definition 3.25. *A point-based update operator K is* strong *for lower (upper) bounds if:*

1. *K is dominated by H for lower (upper) bounds.*
2. *K dominates H^{SP} for lower (upper) bounds.*

(Note that a strong operator is always a conservative operator.)

The fact that an update operator is conservative essentially provides a correctness guarantee, but does not imply that the operator does anything useful. For example, the identity operator is conservative. On the other hand, a strong operator must provide at least as much improvement as the single-point Bellman update. In Chapter 6, we will use this property to guarantee termination of focused value iteration when certain heuristic search algorithms are used.

Definition 3.26. *An* incremental representation *is a tuple $\langle \mathcal{V}, evaluate, V_0, K \rangle$, where $\langle \mathcal{V}, evaluate \rangle$ is a value function representation, $V_0 \in \mathcal{V}$ is a uniformly improvable initial value function, and K is a point-based update operator.*

Definition 3.27. *An incremental representation is conservative (strong) if its point-based update operator K is conservative (strong).*

We are now ready to explore the relationship between uniform improvability and various update operators. Zhang and Zhang (2001) proved an important basic result.

Theorem 3.28. *(Zhang and Zhang, 2001). The Bellman update operator H preserves uniform improvability.*

We can trivially generalize this result to all conservative update operators.

Theorem 3.29. *Conservative update operators preserve uniform improvability.*

Proof. Let K be a conservative update operator for lower bounds and V^L be a uniformly improvable lower bound. Then

$$KV^L \leq HV^L \qquad \text{[definition of conservative update]} \tag{3.14}$$

$$\leq HKV^L. \qquad [V^L \leq KV^L \text{ and } H \text{ is isotone}] \tag{3.15}$$

$KV^L \leq HKV^L$ means KV^L is a uniformly improvable lower bound. There is a symmetrical argument for upper bounds. □

3.5 The Focused Value Iteration Algorithm

Focused value iteration describes a class of value iteration algorithms that make use of value function guarantees, focused updates, and heuristic search. This section presents an algorithm for focused value iteration that can be used with a variety of modules implementing point-based updates and heuristic search, as long as the modules conform to the type signatures we specify. Furthermore, when used with conservative incremental representations for the bounds, focused value iteration is sound in the sense that, if it terminates, it is guaranteed to return a policy whose regret is within the specified bound.

Algorithm 3.1 defines INCREMENTALSIG, a type signature for modules implementing incremental representations. The declarations in the signature correspond directly to elements of the tuple $\langle \mathcal{V}, \text{evaluate}, V_0, K \rangle$ in the definition of an incremental representation. The initial value function V_0 is computed using a call to `initialValueFunction`, which takes no arguments. The update operator K is implemented via the `update` function, with arguments ordered differently for notational convenience.

Algorithm 3.2 defines SEARCHSIG, the type signature for modules implementing heuristic search. Focused value iteration relies on a heuristic search module to select states for point-based updates. A heuristic search implementation needs a way to initialize its internal state and a way to choose the next point to update based on the current bounds and its internal state.

Finally, Algorithm 3.3 defines focused value iteration. The core algorithm is simple, as the modules take care of most of the details. In outline, focused value iteration (1) initializes its lower and upper bound representations and heuristic search

Algorithm 3.1 INCREMENTALSIG, a type signature for incremental representation implementations.

1: **type** $\mathcal{V}$ [the value function representation]
2: **function** evaluate: $\mathcal{V} \times \mathcal{S} \to \mathbb{R}$
3: **function** initialValueFunction: $\emptyset \to \mathcal{V}$
4: **function** update: $\mathcal{V} \times \mathcal{S} \to \mathcal{V}$

Algorithm 3.2 SEARCHSIG, a type signature for heuristic search implementations.

1: **uses implementation** <LB> **conforming to** INCREMENTALSIG
2: **uses implementation** <UB> **conforming to** INCREMENTALSIG
3: **type** $\mathcal{X}$ [the internal state of the search algorithm]
4: **function** initialSearchState: $\emptyset \to \mathcal{X}$
5: **function** chooseUpdatePoint: $(<\text{LB}>.\mathcal{V} \times <\text{UB}>.\mathcal{V} \times \mathcal{X}) \to (\mathcal{S} \times \mathcal{X})$

state, (2) repeatedly applies point-based updates at points selected by the heuristic search, and (3) returns the lower bound V^L when the regret from executing PV^L is guaranteed to be smaller than δ.

The following theorem provides a soundness guarantee for focused value iteration when used with conservative incremental representations.

Theorem 3.30. *Let* <LB> *and* <UB> *be conservative incremental representations for lower and upper bound functions, respectively. Then, if focused value iteration terminates, it returns a lower bound function* V^L *such that* $\text{regret}(PV^L) \leq \delta$.

Proof. Focused value iteration maintains the invariant that the bounds V^L and V^U are uniformly improvable, because

1. They are initialized via a conservative incremental representation; hence they are initially uniformly improvable.
2. They continue to be uniformly improvable after applications of K by Theorem 3.29.

Making use of the uniform improvability, if `focusedValueIteration` terminates, we have

$$\begin{aligned} \text{regret}(PV^L) &\leq V^U(s_0) - V^L(s_0) && \text{[Theorem 3.18]} && (3.16) \\ &\leq \delta. && \text{[termination condition, line 10]} && (3.17) \end{aligned}$$

Algorithm 3.3 Focused value iteration.

```
1:  uses implementation <LB> conforming to INCREMENTALSIG
2:  uses implementation <UB> conforming to INCREMENTALSIG
3:  uses implementation <SEARCH> conforming to SEARCHSIG
4:
5:  function focusedValueIteration(δ) :
6:      [returns a value function V^L such that regret(PV^L) ≤ δ]
7:      x ← <SEARCH>.initialSearchState()
8:      V^L ← <LB>.initialValueFunction()
9:      V^U ← <UB>.initialValueFunction()
10:     while (V^U(s_0) − V^L(s_0)) > δ :
11:         s, x ← <SEARCH>.chooseUpdatePoint(V^L, V^U, x)
12:         V^L ← <LB>.update(V^L, s)
13:         V^U ← <UB>.update(V^U, s)
14:     return V^L
15:
16: function chooseAction(V^L, s) :
17:     [used to select actions during policy execution]
18:     return arg max_a [R(s, a) + γ Σ_s' T(s, a, s')V^L(s')]
```

□

Later we show that with a strong incremental representation and an appropriate heuristic search implementation, focused value iteration is guaranteed to terminate. Heuristic search is covered in detail in Chapter 6.

Chapter 4

POMDP Value Function Representation

The efficiency of focused value iteration strongly depends on how the value function bounds V^L and V^U are represented. This chapter presents a number of novel value function representations and relates them to representations from prior research in a broader overall framework.[1]

The chapter has two unifying themes. First, all of the representations we present are strong incremental representations; they come associated with algorithms to (1) generate initial uniformly improvable bounds, and (2) efficiently perform point-based updates in a way that preserves uniform improvability. This is the thread that ties the representations back to the focused value iteration soundness theorem from the preceding chapter.

Second, we will repeatedly see a trade-off between fast point-based updates and *strong generalization*. We informally say that a representation has strong generalization if a point-based update at a belief b tends to improve the value function not only at b but also at other related beliefs b'. Our various representations fall at different points on the spectrum between fast updates and strong generalization. Experiments indicate that, at least for some benchmark problems, a compromise approach provides best performance.

[1] Software implementations of most of the value function representations described in this chapter are freely available as part of the ZMDP software package, which you can download at `http://www.cs.cmu.edu/~trey/zmdp/`.

4.1 Linear Algebra Notation

Earlier sections presented many calculations that can be interpreted in terms of matrix multiplication, vector addition, and inner products. In order to facilitate later discussions of data structure representation, this section introduces notation that makes the connections to linear algebra operations more explicit.

Recall the M^{ao} linear mapping defined by equation (2.51), §2.8:

$$(M^{ao}b)(s') := \Pr(s', o \mid b, a) = \sum_s O(a, s', o)T(s, a, s')b(s). \tag{4.1}$$

The following formula reduces application of M^{ao} to realizable linear operations and explicit use of stored parameters from the POMDP model:

$$M^{ao}b = \omega^{ao} * (T^a b), \tag{4.2}$$

where the terms on the right-hand side are defined such that

$$\omega^{ao}(s') := O(a, s', o) \qquad [\omega^{ao} \text{ is a length-}|\mathcal{S}| \text{ vector}] \tag{4.3}$$

$$(x * y)(s') := x(s')y(s') \qquad [* \text{ "entry-wise multiplies" two vectors}] \tag{4.4}$$

$$T^a_{s's} := T(s, a, s'). \qquad [T^a \text{ is an } |\mathcal{S}| \times |\mathcal{S}| \text{ matrix}] \tag{4.5}$$

Furthermore, in the process of calculating the Bellman update, given a vector α, one often needs to calculate a vector β such that for every belief b,

$$\beta^T b = \alpha^T (M^{ao} b). \tag{4.6}$$

Solving for β gives

$$\beta = M^{aoT}\alpha = T^{aT}(\omega^{ao} * \alpha). \tag{4.7}$$

Finally, we can explicitly recognize the formula

$$\beta = \tilde{R}(\cdot, a) + \gamma \sum_o M^{aoT}\alpha^o \tag{4.8}$$

as taking the sum of vectors, defining the vectors r^a such that for all s,

$$r^a(s) := \tilde{R}(s, a) \tag{4.9}$$

Algorithm 4.1 Blind policy lower bound initialization (Hauskrecht, 1997).

```
1: function blindPolicyInitialize() :
2:    for a ∈ A :
3:       [policy evaluation for the policy of always taking action a]
4:       α^a ← solve for α: [α = a ⊗ α]
5:    return {α^1, ..., α^|A|}
```

4.2 Constructing Uniformly Improvable Bounds

This section builds on the basic results in §3.3 to discuss specific methods for initializing the bounds used by focused value iteration.

4.2.1 Lower Bound Initialization: The Blind Policy Method

In §3.3 we presented an algorithm schema called policy search lower bound initialization. Algorithms conforming to the schema perform some form of policy search and evaluate the best policy found. The resulting value function is guaranteed to be uniformly improvable.

The Blind Policy Method (Hauskrecht, 1997) is an instance of this schema. The blind policy of always selecting a particular action a has a linear value function α^a that is relatively easy to calculate. Having calculated all the α^a planes, the agent can construct the following meta-policy:

1. At the initial step of execution starting from belief b_0, choose the blind policy whose plane α^a maximizes $\alpha^{aT} b_0$.

2. Throughout execution, always select action a.

It is easy to see that the value function for the meta-policy is the point-wise maximum of the individual blind policies:

$$V^{\text{blind}} := \max \Gamma^{\text{blind}} := \max\{\alpha^1, \ldots, \alpha^{|\mathcal{A}|}\}. \tag{4.10}$$

Applying Theorem 3.19, the value function V^{blind} is a uniformly improvable lower bound, since it is the value function for the meta-policy. Algorithm 4.1 is an implementation of the blind policy method that returns the corresponding max-planes representation Γ^{blind}.

Note that each α^a is the solution to the equation

$$\alpha^a = a \otimes \alpha^a, \tag{4.11}$$

which reduces to the linear system

$$\alpha^a = r^a + \gamma T^{aT} \alpha^a \tag{4.12}$$

We can solve the linear system in at worst $O(|\mathcal{S}|^3)$ time using Gaussian elimination, or we may be able to do better with more sophisticated techniques. Since $|\mathcal{A}|$ planes must be calculated the overall running time of the blind policy method is at worst $O(|\mathcal{A}||\mathcal{S}|^3)$.

The class of blind policies is particularly useful for policy search lower bound initialization because:

1. All POMDPs have blind policies.
2. The value function of a blind policy is easy to compute and linear, so the blind policy method generates a max-planes representation.
3. The class contains only $|\mathcal{A}|$ policies, so it is easy to evaluate all of them.

However, it should be noted that for many problems all of the blind policies have poor performance, which means that the resulting value function V^{blind} provides a weak lower bound, slowing down the overall focused value iteration algorithm. In those cases, one may wish to use a policy class tailored to the particular problem as a way of injecting domain knowledge and speeding up the solution process.

4.2.2 Upper Bound Initialization: The Fast Informed Bound Method

In §3.3 we presented an algorithm schema called optimistic value iteration upper bound initialization. Algorithms conforming to the schema perform value iteration with an update operator K that is "optimistic" in the sense that its result always pointwise dominates that of the true Bellman update H. Any fixed point of K is guaranteed to be uniformly improvable.

Hauskrecht (2000) showed that the operators H^{MDP} and H^{FIB} defined below are optimistic, and that their fixed points are valid upper bounds. Theorem 3.20 implies that they are also uniformly improvable.

Definition 4.1 (See, for example, Hauskrecht (2000))**.** *Define the* MDP update operator *according to*

$$(H^{MDP}V)(b) := \sum_s b(s) \max_a \left[R(s,a) + \gamma \sum_{s'} T(s,a,s') V(s') \right]. \tag{4.13}$$

$$\tag{4.14}$$

Algorithm 4.2 Fast informed bound upper bound initialization (Hauskrecht, 2000).

```
1:  function fastInformedBoundInitialize(δ) :
2:    [returns a δ-approximation of the fixed point of H^FIB]
3:    α^MDP ← the optimal value function of the underlying MDP
4:    for a ∈ A :
5:      α^a ← α^MDP
6:    loop:
7:      for a ∈ A :
8:        for o ∈ O :
9:          for s ∈ S :
10:           β^ao(s) ← max_{α∈{α^1,...,α^|A|}} Σ_{s'} T(s,a,s')O(a,s',o)α(s')
11:       β^a ← r^a + γ Σ_o β^ao
12:     ε ← max_{s,a} |β^a(s) − α^a(s)|
13:     for a ∈ A :
14:       α^a ← β^a
15:     if ε/(1 − γ) ≤ δ :
16:       return {α^1, ..., α^|A|}
```

Definition 4.2 (Hauskrecht (2000))**.** *Define the* Fast Informed Bound (FIB) update operator *according to*

$$(H^{FIB}V)(b) := \max_a \sum_s b(s) \left[R(s,a) + \gamma \sum_o \max_{\alpha \in \Gamma} M^{aoT} \alpha \right]. \tag{4.15}$$

The fast informed bound method generates a uniformly improvable upper bound by calculating the fixed point V^{FIB} of the operator H^{FIB}. Hauskrecht proposed an approach for exactly calculating V^{FIB} based on the observation that it is equal to the optimal value function of a certain MDP with $|\mathcal{A}||\mathcal{O}||\mathcal{S}|$ states. This MDP can be constructed and solved with computation time polynomial in $|\mathcal{A}||\mathcal{O}||\mathcal{S}|$.

Our implementation uses a simple iterative approach to calculate an approximation to V^{FIB}. Algorithm 4.2 returns an approximation $\hat{V}$ such that

$$\|\hat{V} - V^{\text{FIB}}\|_\infty \leq \delta. \tag{4.16}$$

Each iteration of the main loop in the implementation applies the H^{FIB} operator once to the current value function. When the residual between successive loops is sufficiently small, the max-planes representation of the value function is returned.

Note that on line 5 we initialize each α vector with the optimal value function

V^{MDP} of the underlying MDP. This does not affect the final result since iteration of H^{FIB} is globally convergent, but it often happens that $V^{\text{MDP}} \approx V^{\text{FIB}}$, speeding convergence. As with the overall algorithm, one can solve exactly for the MDP optimal value function in polynomial time using linear programming or use an approximate iterative approach such as MDP value iteration.

The computation time of Algorithm 4.2 is dominated by the inner loop at line 10, which takes $O(|\mathcal{A}|^2|\mathcal{O}||\mathcal{S}|^2)$ time per loop. The contraction factor of γ in H^{FIB} provides first-order convergence and makes it possible to bound the number of loops required to reach a specific error δ, but we will not provide a detailed analysis. In practice we find this iterative approach is both easy to implement and efficient enough that running it to "completion" near machine precision is a small time investment relative to later steps in focused value iteration.

4.3 Adding Planes to the Max-Planes Representation

Most existing point-based value iteration algorithms use the max-planes representation and generate a new α vector for a belief point b by in effect calculating the gradient of HV at b. A set of these α vectors may be calculated in batch (Spaan and Vlassis, 2005; Pineau et al., 2006) or asynchronously (Hauskrecht, 2000).

This section presents a strong incremental representation called ADDPLANE that formalizes the asynchronous approach. ADDPLANE is defined as follows:

- A value function is represented using a set Γ of α vectors.
- The evaluation semantics are defined by the operator J^{AP}, which takes the point-wise maximum of the vectors in Γ.
- The initial uniformly improvable bound Γ_0 is generated using the blind policy method.
- The ADDPLANE point-based update operator, H^{AP}, adds one vector to Γ in the manner we describe below.

Now we provide more detail. J^{AP} takes the point-wise maximum of the vectors in Γ:

$$J^{\text{AP}}\Gamma := \max \Gamma, \tag{4.17}$$

in other words

$$J^{\text{AP}}\Gamma(b) = \max_{\alpha \in \Gamma} \alpha^T b. \tag{4.18}$$

Algorithm 4.3 ADDPLANE, an incremental representation.

```
1:  function ADDPLANE.evaluate(Γ, b) :
2:      return max_{α∈Γ} α^T b
3:
4:  function ADDPLANE.initialValueFunction() :
5:      return blindPolicyInitialize()
6:
7:  function ADDPLANE.update(Γ, b) :
8:      for a ∈ A :
9:          for o ∈ O :
10:             b^{ao} ← ω^{ao} * (T^a b)
11:             β^{ao} ← arg max_{α∈Γ} α^T b^{ao}
12:         β^a ← r^a + γ Σ_o T^{aT} (ω^{ao} * β^{ao})
13:     β* ← arg max_{β∈{β^1,...,β^{|A|}}} β^T b
14:     return Γ ∪ {β*}
```

The update operator H^{AP} is formally defined as a mapping on max-planes representations. It takes a vector set Γ and adds a single new α vector as computed by the `update` function of Algorithm 4.3. (We provide a more intuitive informal definition below.)

In general, it is natural to think of an update operator on value function representations as an update operator on value functions themselves. Thus we extend our notation such that, when $V = J^{\text{AP}}\Gamma$,

$$H_b^{\text{AP}} V := J^{\text{AP}} H_b^{\text{AP}} \Gamma. \tag{4.19}$$

Note that $H_b^{\text{AP}} V$ is obviously ill-defined when V is not representable using the max-planes representation. It is also potentially ill-defined when there are many possible representations for a particular value function V (none of which is canonical). Thus we will only use this convention in context when it is obvious how V is represented. The same shorthand generalizes to other matched pairs of evaluation and update operators defined later; for example, the J^{CH} and H^{CH} operators in §4.6.1.

Similarly, we can think of operators nominally defined on value functions as operators on value function representations. For example, when $V = J^{\text{AP}}\Gamma$ and $HV = J^{\text{AP}}\Gamma'$, we use the extended notation

$$H\Gamma := \Gamma'. \tag{4.20}$$

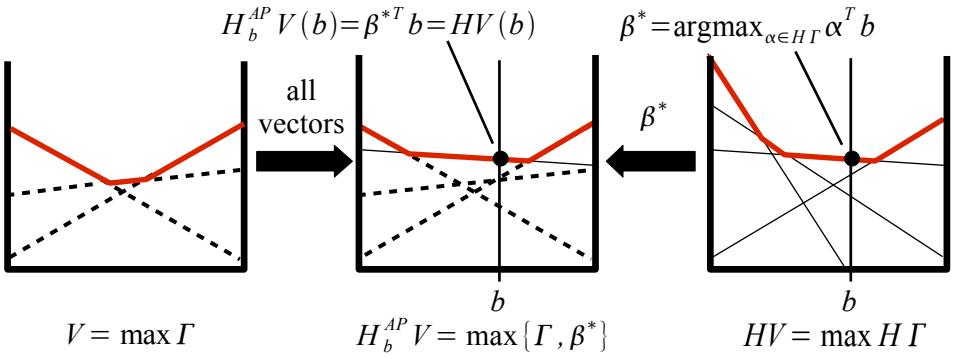


Figure 4.1: The relationship between V, $H_b^{\text{AP}}V$, and HV.

The same caveats apply concerning existence and canonical representation. We use this second convention mostly to provide informal intuition.

This leads us to an informal definition of the ADDPLANE point-based update operator:

$$H_b^{\text{AP}}\Gamma := \Gamma \cup \{\beta^*\}, \tag{4.21}$$

where

$$\beta^* = \arg \max_{\alpha \in H\Gamma} \alpha^T b. \tag{4.22}$$

H^{AP} locally improves V by by adding to Γ the α vector from $H\Gamma$ that is maximal at b.[2] Figure 4.1 shows this relationship graphically. We see V, $H_b^{\text{AP}}V$, and HV at left, middle, and right, respectively. $H_b^{\text{AP}}V$ is constructed by taking the maximum of (1) all vectors in Γ, drawn with dashed lines, and (2) the single vector β^* from $H\Gamma$ that maximizes the value of b, drawn with a thin solid line.

It is well known that, using the H^{AP} computation defined by Algorithm 4.3, the vector β^* that is added to Γ is a *subgradient* of HV at b, meaning:

$$\beta^{*T} b = HV(b) \tag{4.23}$$

$$\beta^{*T} b' \leq HV(b') \text{ for every belief } b'. \tag{4.24}$$

See, for example, Zhang and Zhang (2001). Based on this understanding we can prove the following theorem.

Theorem 4.3. *H^{AP} is a strong point-based update operator for lower bounds,*

[2] We are glossing over issues of tie-breaking; Littman (1996) includes a detailed discussion.

making ADDPLANE *a strong incremental representation.*

Proof. Let $V = J^{\text{AP}}\Gamma$ be a uniformly improvable lower bound, let b be a belief, and let β^* be the vector H_b^{AP} adds to Γ. We have:

$$\begin{aligned}
H_b^{\text{AP}}V &= J^{\text{AP}}H_b^{\text{AP}}\Gamma && \text{[def. of } H_b^{\text{AP}} \text{ for value functions]} && (4.25)\\
&= J^{\text{AP}}(\Gamma \cup \{\beta^*\}) && \text{[def. of } H^{\text{AP}}, \beta^*] && (4.26)\\
&= \max\{J^{\text{AP}}\Gamma, \beta^*\} && \text{[def. of } J^{\text{AP}}] && (4.27)\\
&\geq J^{\text{AP}}\Gamma && \text{[def. of max]} && (4.28)\\
&= V. && \text{[def. of } V] && (4.29)
\end{aligned}$$

This shows that $H_b^{\text{AP}}V$ dominates V. We also have

$$\begin{aligned}
H_b^{\text{AP}}V(b) &\geq \beta^{*T}b && \text{[(4.27)]} && (4.30)\\
&= HV(b). && \text{[(4.23)]} && (4.31)
\end{aligned}$$

Together, (4.29) and (4.31) show that H_b^{AP} dominates the single-point Bellman update H_b^{SP}. Decomposing the two sides of the max operation in (4.27), we have

$$\begin{aligned}
J^{\text{AP}}\Gamma &= V && \text{[def. of } V] && (4.32)\\
&\leq HV. && [V \text{ unif. improvable}] && (4.33)
\end{aligned}$$

$$\beta^* \leq HV. \qquad \text{[(4.24)]} \qquad (4.34)$$

Together, (4.27), (4.33), and (4.34) establish that H_b^{AP} is dominated by H. Since H_b^{AP} dominates H^{SP} and is dominated by H for all b, we find that H^{AP} is a strong point-based update operator. □

4.4 Leveraging Sparsity with the Max-Planes Representation

POMDP problems often have sparse structure. The initial belief b_0 may be sparse, meaning that the initial state is known to be drawn from one of only a few possibilities. The transition function T may be sparse, meaning that any state has only a few possible successors. Finally, in some problems these properties combine to cause *persistent sparsity*, meaning that transitions tend to retain belief sparsity so that all reachable beliefs in the POMDP search graph are sparse.

This section discusses three low-level implementations of the ADDPLANE focused update algorithm. DENSE is the simplest implementation; COMPRESSED

	Per operation			Total time spent in line		
	$\alpha^T b^{ao}$	$T^a b$	$T^{aT}\beta$	Line 10	Line 11	Line 12
DENSE	$\|\mathcal{S}\|$	$\|\mathcal{S}\|^2$	$\|\mathcal{S}\|^2$	$\|\mathcal{A}\|\|\mathcal{O}\|\|\mathcal{S}\|^2$	$\|\mathcal{A}\|\|\mathcal{O}\|\|\Gamma\|\|\mathcal{S}\|$	$\|\mathcal{A}\|\|\mathcal{O}\|\|\mathcal{S}\|^2$
COMPR.	$\rho\|\mathcal{S}\|$	$\rho^2\|\mathcal{S}\|^2$	$\rho\|\mathcal{S}\|^2$	$\|\mathcal{A}\|\|\mathcal{O}\|\rho^2\|\mathcal{S}\|^2$	$\|\mathcal{A}\|\|\mathcal{O}\|\|\Gamma\|\rho\|\mathcal{S}\|$	$\|\mathcal{A}\|\|\mathcal{O}\|\rho\|\mathcal{S}\|^2$
MASKED	$\rho\|\mathcal{S}\|$	$\rho^2\|\mathcal{S}\|^2$	$\rho^2\|\mathcal{S}\|^2$	$\|\mathcal{A}\|\|\mathcal{O}\|\rho^2\|\mathcal{S}\|^2$	$\|\mathcal{A}\|\|\mathcal{O}\|\rho\|\Gamma\|\rho\|\mathcal{S}\|$	$\|\mathcal{A}\|\|\mathcal{O}\|\rho^2\|\mathcal{S}\|^2$

Table 4.1: Asymptotic time complexity of individual operations for different implementations of ADDPLANE.

and MASKED are alternate versions designed to improve efficiency in the presence of sparse structure. The implementations differ primarily in terms of how they realize the linear operations at lines 10, 11, and 12 of Algorithm 4.3, which dominate the running time of the `update` function.

Table 4.1 collects performance information about the various implementations. It will be explained in detail as we present each implementation. Our data structure terminology is drawn from the BLAS linear algebra package (Dongarra et al., 1988). Time complexity estimates for linear algebra operations are based on widely available algorithms optimized for these data structures.

Our simplest low-level implementation, called DENSE, uses dense storage mode for all matrices and vectors. With this representation, inner products like $\alpha^T b^{ao}$ require $O(|\mathcal{S}|)$ time, and matrix-vector multiplications like $T^a b$ and $T^{aT}\alpha$ require $O(|\mathcal{S}|^2)$ time. Line 10 of Algorithm 4.3 is executed $O(|\mathcal{A}||\mathcal{O}|)$ times and performs a matrix-vector multiplication at each iteration for an overall running time of $O(|\mathcal{A}||\mathcal{O}||\mathcal{S}|^2)$. Line 11 is executed $O(|\mathcal{A}||\mathcal{O}|)$ times and computes $|\Gamma|$ inner products at each iteration for an overall running time of $O(|\mathcal{A}||\mathcal{O}||\Gamma||\mathcal{S}|)$. Line 12 is executed $O(|\mathcal{A}|)$ times and performs $|\mathcal{O}|$ matrix-vector multiplications at each iteration for an overall running time of $O(|\mathcal{A}||\mathcal{O}||\mathcal{S}|^2)$. These figures are summarized in Table 4.1.

4.4.1 Compressed Data Structures

Our second low-level implementation, called COMPRESSED, straightforwardly uses sparse data structures to improve update efficiency for sparse problems. To analyze this implementation we need some quantitative assumptions about persistent sparsity, namely:

(S1) The fill-in ratio for the T^a matrices is $\rho \ll 1$. In particular, each column of T^a has at most $\rho|\mathcal{S}|$ non-zeros.

(S2) The fill-in ratio for reachable beliefs b is also ρ, meaning each belief has at most $\rho|\mathcal{S}|$ non-zeros.

However, it turns out that the assumption of persistent sparsity is not sufficient to guarantee the sparsity of the α and β vectors generated by ADDPLANE. This is because in the line

$$\beta^a \leftarrow r^a + \gamma \sum_o T^{aT}(\omega^{ao} * \beta^{ao}), \qquad \text{[Line 12 of Algorithm 4.3]} \qquad (4.35)$$

the vector r^a is not guaranteed to be sparse. In particular, it is common to formulate POMDPs such that all actions outside a goal state incur a cost, which guarantees that r^a is dense.

Even if r^a is sparse, the non-zero entries of the various vectors in the sum are likely not to be aligned, so that over multiple applications of `update` the α vectors become progressively less sparse. Therefore our complexity analysis is based on the worst-case assumption that α vectors are dense.

To take advantage of whatever sparsity is available, COMPRESSED stores T^a matrices in compressed sparse column (CSC) mode, and beliefs b and vectors α and β in compressed mode. With this representation, inner products like $\alpha^T b^{ao}$ require $O(\rho|\mathcal{S}|)$ time, and matrix-vector multiplications like $T^a b$ require $O(\rho^2|\mathcal{S}|^2)$ time. As a result, matrix-vector multiplications in the form $T^{aT}\beta$ involve a non-sparse vector and require $O(\rho|\mathcal{S}|^2)$ time. Running times for lines 10-12 are summarized in Table 4.1.

It should be clear that the DENSE and COMPRESSED implementations generate equivalent value functions, so COMPRESSED is another strong incremental representation.

4.4.2 Alpha Vector Masking (Novel Approach)

Our third low-level implementation, called MASKED, is a novel approach that more thoroughly revises the overall algorithm so that sparse α vectors can be used effectively. The value functions generated by the MASKED are *not* equivalent to those generated by DENSE and COMPRESSED.

The key insight of the MASKED approach is that, in problems with persistent sparsity, most entries of a new dense α vector tend to be irrelevant, so that little is lost by discarding those entries and retaining a sparse representation, as long as some steps are taken to ensure that the resulting algorithm still has the properties of a strong point-based update operator.

Recall that each α vector in the max-planes representation is the value function of a particular policy tree. In particular, when a new α vector is generated through a point-based update, it corresponds to a policy tree π that is optimized to perform well if executed starting at belief b. But there is little reason to expect that π and α will perform well for another belief b' unless it is somehow related to b.

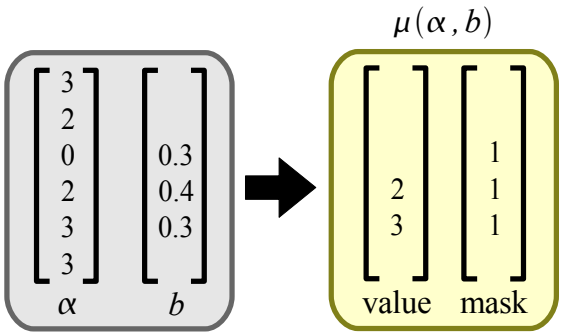


Figure 4.2: Converting an α vector to masked form.

To take advantage of this line of reasoning, we can annotate each α vector with additional information about the belief b that generated it. Then, when looking for an α vector to maximize $\alpha^T b'$ for another belief b', we can reject α from consideration if we judge that b and b' are not closely enough related. The rejection is justified by the intuition that α is unlikely to be optimal at b', although this is not guaranteed to be the case.

In particular, we annotate each α vector with a mask that records which entries of b are non-zeros, and we consider the inner product $\alpha^T b'$ only if the non-zeros of b' "align with the mask" in the sense that all non-zeros of b' are also non-zeros in the mask.

We call the annotated data structure for each α vector the *masked representation*, and conversion to the masked representation is denoted with the operator μ. The data structure $\mu(\alpha, b)$ is composed of two vectors stored in compressed mode. The first is the binary-valued *mask vector*, which records which entries of b are non-zeros. The second is the real-valued *value vector*, which records only the non-zero values of α that align with the mask. Figure 4.2 shows an example of conversion to the masked representation.

To formalize the notion of rejection, we say that a masked vector α *supports* a belief b' if the non-zeros of b' align with its mask. Likewise, the set of all beliefs supported by α is called the *support* of α. Note that the support of a masked vector is always either the whole belief simplex or some hyperface that lies on its boundary, defined by the intersection of constraints in the form $b'(s) = 0$. If all entries in the mask are non-zeros, we say that α is a *full-support vector*.

After converting to the masked representation, we can determine whether α supports a belief b' using the mask vector, and if this check is passed we can calculate $\alpha^T b'$ using its value vector. Note that if α supports b' the inner product sum

Algorithm 4.4 MASKED, an alternate implementation of ADDPLANE.

```
1:  function MASKED.evaluate(Γ, b) :
2:      return max_{α∈σ(Γ,b)} α^T b
3:
4:  function MASKED.initialValueFunction() :
5:      return blindPolicyInitialize()
6:
7:  function MASKED.update(Γ, b) :
8:      for a ∈ A :
9:          for o ∈ O :
10:             b^{ao} ← ω^{ao} * (T^a b)
11:             β^{ao} ← arg max_{α∈σ(Γ,b^{ao})} α^T b^{ao}
12:         β^a ← μ(r^a + γ Σ_o T^{aT}(ω^{ao} * μ(β^{ao}, b^{ao})), b)
13:     β* ← arg max_{β∈{β^1,...,β^{|A|}}} β^T b
14:     return Γ ∪ {β*}
```

depends only on the non-zeros of α that align with the mask, so we have enough information to calculate the inner product even though we discarded the rest of the original α vector.

It is tempting to think that we could check if α supports b' using the non-zeros of the value vector as an implicit mask, which would eliminate the need to explicitly store the mask vector. Unfortunately, that idea breaks down when α happens to have a zero value that falls within the true mask. In that case, using the value vector as an implicit mask would paradoxically result in the masked vector not supporting the very belief b that generated it.

A set of masked vectors can be used to represent a value function in what we call the *masked max-planes representation*. Let Γ be a set of masked vectors. Define the support filtering operator σ such that

$$\sigma(\Gamma, b) := \{\alpha \in \Gamma \mid \alpha \text{ supports } b\}. \tag{4.36}$$

The J^{MAP} operator defines the evaluation semantics of the MASKED implementation of ADDPLANE:

$$J^{\text{MAP}}\Gamma(b) := \max_{\alpha \in \sigma(\Gamma,b)} \alpha^T b. \tag{4.37}$$

The point-based update operator for the MASKED implementation is denoted H^{MAP}, and defined according to the `update` function in Algorithm 4.4. H^{MAP} essentially

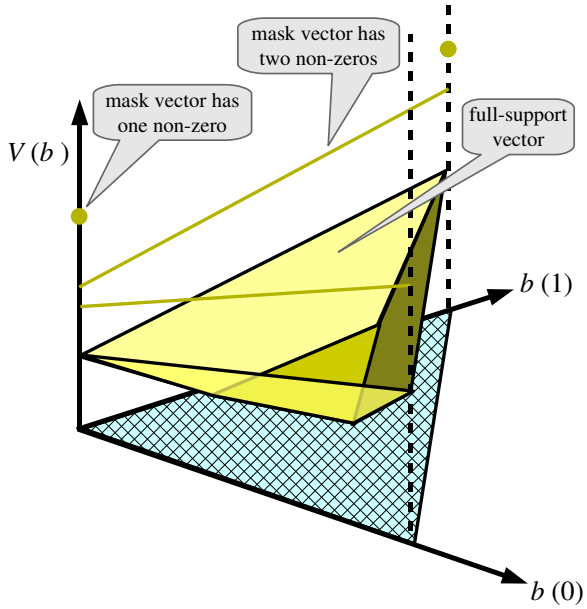


Figure 4.3: The masked max-planes representation.

generates the same α vector as H^{AP}, but converts it to the masked representation before adding it to Γ.

Figure 4.3 shows an example masked max-planes value function for a three-state POMDP. Any masked vector generated at a belief in the interior of the belief simplex supports the entire belief simplex. In the figure these vectors appear as planes. Any masked vector generated on the boundary of the belief simplex supports only the lowest-dimensional boundary hyperface that includes the generating belief. In the figure these vectors appear as lines or points at the edges or corners of the belief simplex. Note that a masked max-planes value function is guaranteed to be convex over the belief simplex, and is even PWLC if one accepts the low-dimensional masked vectors as "linear pieces".

Masked vectors do not provide any advantage if the goal is to exactly represent an optimal value function V_h^* over the entire belief simplex. The smallest such exact representation is always composed solely of full-support α vectors. Rather, masked vectors reduce the memory required to store an approximate representation in cases where accuracy of the lower bound is more important along particular boundaries of the belief simplex than in the interior. Unsurprisingly, as we will see in Chapter 6, accuracy is "more important" in regions of the belief simplex that contain beliefs likely to be reached by good policies.

Algorithm 4.4 presents MASKED, a revision of ADDPLANE that makes efficient use of the masked max-planes value function representation. As with the COMPRESSED low-level implementation, the matrices T^a and beliefs b are stored in compressed mode. The vectors α and β use the masked representation.

Conjecture 4.4. *H^{MAP} is a strong point-based update operator for lower bounds, so the* MASKED *implementation of* ADDPLANE *is a strong incremental representation.*

Proof sketch. We claim that the original operator H_b^{AP} and the masked operator H_b^{MAP} both add the same vector β^* to Γ. The only difference is that the masked update restricts the support of β^* to match the non-zeros of b just before inserting β^* into Γ. Restricting the support in this way makes β^* weaker, so overall the masked update is dominated by the standard point-based update and

$$V \leq H_b^{\text{MAP}} V \leq H_b^{\text{AP}} V \leq HV. \tag{4.38}$$

Since after the restriction β^* is still guaranteed to support b, we also have that

$$H_b^{\text{MAP}} V(b) = H_b^{\text{AP}} V(b) = HV(b). \tag{4.39}$$

Together (4.38) and (4.39) imply that H_b^{MAP} is a strong point-based update.

This argument implicitly assumes that before the update the value function V was composed only of full-support vectors; if it included low-dimensional masked vectors, the result $H_b^{\text{AP}} V$ of the standard point-based update would not be well-defined. A full proof would need to account for that case.

This informal argument is backed by our experience using H^{MAP} to solve benchmark POMDPs; in practice, it appears to be a strong point-based update operator. □

4.4.3 Masked Vector Performance Analysis

In order to show how MASKED improves performance for highly structured problems, we base our complexity analysis on some strong additional assumptions:

(S3) We assume that the problem's persistent sparsity arises because some components of the world state are fully observable. Specifically, we assume the problem state can be decomposed into k binary-valued state variables, of which j are fully observable in the sense that their value is known with certainty for all reachable beliefs. Under this assumption $|\mathcal{S}| = 2^k$ and the problem is persistently sparse with fill-in ratio $\rho = 1/2^j$.

This limited assumption of full observability also has the effect of partitioning $\mathcal{S}$ into 2^j subsets of size $\rho|\mathcal{S}|$, one subset for each possible joint assignment to the fully observable variables. For any reachable belief b, all non-zeros of b fall into a single set of the partition. Therefore all masked vectors α also have masks that fall into a single subset.

(S4) We also assume that the α vectors of Γ are distributed more or less evenly between the subsets in the partition of $\mathcal{S}$, so that at most $O(\rho|\Gamma|)$ of the α vectors fall into each subset.

Whether this assumption holds in practice depends on the problem and the search strategy used to select beliefs for updating. In any case, if the problem is structured so that there are many relevant assignments to the fully observable variables, it is reasonable to expect that only a small fraction of the α vectors will fall into any given subset.

Note that we do not depend on (S1)-(S4) for the correctness of MASKED as a point-based update. These assumptions are used only for the complexity analysis.

The running time of updates using MASKED also depends on the detailed implementation of the support filtering operator σ used at Algorithm 4.4, line 11. To allow efficient filtering, we maintain along with Γ an additional data structure, an array of *support lists* that is indexed by state. The support list for a given state s is a list of pointers to the vectors in Γ whose masks have a non-zero at s. Storing the support lists requires $O(|\Gamma|\rho|\mathcal{S}|)$ space, comparable to the space required for Γ itself. Maintaining the support lists as new vectors are generated does not significantly add to the time overhead.

In order to compute $\sigma(\Gamma, b)$ we do the following:

1. Choose s to be the index of the first non-zero entry in the compressed representation of b.

2. The first stage of the filter provisionally accepts vectors that are referenced in the support list for s. These accepted vectors α are exactly the ones that satisfy $\alpha(s) \neq 0$, which is necessary but not sufficient for α supporting b. Thus there may be some "false positives" in this list, requiring a second stage of filtering. Assumption (S4) implies that $O(\rho|\Gamma|)$ vectors pass the first stage.

3. For each remaining vector α, check if α supports b by comparing the non-zeros of b and the mask for α. This stage of the filter requires $O(\rho|\mathcal{S}|)$ time per vector, or $O(\rho|\Gamma|\rho|\mathcal{S}|)$ time overall.

	Update time	Γ storage space
DENSE	$\lvert\mathcal{A}\rvert\lvert\mathcal{O}\rvert\ (\lvert\Gamma\rvert\lvert\mathcal{S}\rvert + \lvert\mathcal{S}\rvert^2)$	$\lvert\Gamma\rvert\lvert\mathcal{S}\rvert$
COMPRESSED	$\lvert\mathcal{A}\rvert\lvert\mathcal{O}\rvert\ (\lvert\Gamma\rvert\lvert\mathcal{S}\rvert + \lvert\mathcal{S}\rvert^2)\ \rho$	$\lvert\Gamma\rvert\lvert\mathcal{S}\rvert$
MASKED	$\lvert\mathcal{A}\rvert\lvert\mathcal{O}\rvert\ (\lvert\Gamma\rvert\lvert\mathcal{S}\rvert + \lvert\mathcal{S}\rvert^2)\ \rho^2$	$\lvert\Gamma\rvert\lvert\mathcal{S}\rvert\rho$

Table 4.2: Asymptotic complexity of different implementations of ADDPLANE.

Note that, although for convenience we define the filter σ as a mapping on masked vector sets, in practice we never explicitly construct its result set. Instead, we generate and use the result elements one at a time in lazy fashion.

We should also clarify that our blind policy initialization works with α vectors in dense storage mode regardless of which low-level implementation we use for ADDPLANE; if the MASKED implementation is subsequently used, the resulting dense vectors are converted to full-support masked vectors. The presence of these full-support vectors in Γ ensures that throughout the value iteration process every belief b is supported by at least one vector. Since there are at most $|\mathcal{A}| \ll |\Gamma|$ such full-support vectors, we neglect their impact on the update time.

Using MASKED, inner products like $\alpha^T b^{ao}$ require $O(\rho|\mathcal{S}|)$ time, and matrix-vector multiplications like $T^a b$ and $T^{aT}\beta$ require $O(\rho^2|\mathcal{S}|^2)$ time. Line 11 is executed $O(|\mathcal{A}||\mathcal{O}|)$ times. Each time it is executed, the filtering operation $\sigma(\Gamma, b^{ao})$ requires $O(\rho|\Gamma|\rho|\mathcal{S}|)$ time, and the inner product $\alpha^T b^{ao}$ is computed $\rho|\Gamma|$ times, which also requires $O(\rho|\Gamma|\rho|\mathcal{S}|)$ time. Thus the overall running time for line 11 is $O(|\mathcal{A}||\mathcal{O}|\rho|\Gamma|\rho|\mathcal{S}|)$. Running times for lines 10-12 are summarized in Table 4.1.

4.4.4 Complexity Comparison

Table 4.2 summarizes the overall asymptotic complexity of the three implementations of ADDPLANE under the problem structure assumptions (S1)-(S4). The update time field is the result of adding the overall running times of lines 10-12 from Table 4.1.

Relative to DENSE, COMPRESSED updates are faster by a factor of ρ but the same storage space is required for Γ. MASKED updates, on the other hand, are faster by a factor of ρ^2, and storage space for Γ is reduced by a factor of ρ—but the generated α vectors are weaker in the sense that they do not support the entire belief simplex. Our theoretical analysis does not allow us to predict whether this trade-off is worthwhile, as other factors, including the choice of search algorithm, come into play. The trade-off is studied empirically in §4.8.

4.5 Pruning the Max-Planes Representation

Many of the α vectors generated by point-based updates are useful in the sense that they significantly improve the value function at some belief of interest. But vectors generated early in the focused value iteration process often become less useful as later vectors come to dominate them. The time and space complexity of later updates can often be drastically reduced by pruning these less useful older vectors from the max-planes representation.

A pruning algorithm takes as input a vector set Γ and returns a subset $\Gamma' \subseteq \Gamma$. The algorithm is *sound* if, for a given error bound $\epsilon \geq 0$ and region of interest $\tilde{\mathcal{B}} \subseteq \mathcal{B}$, it guarantees that

$$\sup_{b \in \tilde{\mathcal{B}}} |J\Gamma(b) - J\Gamma'(b)| \leq \epsilon. \tag{4.40}$$

Note that most treatments of pruning discuss only the exact case with error bound $\epsilon = 0$. Setting $\epsilon > 0$ allows the pruning algorithm to eliminate vectors that are almost dominated. In practice, we set ϵ just large enough to prune "false duplicate" α vectors introduced by round-off errors in the update process.

We say that a pruning algorithm is *optimal* if it always returns the minimum-size subset Γ' that is sound. It is *locally optimal* if it returns a set Γ' such that no subset of Γ' is sound. Optimality implies local optimality, and the two properties are equivalent if $\epsilon = 0$, but in general are not if $\epsilon > 0$.

The pruning problem has attracted considerable attention due to its strong impact on overall performance. There are at least three basic prior approaches, covered in the following subsections.

Most pruning algorithms are batch operations that apply to the entire vector set. These algorithms can be integrated into the focused value iteration main loop by applying pruning every time the vector set grows by a specified *pruning increment* relative to the size it had immediately after pruning was last applied. Our experiments used this approach for batch pruning operations with a pruning increment of 10%. This setting was frequent enough to avoid much excess growth in the vector set, but not so frequent that there was substantial pruning overhead. Overall performance was not very sensitive to the size of the pruning increment.

Passive bounded pruning (§4.5.4), on the other hand, eliminates α vectors one at a time based on calculations performed during the update process, and does not need explicit pruning episodes.

Algorithm 4.5 PAIRWISEPRUNE, a pruning algorithm for the max-planes representation.

```
1:  function PAIRWISEPRUNE.prune(Γ, ε) :
2:      Γ' ← ∅
3:      for α ∈ Γ :
4:          for β ∈ Γ' :
5:              if β ε-dominates α :
6:                  reject α and skip to next iteration of outer loop
7:          for β ∈ Γ' :
8:              if α ε-dominates β :
9:                  remove β from Γ'
10:         Γ' ← Γ' ∪ {α}
11:     return Γ'
```

4.5.1 Pairwise Pruning (Prior Approach)

Pairwise pruning, sometimes described as "removing pointwise-dominated vectors", is perhaps the simplest useful pruning approach for the max-planes representation. It is tractable and sound but not optimal or locally optimal.

Given a pair of vectors $\alpha, \beta \in \Gamma$, we say that α ϵ-dominates β if for all beliefs b, we have $\alpha^T b \geq \beta^T b - \epsilon$. Checking for ϵ-domination reduces to checking that $\alpha(s) \geq \beta(s) - \epsilon$ for all states s, which requires $O(|\mathcal{S}|)$ time.

One can verify that the soundness condition holds if every vector in Γ is ϵ-dominated by a vector in Γ'. PAIRWISEPRUNE, Algorithm 4.5, builds such a set Γ'. In the worst case it requires a dominance comparison for every pair of vectors, taking $O(|\Gamma|^2|\mathcal{S}|)$ time overall.

Pairwise is not optimal or locally optimal. Sometimes there is a useless vector that is not dominated by any other individual vector, but is dominated by a combination of other vectors. For example, the vector labeled B in Figure 4.4 is dominated, but only by vectors C and E together.

4.5.2 Lark's Filtering Algorithm (Prior Approach)

Lark's filtering algorithm is an alternative pruning approach that is optimal for $\epsilon = 0$, but expensive to compute. The algorithm is to Lark (Littman, 1996).

Whereas pairwise pruning checks whether a new vector α is dominated by any individual vector in Γ', Lark's filtering algorithm, Algorithm 4.6, solves a linear program (LP) which either (1) returns a belief b^* where α is at least ϵ better than

Algorithm 4.6 LARKPRUNE, a pruning algorithm for the max-planes representation.

```
1:  function LARKPRUNE.prune(Γ, ε) :
2:    Δ ← Γ
3:    Γ' ← ∅
4:    while Δ ≠ ∅ :
5:      α ← an arbitrary vector in Δ
6:      [the following line requires solving an LP]
7:      δ, b* ← max, arg max_{b∈B}(α^T b − max_{β∈Γ'} β^T b)
8:      if δ > ε :
9:        β* ← arg max_{β∈Δ} β^T b*
10:       Γ' ← Γ' ∪ {β*}
11:       remove β* from Δ
12:     else:
13:       remove α from Δ
14:   return Γ'
```

all vectors in Γ', or (2) signals that there is no such belief, implying that α is ϵ-dominated.

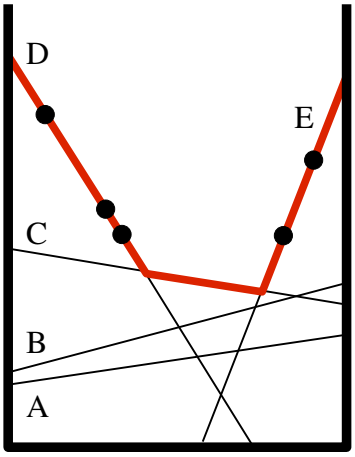
In case (1), a vector needs to be added to Γ' to ensure soundness. Lark's algorithm does not necessarily add α. Instead, it adds the vector β that, among all vectors not yet in Γ', maximizes $\beta^T b^*$.[3] This procedure is intended to ensure that the added vector β will not be dominated by a vector that is considered later in the pruning process.

Lark's algorithm is known to be optimal the case $\epsilon = 0$ (Littman et al., 1996). However, this optimality comes at the price of additional time complexity. Each pruning episode must solve $|\Gamma|$ LPs, each with a constraint matrix of size $O(|\Gamma||\mathcal{S}|)$. Even solving the LPs with interior point methods that run in polynomial time, this is much more expensive than pairwise pruning. Thus Lark's algorithm is preferred to pairwise pruning only for problems over which its superior pruning quality outweighs its greater overhead. This trade-off is so far not well understood.

4.5.3 Bounded Pruning (Prior Approach)

Bounded pruning, Algorithm 4.7, is an alternative approach that works under the assumption that the set $\tilde{\mathcal{B}}$ of beliefs of interest is finite. It is "bounded" in the sense

[3] An appropriate (lexicographic) tie-breaking rule should be used when selecting β, but we will not explain this issue in detail.



Algorithm	Pruned vectors
Pairwise	A
Lark's	A, B
Bounded	A, B, C

Figure 4.4: Pruning results for various algorithms.

that the resulting vector set Γ' is guaranteed to be no larger than $|\tilde{\mathcal{B}}|$.

Over the finite belief set $\tilde{\mathcal{B}}$, bounded pruning is both sound and locally optimal. Its result Γ' is typically smaller than the minimum-size subset that is sound with respect to the whole belief simplex. In this sense it prunes more "aggressively" than Lark's algorithm at the cost of neglecting beliefs outside $\tilde{\mathcal{B}}$.

The running time for bounded pruning is dominated by the time required to calculate inner products. With caching, the inner product $\alpha^T b$ needs to be calculated only once for each $\alpha \in \Gamma$ and each $b \in \tilde{\mathcal{B}}$, so the overall running time is $O(|\Gamma||\tilde{\mathcal{B}}||\mathcal{S}|)$. This complexity is typically intermediate between that of pairwise pruning and Lark's algorithm.

Figure 4.4 compares the results of the various pruning approaches on an example vector set Γ. Thin lines represent individual α vectors. The thicker line indicates the upper surface or maximum of the individual vectors. Points on the upper surface show the locations of beliefs in $\tilde{\mathcal{B}}$. In this example $\epsilon = 0$.

Pairwise pruning prunes vector A, which is the only pointwise dominated vector, dominated by B. Lark's algorithm is more aggressive, pruning both A and B because neither is part of the upper surface. Bounded pruning is most aggressive, pruning A, B, and C because none of them is maximal for any of the points in $\tilde{\mathcal{B}}$.

4.5.4 Passive Bounded Pruning (Novel Approach)

Bounded pruning seems like a natural fit with focused value iteration, which is already designed to use a search algorithm to focus on particularly important beliefs.

Algorithm 4.7 BOUNDEDPRUNE, a pruning algorithm for the max-planes representation.

```
1: function BOUNDEDPRUNE.prune(Γ, B̃, ε) :
2:     for α ∈ Γ :
3:         refCount(α) ← 0
4:     for b ∈ B̃ :
5:         α* ← arg max_{α∈Γ} α^T b
6:         increment refCount(α*)
7:     return {α ∈ Γ | refCount(α) > 0}
```

We can use the choices of the search algorithm as a guide for generating the belief set $\tilde{\mathcal{B}}$ for bounded pruning. In particular, we set $\tilde{\mathcal{B}}$ to be the set containing (1) all points updated so far, and (2) all the immediate successors b^{ao} for updated points b.

However, when using our best search algorithms and pruning periodically, we find that this set $\tilde{\mathcal{B}}$ grows much faster than Γ, so the running time $O(|\Gamma||\tilde{\mathcal{B}}||\mathcal{S}|)$ of bounded pruning quickly outstrips the running time $O(|\Gamma|^2|\mathcal{S}|)$ of pairwise pruning. Ideally, we would like to find a pruning approach that combines the speed of pairwise pruning with the aggressiveness of bounded pruning.

These insights led to a novel pruning approach called *passive bounded pruning* that runs "in the background" during the focused value iteration process. Bounded pruning would be easy if for every belief $b \in \tilde{\mathcal{B}}$ we knew the vector $\alpha \in \Gamma$ that maximized $\alpha^T b$. But recall that inner products in the form $\alpha^T b$ are being calculated constantly as part of the process of performing point-based updates. We can make dual use of those computations to speed pruning by caching them efficiently. We take the approach of persistently caching three pieces of information for each belief:

1. bestValue(b) is the largest value of the inner product $\alpha^T b$ that has been encountered so far during the course of evaluating inner products for point-based updates.

2. bestAlpha(b) is a pointer to the vector α^* that achieves the inner product bestValue(b).

We also store one piece of information for each α vector:

1. refCount(α) is a count of the number of beliefs b such that bestAlpha$(b) = \alpha$.

Passive bounded pruning is triggered by changes to the cached data that occur in the process of calculating point-based updates. The bestValue(b) and bestAlpha(b) fields are used to determine when the refCount(α) fields should be incremented or decremented. If refCount(α) drops to zero for a particular α vector, it is pruned from Γ.

The relationship to reference counting makes it easy to verify that, like standard bounded pruning, passive bounded pruning limits the growth of $|\Gamma|$ so that it remains bounded by $|\tilde{\mathcal{B}}|$. Furthermore, because it does not need to calculate any inner products of its own, it has extremely low overhead.

However, because passive bounded pruning uses cached data that may not be up to date, we obtain weaker soundness guarantees about the resulting vector set. In particular, it is possible for a vector α to be pruned even if it is still useful; that is, even if there is a belief $b^* \in \tilde{\mathcal{B}}$ such that $\alpha^T b^* \geq J\Gamma'(b) + \epsilon$. This can happen if α is added to Γ after b^* is added to $\tilde{\mathcal{B}}$, and the inner product $\alpha^T b^*$ is never evaluated during point-based updates up to the time when the reference count for α drops to zero.

This flaw is not as significant as it might at first seem. The soundness guarantees that passive bounded pruning *does* provide are:

1. The cached value bestValue(b) never decreases for any $b \in \tilde{\mathcal{B}}$.

2. The cached value bestValue(b) is always at least as large as $V(b)$ was just after the most recent point-based update at b. This is because when a new α vector is generated at b, the inner product $\alpha^T b$ is calculated (and cached).

3. There is always a vector $\alpha \in \Gamma$ such that $\alpha^T b \geq$ bestValue(b).

Thus, although it is possible for useful vectors to be pruned by passive bounded pruning, this loss of information does not have the flavor of reversing the local effect of a point-based update. Rather, much like the use of masked vectors, it weakens the ability of a vector α to propagate improvement from its generating belief b to other beliefs b', because α might be pruned before the cache is updated to reflect the fact that α is dominant at b'. And note that a vector α can be pruned only if a new vector β comes to dominate α at its generating belief b. In that case we would expect β to dominate α at b' as well, although this is not guaranteed to be the case.

Another drawback of passive bounded pruning is that it is less aggressive than standard bounded pruning. If there are "inactive" beliefs $b \in \tilde{\mathcal{B}}$ that are not updated or evaluated for a long time, then dominated α vectors that are referenced by those beliefs remain in Γ.

4.5.5 Combined Passive+Pairwise Pruning

Because passive bounded pruning fails to prune α vectors referenced by inactive beliefs, it is possible to reduce $|\Gamma|$ still further using other pruning algorithms with different dominance criteria. For this reason we complement passive bounded pruning with periodic episodes of pairwise pruning.

Note that with focused value iteration the sets $\tilde{\mathcal{B}}$ and Γ are constructed incrementally, so that many of the same beliefs and vectors are present across several pruning episodes. The combined algorithm speeds up pairwise pruning by using additional per-belief cached information:

1. mtime(b) is the "modification time" when bestAlpha(b) was last updated to equal $\arg\max_{\alpha\in\Gamma} \alpha^T b$. This time stamp will be used only for determining the order of various changes to the cache, so it could just as well record the value of a global counter rather than actual time information.

And additional per-α vector information:

1. backPointers(α) is a set of pointers to beliefs b such that bestAlpha$(b) = \alpha$. (The size of this set is equal to refCount(α).)

2. ctime(α) is the "creation time", recording the time of the point-based update that generated α.

In order to work with passive bounded pruning, the standard pairwise pruning algorithm must be slightly modified to perform some additional cache maintenance. Specifically, whenever a vector α is rejected from inclusion in Γ' due to being ϵ-dominated by another vector β, every belief b that references α as its best vector is updated to reference β instead, and bestValue(b) is set to $\beta^T b$. This maintenance can be performed efficiently using the backPointers(α) information in the cache.

We trigger an episode of pairwise pruning every time $|\Gamma|$ grows by 10%. At this frequency pruning empirically takes only a small fraction of the overall running time, and there is not much to be gained by increasing the frequency.

The time stamps allow us to avoid redundant computation in the following ways:

1. If for a particular belief b and vector α, the best vector for b has been recalculated since α was generated, then we are guaranteed that bestValue$(b) \geq \alpha^T b$. This is useful when searching for the maximal vector at b; we can use the guarantee to reject an older vector α without recalculating $\alpha^T b$. In this way, we use the cache to speed not just pruning but also point-based updates.

2. If there are two vectors in Γ that both have creation times predating the last pairwise pruning episode, then we are guaranteed that neither ϵ-dominates the other; otherwise one of them would have been pruned. We can use this fact to avoid performing a dominance check at the next pairwise pruning episode.

For sparse problems, both passive bounded pruning and pairwise pruning can be modified to take advantage of compressed or masked data structures. The optimizations involved are much like those for the point-based update; as such we will not discuss them in detail.

Our treatment of pruning is unusual in that we explicitly define an approximation bound ϵ for the pruning process. In practice we choose ϵ to be on the order of the round-off error we expect from other parts of the algorithm. This helps to eliminate vectors that appear to be useful only due to round-off. For example, it sometimes happens that exact updates at two different beliefs should yield the same α vector, but due to round-off the actual updates generate two slightly different vectors. In that case using an ϵ-domination criterion allows us to prune one of the vectors. It is our view that the round-off error encountered in real implementations tends to moot questions of tie-breaking that have often been discussed in the prior work on pruning.

Unfortunately, the pruning process is currently a gap in our theoretical understanding of focused value iteration. Our arguments for convergence and validity of bounds rely on maintaining the invariant that the bounds are uniformly improvable, but the passive+pairwise pruning approach described in this section does not preserve uniform improvability. Nonetheless we find empirically that lower bounds V^L produced with passive+pairwise pruning, like uniformly improvable bounds, satisfy the key property that the one-step lookahead policy PV^L achieves at least the expected value specified by V^L.

We conjecture that this property is guaranteed, at least approximately within an error bound related to ϵ. Proving this would involve formalizing the idea that passive+pairwise pruning preserves uniform improvability but only "within ϵ" and "over the set $\tilde{\mathcal{B}}$". Until such an extended analysis is in hand, practitioners who are particularly concerned about quality guarantees may wish to avoid bounded pruning and set $\epsilon = 0$.

4.6 Upper Bound Representation

The max-planes representation is not well suited to representing the upper bound in focused value iteration. We incrementally update the max-planes vector set by adding a new α vector, which has the effect of increasing some part of the value

function. In the case of the lower bound V^L such an increase is an improvement, bringing the bound closer to V^*. In the case of the upper bound V^U, such an increase would actually make the bound weaker.

4.6.1 Convex Hull Projection (Prior Approach)

The idea of using a set of belief/value pairs together with convexity to upper bound V^* is implicit in certain linear interpolation approaches (Lovejoy, 1991). Hauskrecht (2000) proposed an upper bound based on convex hull projection, with incremental addition of new belief/value pairs.

This section situates Hauskrecht's approach in our theoretical framework as CONVEXHULL, a strong incremental representation. CONVEXHULL is defined as follows:

- A value function is represented using a set Υ of belief/value pairs.
- The evaluation semantics are defined by the operator J^{CH}, which projects a belief onto the convex hull of the pairs in Υ (explained in detail later).
- The initial uniformly improvable bound Υ_0 is generated using the Fast Informed Bound method.
- The CONVEXHULL point-based update operator, H_b^{CH}, adds the pair $(b, HV(b))$ to Υ.

Now we go into more detail. A *point set* is a finite set of belief/value pairs

$$\Upsilon = \{(b^1, v^1), \ldots, (b^n, v^n)\}. \tag{4.41}$$

Each pair $(b^i, v^i) \in \Upsilon$ is interpreted as a constraint of the form

$$V^*(b^i) \leq v^i. \tag{4.42}$$

This interpretation immediately provides an upper bound on the values $V^*(b^i)$ of beliefs that appear in Υ. In order to upper bound the values of other beliefs, we rely on the convexity of V^*. Convexity implies that if b is a convex combination of beliefs that appear in Υ, $V^*(b)$ is upper bounded by the same convex combination of the corresponding values in Υ. Note that, if the implied upper bound to be finite over the entire belief simplex, the beliefs at the corners of the simplex must be present in the point set.

Let Y be the matrix whose columns are the beliefs b^i of Υ, and let v be the column vector whose entries are the corresponding values v^i. Let u be a length-$|\Upsilon|$ vector of positive weights that sum to one. Then the convexity of V^* implies

that if $Yu = b$,

$$V^*(b) \leq u^T v. \tag{4.43}$$

In order to generate the strongest possible upper bound value for any particular belief b, we need to find the vector u that minimizes $u^T v$. If Υ is a point set and $V = J^{\text{CH}}\Upsilon$ is the value function it represents, we define

$$V(b) = J^{\text{CH}}\Upsilon(b) := u^T v, \tag{4.44}$$

where u is a solution to the LP

$$\begin{array}{l} \text{minimize } u^T v \text{ with decision variable } u \\ \text{subject to} \\ \quad \forall i,\ u(i) \geq 0 \\ \quad Yu = b \end{array} \tag{4.45}$$

Note that we have omitted the constraint that the vector u of weights sums to one. That is because it is redundant with the constraint $Yu = b$ under the assumption that both b and the beliefs of Υ are valid probability distributions. Also note that the LP is well defined even if the entries of b do not sum to 1; scaling b simply scales the result accordingly, so that if $\phi > 0$ is a scalar,

$$J^{\text{CH}}\Upsilon(\phi b) = \phi J^{\text{CH}}\Upsilon(b). \tag{4.46}$$

The LP is guaranteed to be feasible and bounded as long as the corner points of the belief space are found in Υ. Figure 4.5 shows a geometric interpretation of $J^{\text{CH}}\Upsilon$ for a two-state POMDP. Evaluating the upper bound at a belief b can be interpreted as projecting b onto the convex hull of the belief/value pairs in Υ. Circles in the figure indicate belief/value pairs. Thick lines show the implied convex hull. The open circle marks the projection of a particular belief b onto the convex hull.

A value function represented as a point set can be incrementally improved by adding a point to the set. Specifically, let V be a value function with representation Υ. Then the convex hull update operator H_b^{CH} is defined such that

$$H_b^{\text{CH}} V := J^{\text{CH}}\Upsilon', \text{ where} \tag{4.47}$$

$$\Upsilon' := \Upsilon \cup \{(b, HV(b))\}. \tag{4.48}$$

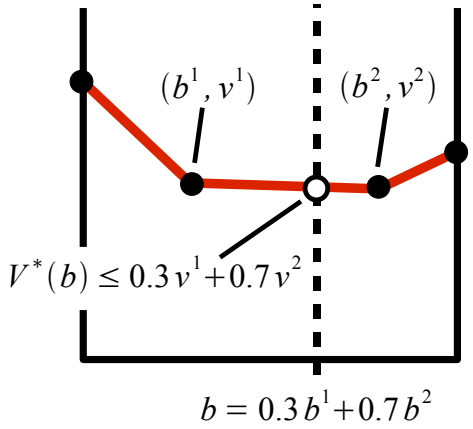


Figure 4.5: The convex hull representation.

$HV(b)$ is calculated using the formula

$$HV(b) = \max_a \left[\tilde{R}(b,a) + \sum_o \Pr(o \mid b,a) V(b^{ao}) \right] \tag{4.49}$$

$$= \max_a \left[r^{aT} b + \sum_o \Pr(o \mid b,a) J^{\text{CH}} \Upsilon(b^{ao}) \right] \tag{4.50}$$

$$= \max_a \left[r^{aT} b + \sum_o J^{\text{CH}} \Upsilon(\Pr(o \mid b,a) b^{ao}) \right] \tag{4.51}$$

$$= \max_a \left[r^{aT} b + \sum_o J^{\text{CH}} \Upsilon(\omega^{ao} * (T^a b)) \right]. \tag{4.52}$$

In Figure 4.6, we see V, $H_b^{\text{CH}} V$, and HV at left, middle, and right, respectively. $H_b^{\text{CH}} V$ is constructed by combining (1) all points in Υ, shown as filled circles, and (2) the single point $(b, HV(b))$ from HV, shown as an open circle. The resulting value function $H_b^{\text{CH}} V$ is drawn with a solid line in the middle panel.

CONVEXHULL, Algorithm 4.8, implements the rules just described for convex hull projection and updates. In order to use CONVEXHULL in focused value iteration, we need to show that it is a strong incremental representation.

Theorem 4.5. *H^{CH} is a strong point-based update operator for upper bounds.*

Proof. For any value function V, define the canonical point set representation of

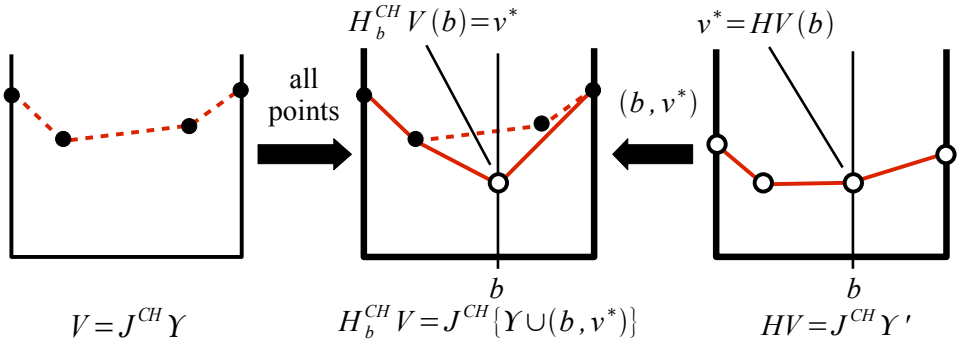


Figure 4.6: A point-based update to the convex hull representation.

V to be the epigraph

$$XV := \{(b, z) \mid b \in \mathcal{B}, z \geq V(b)\}. \tag{4.53}$$

One can verify that the canonical point set representation has the following basic properties:

(C1) For any value functions V, V' we have

$$XV \subseteq XV' \quad \Leftrightarrow \quad V' \leq V. \tag{4.54}$$

(C2) For any point set Ψ we have $\Psi \subseteq X J^{\text{CH}} \Psi$.

(C3) If a value function V is convex, then we have $J^{\text{CH}} XV = V$.

Let Υ be the point set representation for a uniformly improvable value function V^U, and let Υ' be the point set representation of $H_b^{\text{CH}} V^U$. We have

(P1) $HV^U \leq V^U$, because V^U is uniformly improvable.

(P2) $HV^U \leq J^{\text{CH}} \Upsilon$, by (P1) and $V^U = J^{\text{CH}} \Upsilon$.

(P3) $X J^{\text{CH}} \Upsilon \subseteq X HV^U$, by (P2) and (C1).

(P4) $\Upsilon \subseteq X HV^U$, by (P3) and (C2).

(P5) $(b, HV^U(b)) \in X HV^U$, by the definition (4.53) of X.

(P6) $\Upsilon' = \Upsilon \cup \{(b, HV^U(b))\}$, by the definition (4.48) of Υ'.

(P7) $\Upsilon' \subseteq X HV^U$, by (P4)-(P6).

Algorithm 4.8 CONVEXHULL, an incremental representation.

```
1:  function CONVEXHULL.evaluate(Υ, b) :
2:     Y ← the matrix whose columns are the beliefs of Υ
3:     v ← the vector whose entries are the values of Υ
4:     solve the LP :
5:        maximize u^T v with decision variable u
6:        subject to :
7:           ∀i, u(i) ≥ 0
8:           Yu = b
9:     return u^T v
10:
11: function CONVEXHULL.initialValueFunction() :
12:    return fastInformedBoundInitialize()
13:
14: function CONVEXHULL.update(Υ, b) :
15:    v* ← max_a [r^{aT} b + γ Σ_o evaluate(Υ, ω^{ao} * (T^a b))]
16:    return Υ ∪ (b, v*)
```

(P8) $J^{\text{CH}} X H V^U \leq J^{\text{CH}} \Upsilon'$, by (P7) and (C1).

(P9) $HV^U \leq J^{\text{CH}} \Upsilon'$, by (P3), (C3), and the convexity of HV^U.

(P10) $HV^U \leq H_b^{\text{CH}} V^U$, by (P9) and the definition $H_b^{\text{CH}} V^U = J^{\text{CH}} \Upsilon'$.

(P11) $H_b^{\text{CH}} V^U \leq V^U$, because adding points to the point set can only reduce the resulting function.

(P12) $H_b^{\text{CH}} V^U(b) = HV(b)$, because $(b, HV(b))$ is in the point set representation Υ' of $H_b^{\text{CH}} V^U$.

Together (P10)-(P12) establish that H_b^{CH} is a strong update operator. □

4.6.2 Sawtooth Projection (Prior Approach)

This section presents SAWTOOTH, a particularly efficient strong updatable representation based on the same point set data structure used by CONVEXHULL (Hauskrecht, 2000). SAWTOOTH uses approximate linear programming and leverages the special structure of the belief simplex to improve efficiency. Relative to CONVEXHULL with the same point set, SAWTOOTH provides a weaker bound but much faster function evaluations.

The point set representation with CONVEXHULL supports incremental updating, but each function evaluation requires solving a large linear program, which is computationally expensive. To speed up function evaluation, we can calculate a weaker bound through approximate linear programming—any feasible value for the decision variable u provides a valid upper bound. In general, if we fall back from using an exact LP solution algorithm to an approximate algorithm, we find that:

- The resulting updatable representation is conservative if (1) the approximate LP algorithm is guaranteed to return a feasible value for u, and (2) adding new points to the point set never increases the value $u^T v$ for any query point b.

- The resulting updatable representation is strong if the approximate LP algorithm has the additional property that when (b^i, v^i) is in the point set and b^i is the query point, the result u always satisfies

$$u^T v \leq v^i. \tag{4.55}$$

The approximate LP algorithm used by the SAWTOOTH representation satisfies both properties. It simplifies solving the LP by adding additional constraints. In true projection onto the convex hull, the weight vector u can assign non-zero weight to any combination of beliefs. In the sawtooth projection, u is constrained to assign non-zero weight to (1) any number of beliefs at the corners of the belief simplex, and (2) at most one non-corner belief.

Figure 4.7 shows the distinctive shape that gives the sawtooth value function its name. Each thin line shows the constraint induced by an individual non-corner belief/value pair in conjunction with the corners. The lower surface of these constraints is the sawtooth bound, indicated with the thick solid lines. The dashed lines show the stronger bound that would be provided by the convex hull projection for the same point set.

Because the sawtooth projection distinguishes corners from other beliefs, we represent the point set in a slightly different way. In particular, we define the set $\mathcal{Y}$ and the length-$|\mathcal{S}|$ vector w such that

$$\mathcal{Y} := \{(b^i, v^i) \in \Upsilon \mid b^i \text{ is not a corner point}\} \tag{4.56}$$

$$w(s) := \min\{v^i \mid (b^i, v^i) \in \Upsilon, b^i(s) = 1\}. \tag{4.57}$$

In other words, $\mathcal{Y}$ stores the information about Υ that relates to non-corner beliefs, and w stores the v^i values for the $|\mathcal{S}|$ corner beliefs.

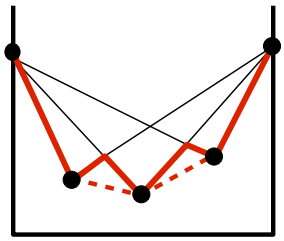


Figure 4.7: The sawtooth representation.

To derive the formula for the sawtooth projection, first note that any belief b can be trivially expressed as a convex combination of the corner beliefs. Since the corner beliefs are simply unit vectors in the basis directions, each entry $u(s)$ in the weight vector for the convex combination must be equal to $b(s)$. The resulting bound due to convexity is

$$V^*(b) \leq w^T b. \tag{4.58}$$

Next we consider combinations involving one non-corner belief b^i from Υ. Suppose that for some $\phi \geq 0$ and $c \in \mathcal{B}$, we have

$$b = \phi b^i + c. \tag{4.59}$$

Then, due to convexity,

$$V^*(b) \leq \phi V^*(b^i) + V^*(c). \tag{4.60}$$

But $V^*(b^i) \leq v^i$ and $V^*(c) \leq w^T c$, so

$$\begin{aligned} V^*(b) &\leq \phi v^i + w^T c && (4.61)\\ &\leq \phi v^i + w^T(b - \phi b^i) && (4.62)\\ &\leq w^T b + \phi(v^i - w^T b^i). && (4.63) \end{aligned}$$

We know the values of all the variables in (4.63) except for ϕ. In fact, we are free to choose whatever feasible value for $\phi \geq 0$ gives the best bound.

Since (4.63) is linear in ϕ, it will be minimized at one of the extremal values of ϕ. The extreme $\phi = 0$ can be ignored because it recovers the earlier bound $V^*(b) \leq w^T b$. Thus, in order to get the best bound, we wish to find the maximum value for ϕ that is consistent with our other assumptions. In particular, since we

used the fact that $c \in \mathcal{B}$, all the entries of c must be positive, meaning that for all s,

$$c(s) = b(s) - \phi b^i(s) \geq 0. \tag{4.64}$$

The maximal ϕ that satisfies this set of constraints is

$$\phi = \min\{b(s)/b^i(s) \mid s \in \mathcal{S}, b^i(s) > 0\}. \tag{4.65}$$

If Υ is a point set and $V = J^{\text{ST}}\Upsilon$ is the value function it represents under the sawtooth projection, we define

$$V(b) = J^{\text{ST}}\Upsilon(b) := \min_i x^i, \tag{4.66}$$

where

$$x^0 = w^T b \tag{4.67}$$

$$x^i = w^T b + \phi^i(v^i - w^T b^i), \;\; i \geq 0 \tag{4.68}$$

$$\phi^i = \min\{b(s)/b^i(s) \mid s \in \mathcal{S}, b^i(s) > 0\}. \tag{4.69}$$

Let V be a value function with sawtooth representation Υ. Then the sawtooth point-based update operator H_b^{ST} is defined such that

$$H_b^{\text{ST}} V := J^{\text{ST}}\Upsilon', \text{ where} \tag{4.70}$$

$$\Upsilon' := \Upsilon \cup \{(b, HV(b))\}, \tag{4.71}$$

and $HV(b)$ is calculated using the formula

$$HV(b) = \max_a \left[r^{aT} b + \sum_o J^{\text{ST}}\Upsilon(\omega^{ao} * (T^a b)) \right]. \tag{4.72}$$

SAWTOOTH, Algorithm 4.9, implements the sawtooth projection and point-based update using the modified $(\mathcal{Y}, w)$ point set representation.

Theorem 4.6. *H^{ST} is a strong update operator for upper bounds.*

Proof. The argument is precisely parallel to the proof of Theorem 4.5, replacing all instances of J^{CH} with J^{ST} and all instances of H_b^{CH} with H_b^{ST}. □

The LP for convex hull projection can be solved using interior point methods in polynomial time. Sawtooth projection, on the other hand, takes $O(|\mathcal{S}|)$ time per non-corner belief, or $O(|\mathcal{Y}||\mathcal{S}|)$ time overall. This provides significant speedup in practice.

Algorithm 4.9 SAWTOOTH, an incremental representation (Hauskrecht, 2000).

function SAWTOOTH.evaluate($(\mathcal{Y}, w), b$) **:**
  $x^0 \leftarrow w^T b$
  for $(b^i, v^i) \in \mathcal{Y}$ **:**
    $\phi \leftarrow \min\{b(s)/b^i(s) \mid s \in \mathcal{S}, b^i(s) > 0\}$
    $x^i \leftarrow x^0 + \phi(v^i - w^T b^i)$
  return $\min_i x^i$

function SAWTOOTH.initialValueFunction() **:**
  return fastInformedBoundInitialize()

function SAWTOOTH.update($(\mathcal{Y}, w), b$) **:**
  $v^* \leftarrow \max_a \left[r^{aT} b + \gamma \sum_o \text{evaluate}((\mathcal{Y}, w), \omega^{ao} * (T^a b)) \right]$
  if b is the corner point for some state s **:**
    $w(s) \leftarrow v^*$
    return $(\mathcal{Y}, w)$
  else:
    return $(\mathcal{Y} \cup (b, v^*), w)$

4.6.3 Leveraging of Sparsity with Sawtooth (Novel Approach)

The sawtooth representation can be modified to leverage sparsity. Simply using compressed vectors to represent beliefs reduces the complexity of sawtooth evaluation to $O(|\mathcal{Y}||\mathcal{S}|\rho)$.

Sparsity can be leveraged further using techniques akin to the masked vectors and support lists introduced for the lower bound in §4.4. Under assumptions similar to (S1)-(S4) of §4.4, the complexity of "masked" sawtooth evaluation is $O(|\mathcal{Y}||\mathcal{S}|\rho^2)$. Like masked vectors for the lower bound, this improved efficiency comes at the cost of reducing the support of individual belief/value pairs, providing a weaker bound overall.

4.6.4 Pruning the Sawtooth Representation

Pruning the sawtooth representation is simpler than pruning the max-planes representation. We use the same definitions of validity, optimality, local optimality, and ϵ-domination that we introduced for lower bound pruning in 4.5, but remembering to reverse the direction of inequalities in the case of the upper bound.

A key fact is that adding a point (b^i, v^i) to Υ always has its greatest impact

on $J^{\mathrm{ST}}\Upsilon$ at the point b^i itself. This leads to the following technique for checking pairwise ϵ-domination. Let (b^i, v^i) and (b^j, v^j) be two belief/value pairs. Suppose that in conjunction with the corner points, (b^j, v^j) induces the constraint $V^*(b^i) \leq z$. Then (b^j, v^j) ϵ-dominates (b^i, v^i) if $z \leq v^i + \epsilon$.

This pairwise dominance check can be used to perform pairwise pruning as with the max-planes representation. Pairwise pruning builds up the result set $\mathcal{Y}'$ by iterating through $\mathcal{Y}$ and adding each pair (b^i, v^i) that is not already dominated by some pair in $\mathcal{Y}'$. The corner values in the vector w are not modified by the pruning process.

As with max-planes pairwise pruning, sawtooth pairwise pruning is relatively efficient. Each pairwise dominance check requires $O(|\mathcal{S}|)$ time, and dominance may need to be checked for $O(|\mathcal{Y}|^2)$ pairs of belief/value pairs, for $O(|\mathcal{Y}|^2|\mathcal{S}|)$ time overall.

Furthermore, it turns out that sawtooth pairwise pruning also carries the benefits that we associated with other algorithms in the lower bound case. Specifically:

1. Like Lark's algorithm for the max-planes representation, sawtooth pairwise pruning is locally optimal for any ϵ, optimal for $\epsilon = 0$, and valid over all beliefs in $\mathcal{B}$.

2. Like bounded pruning for the max-planes representation, sawtooth pairwise pruning is aggressive in that it retains only elements of the representation that are needed to bound a restricted subset of the belief simplex—in this case, the corners and the beliefs found in $\mathcal{Y}$. We have reason to believe that the beliefs of $\mathcal{Y}$ are important, because they were chosen for point-based updates by the focused value iteration search algorithm.

Thus, overall, the pruning analysis is easier for the sawtooth representation than it was for the max-planes representation; we can simply use the pairwise pruning algorithm, which wins on all fronts.

4.7 Hybrid Tabular Representations (Novel Approach)

A recurring theme in this chapter has been speeding up function evaluation even at the cost of providing a weaker overall bound. The natural endpoint of this progression is the *tabular representation*, which stores values for a finite number of beliefs and does not attempt to provide a bound for other beliefs. When implemented appropriately, evaluating a belief under the tabular representation requires only $O(|\mathcal{S}|)$ time; no evaluation scheme that depends on all entries of the belief can do better. In the following discussion we use the tabular representation as a

lower bound, but it can also be used as an upper bound simply by changing signs in a few places.

The tabular representation is formally defined as a point set

$$\Upsilon = \{(b^1, v^1), \ldots, (b^n, v^n)\}. \tag{4.73}$$

If $V = J^{\text{Tab}}\Upsilon$ is the value function it represents under the tabular representation, we define

$$V(b) = J^{\text{Tab}}\Upsilon(b) := \begin{cases} v^i & \text{if there is a pair } (b^i, v^i) \in \Upsilon \text{ with } b = b^i \\ -\infty & \text{otherwise.} \end{cases} \tag{4.74}$$

To support efficient evaluation, the point set is stored as a hash table with b^i keys and v^i values. We evaluate the function at a belief b with a hash table lookup. This includes (1) calculating the hash function for b in $O(|\mathcal{S}|)$ time, and (2) checking equality of b with $O(1)$ other beliefs b^i, each check requiring $O(|\mathcal{S}|)$ time. Thus the overall evaluation time is $O(|\mathcal{S}|)$.

Point-based updates to the tabular representation are calculated in much the same way as for other representations based on point sets. Let V be a value function with tabular point set Υ. Then the tabular point-based update operator H_b^{Tab} is defined such that

$$H_b^{\text{Tab}}V := J^{\text{Tab}}\Upsilon', \text{ where} \tag{4.75}$$

$$\Upsilon' := \Upsilon \cup \{(b, HV(b))\}, \tag{4.76}$$

and $HV(b)$ is calculated using tabular function evaluations with the formula

$$HV(b) = \max_a \left[r^{aT} b + \sum_o J^{\text{Tab}}\Upsilon(b^{ao}) \right]. \tag{4.77}$$

The tabular representation is not very useful by itself because it does not provide any bound on the values of beliefs not present in the point set. In practice, we use a tabular value function V_1 in combination with some other function representation V_2 that has full support and acts as a "fallback", so that the hybrid value function is

$$V_{\text{hybrid}}(b) = \begin{cases} V_1(b) & V_1(b) \neq -\infty \\ V_2(b) & \text{otherwise.} \end{cases} \tag{4.78}$$

Point-based updates to the hybrid representation add points only to the tabular representation point set, but the function evaluations used to calculate $HV(b)$ for

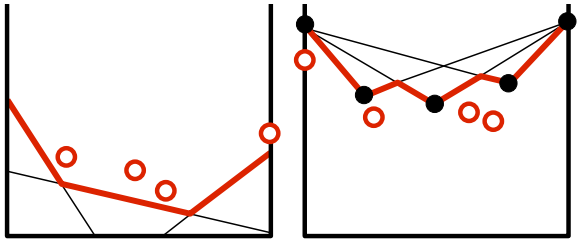


Figure 4.8: Hybrid tabular representations: (left) Hybrid tabular + max-planes lower bound. (right) Hybrid tabular + sawtooth upper bound.

an update make use of the fallback.

Figure 4.8 shows two hybrid representations. The left panel shows a hybrid tabular + max-planes representation for the lower bound. Points in the tabular point set are indicated with open circles above the max-planes fallback. The right panel shows a hybrid tabular + sawtooth representation for the upper bound. Points in the tabular point set are again indicated with open circles, this time below the fallback.

The hybrid representation provides several potential benefits relative to simply using the full support function V_2:

1. If most function evaluations relate to beliefs in the tabular point set, V_1 can in effect act as a cache with cheap function evaluation in front of V_2, which presumably has more expensive function evaluation.

2. Because point-based updates modify only V_1, we are free to use a representation for V_2 that does not support incremental improvement. For example, we could use the max-planes representation for the upper bound.

3. The size of the V_2 representation remains fixed over the course of focused value iteration, bounding the amount of time required for individual function evaluations and eliminating the need for pruning of V_2.

Hybrid tabular representations provide fast function evaluation, and thus fast point-based updates, but each update is not as beneficial because the resulting belief/value pair in the point set supports only the belief that generated it—there is no generalization to other beliefs. In the absence of generalization there is also no obvious way to prune the point set; thus memory requirements become more of a concern. The relative costs and benefits are problem-dependent, and we do not have a general theory for predicting when the trade-off is worthwhile. The question is studied empirically in §4.8.

Practical implementations of the tabular representation should also account for round-off errors in belief updates. When calculating belief updates exactly, it sometimes happens that the same belief can be reached from b_0 through two distinct sequences of actions and observations. In the presence of round-off error, however, the sequences may result in two slightly different beliefs b and b'. We would like for a point-based update at either of these beliefs to affect the other, as would happen without round-off error.

A simple way to achieve this is to resolve each belief to the nearest grid point of a Cartesian grid with resolution ϵ, before passing the belief to the hash table implementation. In order to make it unlikely that two copies b and b' of the "same" belief map to different grid points, we choose ϵ to be significantly larger than the magnitude of the round-off errors we expect. This scheme suffers from two potential problems that are not very important in practice:

1. False positive matches. If b and b' would *not* have been the same in the absence of round-off and yet resolve to the same grid point, a "generalization error" of $|V(b) - V(b')|$ may be introduced. Since b and b' are close neighbors, and the maximum slope of the value function is generally bounded, this error is unlikely to be very significant. Furthermore, informal tests with benchmark problems suggest that false matches are extremely rare relative to the number of true matches if ϵ is chosen appropriately.

2. False negatives. No matter how large ϵ is relative to the distance between b and b', it is always possible for the two points to straddle a grid cell boundary. In this case some generalization is lost, but we are no worse off than before.

4.8 Experimental Performance

We ran performance comparisons for many of the value function representations described in this chapter. Each experiment involved a particular value function representation and POMDP benchmark problem. In the course of the experiments, we calculated an initial uniformly improvable bound and then performed a sequence of point-based updates. At completion we measured the accuracy of the resulting bound, the time required to make the updates, and the storage space required for the representation.

For benchmark problems we selected the *Tag*, *RockSample*[5,7], and *LifeSurvey*1 POMDPs. These are all relatively large sparse problems.

Our experiments compare five lower bound implementations:

1. `comp`: The COMPRESSED implementation of the ADDPLANE point-based update algorithm, with pairwise pruning only.

2. `comp/prune`: The COMPRESSED implementation with more aggressive combined passive+pairwise pruning.

3. `mask`: The MASKED implementation of ADDPLANE, with pairwise pruning only.

4. `mask/prune`: The MASKED implementation with combined passive+pairwise pruning.

5. `tab+comp`: The hybrid tabular representation with `comp` as the fallback function. Note that the fallback value function is not modified during point-based updates, so it does not need to be pruned.

Note that we have omitted detailed comparison with the DENSE implementation of ADDPLANE. Comparison using an earlier version of our software which supported DENSE showed that it was not competitive when applied to sparse problems like the ones in the present experiments. We have not attempted to precisely determine the break-even point where implementations that leverage sparsity become advantageous. We removed support for DENSE from more recent versions of our software because it seemed to be dominated and it was different enough from the other implementations that keeping it in the same code base imposed a high maintenance burden.

To summarize the overall lower bound results, we found that the performance ordering of the various representations, from best to worst, was `mask/prune`, `mask`, `tab+comp`, `comp/prune`, `comp`. The dominance of the `mask` variants was not clear-cut for *Tag*, the smallest problem, but became more pronounced as we scaled up to *LifeSurvey*1.

We also compare three upper bound implementations:

1. `comp`: The SAWTOOTH implementation using compressed storage for belief vectors. Pairwise pruning is used.

2. `mask`: Same as `comp`, but function evaluation uses support lists similar to those used in the MASKED implementation of the lower bound.

3. `tab+comp`: The hybrid tabular representation with `comp` as the fallback function.

Note that we have omitted comparison with the CONVEXHULL upper bound representation, which was not competitive when tested with an earlier version of our software (Smith and Simmons, 2005). More recent versions of our software no longer support the CONVEXHULL representation so as to remove dependence on

	Tag	RS57	LS1
Num. states/actions/observations	870s 5a 30o	3201s 12a 2o	7001s 7a 28o
LB baseline representation	comp	comp	mask
UB baseline representation	comp	comp	comp
LB termination threshold	-14.03	20.047	87.849
UB termination threshold	-0.698	29.08	120.97

Table 4.3: Value function experiments: Problem parameters.

the commercial CPLEX linear program solver, which was used to perform projections onto the convex hull.

We found that the performance ordering of the upper bound representations, from best to worst, was `mask`, `comp`, `tab+comp`. This ordering was clear-cut for all three problems.

For each problem and representation we constructed a sequence of $n = 10^4$ beliefs at which to perform point-based updates. We decided to eliminate one source of variance by using the same baseline sequence across all value function representations. However, we also wanted the experiment to be as relevant as possible to real usage, so we constructed the baseline sequence by running a baseline configuration of focused value iteration.

The choice of baseline configuration was fairly arbitrary. For the search algorithm we selected the high-quality FRTDP algorithm described in §6.4. For the lower and upper bounds, we had a range of available options that traded off strong function generalization vs. fast update time. For most problems, we selected `comp`, the representation with the strongest function generalization, for both bounds. In the case of the *LifeSurvey*1 problem we were not able to perform n `comp` lower-bound updates within a reasonable amount of time, so we used the `mask` representation for the lower bound. The baseline configurations are summarized in Table 4.3 (the termination thresholds will be explained later).

4.8.1 Performance Versus Time Plots

The first set of experiments plots performance vs. wallclock time. Specifically, in terms of performance, we measure:

1. The bound value at the initial belief, either $V^L(b_0)$ or $V^U(b_0)$. Because the beliefs of the baseline sequence were generated by the FRTDP search algorithm, which groups beliefs into "trials", it was natural to record the bound values at the end of each trial. In particular, the first value in each plot was

recorded at the end of the first trial; in some cases, when the representation had relatively slow updates, this was several seconds into the run.

2. The expected value of executing the one-step lookahead policy based on the bound starting from b_0, either $JPV^L(b_0)$ or $JPV^U(b_0)$. The plotted value is the mean value received over 1000 runs in simulation. Because policy evaluation is quite slow, the policy quality was only recorded a few times during each run.

The elapsed wallclock time at the end of each performance vs. time plot roughly corresponds to when `tab+comp`, the representation with the fastest updates, came to the end of the $n = 10^4$ beliefs in the baseline sequence. Wallclock time is measured in seconds. The platform used for these experiments was a Pentium-4 running at 3.4 GHz, with 2 GB of RAM.

When interpreting performance vs. time plots it is helpful to remember that, given our use of uniformly improvable lower bounds V^L, the expected quality $JPV^L(b_0)$ is guaranteed to exceed $V^L(b_0)$ up to statistical error in the quality measurement. Thus, as the lower bound improves, the measured quality of the policy based on the lower bound should also improve. However, there is no corresponding guarantee for the upper bound; the $JPV^U(b_0)$ plots often behave erratically. This is why we recommend using the policy PV^L based on the lower bound during execution.

Tag performance vs. time is shown in Figure 4.9. The upper left panel plots $V^L(b_0)$; higher is better. We see that there is no clear winner, but `tab+comp` lags well behind the other representations. Use of passive bounded pruning has little impact in this case; `comp` and `comp/prune` are nearly identical, as are `mask` and `mask/prune`. The lower left panel plots $JPV^L(b_0)$. Again, `tab+comp` lags well behind. The other representations have nearly identical performance up to the statistical error in the policy quality measurement. The 95% confidence interval was approximately ± 0.4 even after taking the mean over 1000 simulation runs.

The upper right panel plots $V^U(b_0)$; lower is better. Here `mask` clearly performs best, followed by `comp` and `tab+comp` a bit further behind. The lower right panel plots $JPV^U(b_0)$. In this case `tab+comp` is the clear loser, and `mask` and `comp` behave erratically.

RockSample[5,7] performance vs. time is shown in Figure 4.10. Again, `tab+comp` lags behind the other representations in both of the lower bound plots. However, here the `mask` and `mask/prune` representations significantly outperform `comp` and `comp/prune` in the $V^L(b_0)$ plot; there is still no clear winner in the $JPV^L(b_0)$ plot. A possible interpretation of these data is that since the `comp` lower bound provides stronger generalization than `mask`, its corresponding policy

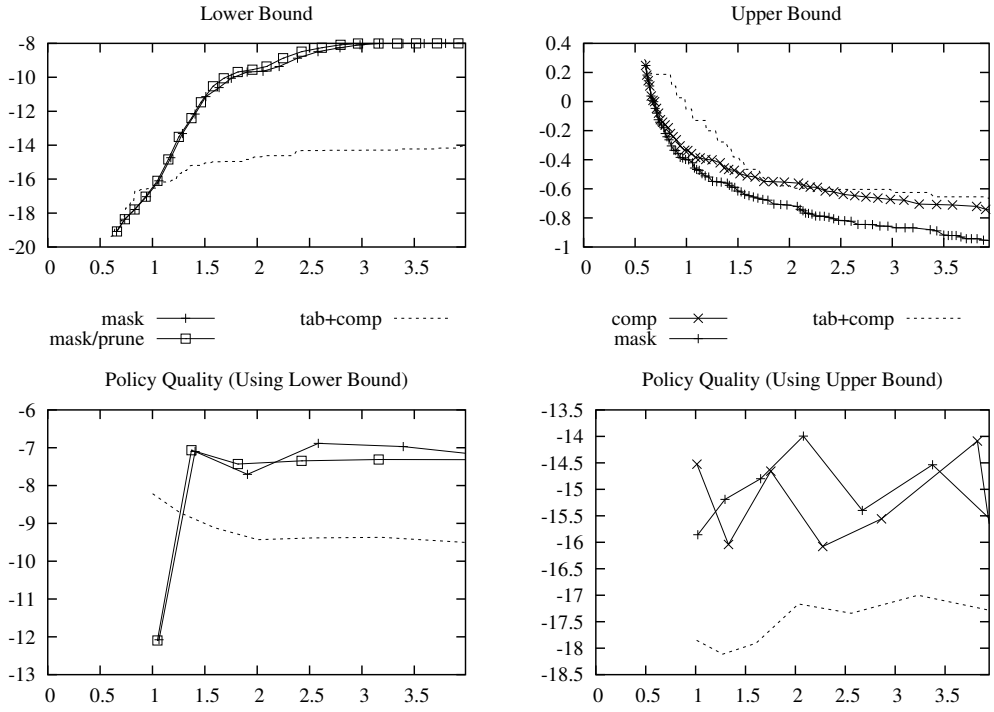


Figure 4.9: *Tag* performance vs. wallclock time (s).

may be more likely to substantially outperform the quality guarantee implied by the bound.

The upper bound plots for *RockSample*[5,7] show the same performance ordering as for *Tag*, with `tab+comp` even further behind the other representations.

*LifeSurvey*1 performance vs. time is shown in Figure 4.11. In this case the `mask` and `mask/prune` lower bounds are the clear winners. The `comp` and `comp/prune` updates are so slow that they do not even finish a single trial before `tab+comp` finishes the baseline sequence; as a result, they do not even appear in the plots.

The $V^U(b_0)$ plots for *LifeSurvey*1 show the same performance ordering as for *Tag*, but performance is so erratic in the $JPV^U(b_0)$ plot that there is no clear ordering at all.

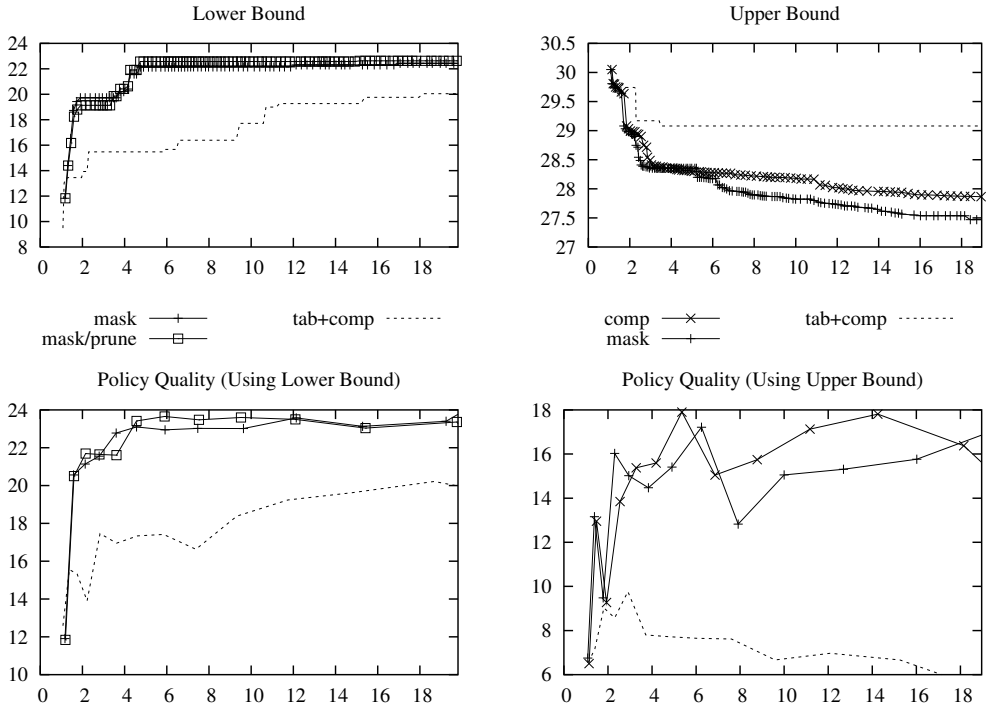


Figure 4.10: *RockSample*[5,7] performance vs. wallclock time (s).

4.8.2 Equal Precision Comparison

In another set of experiments we updated each lower bound representation until it satisfied an "equal precision" criterion. Specifically, we designated a particular bound value as a threshold and terminated the run when $V^L(b_0)$ crossed this threshold. We could then compare the amount of time and space the various representations required to achieve the specified threshold. We performed the same experiments with upper bound representations using a termination threshold for $V^U(b_0)$.

For each problem and each bound type (lower or upper), we selected the precision threshold to be the strongest bound value that could be reached by all of the representations within approximately $n = 10^4$ updates. In practice this was the bound at b_0 calculated by the `tab+comp` representation after n updates, since

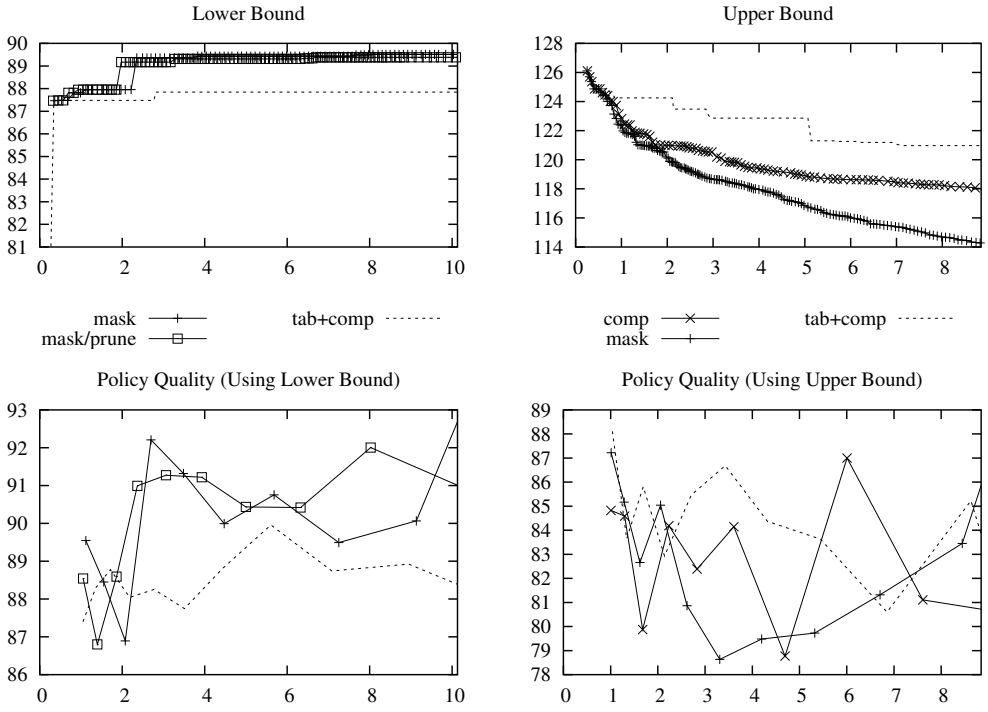


Figure 4.11: *LifeSurvey*1 performance vs. wallclock time (s).

`tab+comp` has the weakest generalization.[4] The termination thresholds we used are given in Table 4.3.

Table 4.4 gives lower bound time performance results for the equal precision experiments. The first set of columns shows the number of point-based updates required to reach the termination threshold bound value; the second set gives the corresponding wallclock time in seconds. The best (smallest) results in each column are in bold type.

As one would expect, the `comp` representation achieves the desired precision with a smaller number of updates, since it has the strongest generalization;

[4]Based on the way the termination thresholds were set, one might expect the number of updates in the `tab+comp` row of Table 4.4 to be approximately $n = 10^4$ for all problems. In fact, it is sometimes much smaller because the $V^L(b_0)$ value for `tab+comp` tends to hold steady at certain "plateau" values for long periods. Thus `tab+comp` may reach the termination threshold early in the baseline sequence but never surpass it before the sequence ends.

	Num. of updates			Wallclock (s)		
	Tag	RS57	LS1	Tag	RS57	LS1
comp	**71**	**171**	**23**	1.31	30.6	27.3
comp/prune	**71**	196	**23**	1.31	26.5	27.2
mask	171	396	146	**1.29**	3.9	1.1
mask/prune	171	396	146	**1.29**	**3.8**	**0.9**
tab+comp	10030	8896	1446	4.06	18.7	2.8

Table 4.4: Lower bound time performance.

`comp/prune` is comparable. However, the updates of `mask` and `mask/prune` can be computed so much faster that these representations outperform `comp` and `comp/prune` on the basis of wallclock time. The `tab+comp` representation has still faster updates, but its overall performance turns out to be slower due to its lack of generalization.

Table 4.5 gives lower bound storage space results. The first set of columns gives the number of α vectors in the vector set Γ for the `comp` and `mask` representations, and the number of points in the tabular point set Υ for the `tab+comp` representation. The second set of columns gives the corresponding storage space required, calculated according to the following rules:

- The space required for an α vector is defined to be the number of non-zeros in the compressed representation.
- For each masked α vector used in the `mask` and `mask/prune` representations, we also add the number of non-zeros in the mask vector.[5]
- For each belief/value pair in the `tab+comp` point set, we count the number of non-zeros in the compressed representation of the belief and add one entry for the value.
- The reported space for the `tab+comp` representation also includes the space required to store the `comp` fallback function.
- In our implementation each "entry" in the above counts corresponds to an 8-byte double-precision floating point value accompanied in most cases by a 4-byte integer index. Thus to upper bound the real memory requirements one can multiply the reported storage space by 12 bytes.

[5]Since mask vector entries are boolean-valued, they could be stored more efficiently than real-valued α vector entries; however, for convenience, our implementation uses the same data structure for both types of vector.

	$\|\Gamma\|$ or $\|\Upsilon\|$			Space		
	Tag	RS57	LS1	Tag	RS57	LS1
comp	64	144	27	54K	461K	189K
comp/prune	**59**	**99**	**22**	50K	317K	154K
mask	121	204	83	32K	**35K**	87K
mask/prune	94	197	63	**25K**	**35K**	**16K**
tab+comp	4467	3583	504	124K	208K	59K

Table 4.5: Lower bound storage space.

	Num. of updates			Wallclock (s)		
	Tag	RS57	LS1	Tag	RS57	LS1
comp	**1507**	**221**	**771**	3.24	1.80	2.05
mask	**1507**	**221**	**771**	**1.83**	**1.70**	**1.62**
tab+comp	1984	1062	7146	3.84	3.68	7.12

Table 4.6: Upper bound time performance.

In this table, we see in the $|\Gamma|$ columns that `comp` and `comp/prune` achieve the termination threshold with fewer vectors, due to the fact that each vector supports the entire belief simplex. Of these two, the `comp/prune` representation requires slightly fewer vectors, which makes sense due to its more aggressive pruning. In the space columns, however, the `mask` and `mask/prune` representations show better performance. Although the masked representations require more α vectors, each vector is so much sparser that the representation requires less overall space. Of the two masked representations `mask/prune` requires fewer vectors and less space due to more aggressive pruning. The `tab+comp` representation lags behind on both metrics.

Table 4.6 gives time performance results for the upper bound representations. The first set of columns shows the number of point-based updates required to reach the termination threshold bound value; the second set gives the corresponding wallclock time in seconds.

In this table the `comp` and `mask` representations require the same number of updates to reach the termination threshold, but the wallclock time measurements show that individual `mask` updates are marginally faster, making it faster overall. The `tab+comp` representation is again weakest both in number of updates and wallclock time, although its individual updates are faster than both of the other algorithms.

Table 4.7 gives upper bound storage space results. The first set of columns

	$\|\Upsilon\|$			Space		
	Tag	RS57	LS1	Tag	RS57	LS1
comp	**952**	**114**	**375**	**47K**	**8.8K**	**7.1K**
mask	**952**	**114**	**375**	**47K**	**8.8K**	**7.1K**
tab+comp	4392	453	1634	118K	28.5K	31.3K
(corners)	870	3201	7001	1K	3.2K	7.0K

Table 4.7: Upper bound storage space.

gives the number of points in the tabular point set Υ. The second set of columns gives the corresponding storage space required, calculated according to the following rules:

- For each belief/value pair in the point set, we count the number of entries in the compressed representation of the belief and add one entry for the value.
- All three representations store the vector w of corner belief values in the same way, as a vector in dense storage mode with one value entry for each corner. The space required to store w is not included in the measurements for the individual representations. Instead, it is called out as a separate row of the table, marked `(corners)`.
- As in the lower bound case, one can upper bound the real memory requirements by multiplying each entry count by 12 bytes.

In this table the `comp` and `mask` upper bound representations show identical storage requirements with the `tab+comp` representation lagging behind as usual.

4.8.3 Error Distributions

Our final set of experiments studied bound precision more carefully. The experiments presented so far have measured bound error only at the initial belief, either $V^L(b_0)$ or $V^U(b_0)$. Here we quantify bound error over a broader distribution of beliefs. In this way we can more directly compare the amount of generalization provided by different representations.

We define the error of an approximate value function $\hat{V}$ to be the random variable

$$\delta(b) := \hat{V}(b) - V^*(b), \tag{4.79}$$

where beliefs b are drawn from some belief distribution A. We hope to get insight into the quality of the approximation $\hat{V}$ by plotting the distribution of δ.

We constructed an approximation $\hat{V}$ for each representation by performing point-based updates over the first 1000 beliefs from the same baseline belief sequence that we used in the earlier experiments.

In order to study the distribution of δ, we needed to choose an appropriate belief distribution. We first thought of using a uniform distribution over the belief simplex. However, every sample from a uniform distribution would be an interior point of the belief simplex with probability 1. This would ignore the fact that particular sparse beliefs tend to be more relevant to the value of good policies than arbitrary beliefs. It would also report large errors for representations like the lower bound `mask` representation, whose elements support only boundary hyperfaces of the belief simplex, even though we know those representations can lead to good policies in practice.

Instead, we elected to use two different belief distributions, called A and B. Both were drawn from subsets of the baseline sequence, which meant they focused on beliefs relevant to good policies. Distribution A assigned equal probability to each unique belief that appeared among the first 1000 beliefs in the baseline sequence. Distribution B assigned equal probability to each unique belief that (1) was not in A and (2) appeared somewhere among the positions 1001-2000 in the baseline sequence.

Note that our construction of $\hat{V}$ ensured it was updated at all of the points in A and none of the points in B. Thus distribution A relates to "local error" at updated beliefs, and distribution B relates to generalization.

Taking a sample from the error distribution δ means taking a sample from one of the belief distributions and then calculating the corresponding value $\delta(b)$. Unfortunately, for our benchmark POMDPs, we had no tractable way to compute the exact value of $V^*(b)$, so we could not compute $\delta(b)$.

Instead, we settled for an approximation. In place of V^*, we used an approximate value function V^C that was constructed by running focused value iteration in the baseline configuration with all $n = 10^4$ beliefs in the baseline sequence. Recall that $\hat{V}$ was constructed using updates at only the first 1000 of these beliefs. Also recall that the baseline configuration that generated V^C used the bounds representations with the strongest generalization. Thus we expected V^C to be a more precise approximation to V^* than $\hat{V}$, justifying its use as a stand-in for V^*.

Figure 4.12 gives the cumulative distribution function (c.d.f.) of δ under various representations for the *RockSample*[5,7] problem. Specifically, the lower bound plots show the function $y = \Pr(\delta \leq x)$ and the upper bound plots show the function $y = \Pr(\delta \geq x)$. In both cases, smaller y values indicate higher accuracy, since they mean that the probability mass of δ clusters near zero. For these "equal updates" plots, the bound $\hat{V}$ for each representation was constructed using 1000 updates.

The two left panels of the figure give lower bound c.d.f.s for distribution A (top) and distribution B (bottom). Similarly, the two right panels give upper bound c.d.f.s.

Note that if a representation has good generalization, that should show up as similar error plots for distribution A (beliefs already updated) and distribution B (beliefs not already updated). Also, because the `tab+comp` representation performs no generalization, we know that its precision over distribution B (beliefs not already updated) is just the precision of the fallback function at initialization.

Examining the plots, there are a number of qualitative conclusions we can draw:

- Since all of the representations were given an equal number of updates, we were not surprised to see that the representations which favor strong generalization over faster updates showed smaller error across all of the plots.
- Variants of the `comp` and `mask` lower bound representations showed remarkably similar error over distributions A and B, indicating good generalization. The stronger full-support vectors generated by `comp` updates performed better than `mask` over both distributions.
- Passive bounded pruning had little impact; `comp` and `comp/prune` showed essentially identical precision, as did `mask` and `mask/prune`.
- In contrast, none of the upper bound representations showed good generalization. Both `comp` and `mask` were little better than `tab+comp` over distribution B. This means that, with respect to beliefs not already updated, their bounds were not much improved relative to the initial upper bound even after 1000 updates.

Figure 4.13 is similar to Figure 4.12, but uses "equal time" construction of $\hat{V}$ rather than equal updates. We allowed each representation the same amount of wallclock time to generate $\hat{V}$, so the number of updates performed differed based on speed; the actual number of updates each representation completed is shown in parentheses in the plot key. The distributions A and B were the same as for the previous experiment, so representations that completed fewer than 1000 updates were only updated over a fraction of the beliefs in distribution A.

Our conclusions from the equal time plots are:

- As before, variants of the `comp` and `mask` lower bounds generalized well, and all upper bound representations generalized poorly.
- This time variants of the `mask` lower bound outperformed `comp` over both distributions, reflecting their much faster updates.

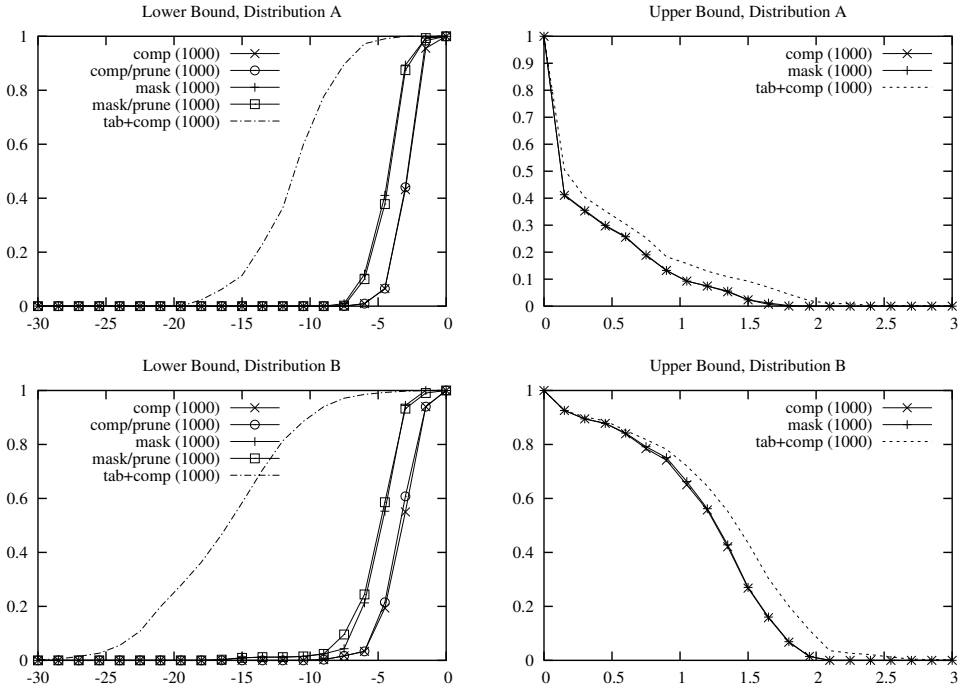


Figure 4.12: *RockSample*[5,7] precision distribution with equal updates.

- Interestingly, the `tab+comp` upper bound representation outperformed `mask` and `comp` over distribution A, even though it made only about 20% more updates than `mask`. This seems to indicate that when no representation offers particularly strong generalization, speed of updates becomes a more important factor in performance.

Error distributions for the other benchmark problems, *Tag* and *LifeSurvey*1, were similar. We have omitted the plots for brevity.

4.9 Conclusions

This chapter both introduced novel value function representations and helped to sketch how they relate to existing representations in a broader theoretical framework.

We presented four novel lower bound representations, labeled in the experiments as `comp/prune`, `mask`, `mask/prune`, and `tab+comp`. There were also two novel upper bound representations, `mask` and `tab+comp`. We showed that

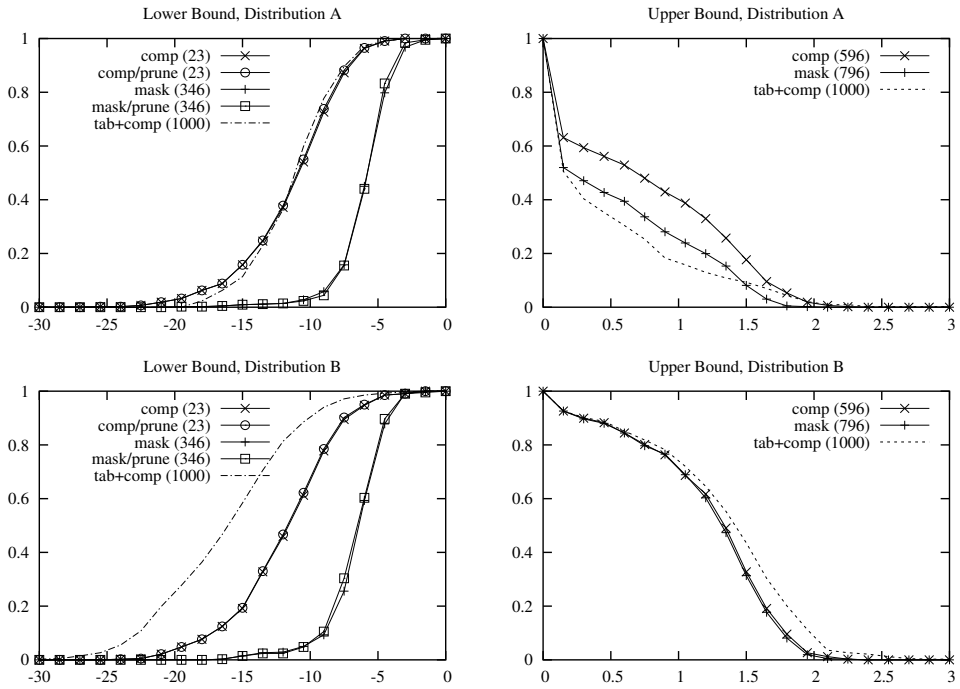


Figure 4.13: *RockSample*[5,7] precision distribution with equal wallclock time.

several existing and novel representations are strong incremental representations that preserve uniform improvability; all of these proofs are novel.

The various representations also showed a trade-off between fast point-based updates and strong generalization. This trade-off is summarized in Table 4.8. We have given each representation a subjective evaluation based on both our theoretical understanding and experiments. Evaluations for the most important columns, "Experimental time" and "Experimental space" are based wallclock time and storage requirements measured experimentally in §4.8.2.

The top part of the table lists lower bound representations and the bottom part lists upper bound representations. In each section the representations are arranged in decreasing order of generalization strength, which corresponds to increasing order of update speed. In both cases, it is interesting to see that compromise approaches that balance these criteria have the best experimental performance.

We expect, however, that the performance ordering for these representations strongly depends on the selected benchmark problems. Particularly in the case of the lower bound, update speed enhancements that leverage sparsity should be less advantageous for denser problems. Future work may provide a better understand-

Lower bound representation	Update generalization	Update speed	Experimental time	Experimental space
DENSE	**best**	poor	n/a†	n/a†
`comp`	**best**	fair	poor	poor
`comp/prune`*	good	fair	poor	poor
`mask`*	fair	good	good	good
`mask/prune`*	fair	good	**best**	**best**
`tab+comp`*	none	**best**	fair	poor

Upper bound representation	Update generalization	Update speed	Experimental time	Experimental space
CONVEXHULL	**best**	poor	n/a†	n/a†
`comp`	poor	fair	good	**best**
`mask`*	poor	fair	**best**	**best**
`tab+comp`*	none	**best**	fair	fair

* - representations with significant novel features
† - not tested; showed poor time performance in prior experiments

Table 4.8: Summary of representation comparison.

ing of this issue.

Chapter 5

Max-Planes Approximation Bounds

Many existing algorithms use the max-planes representation of the POMDP optimal value function V^*, so it is natural to ask how many α vectors are required to approximate V^* within ϵ.

Pineau et al. (2006) developed one well-known bound. A finite set $\tilde{\mathcal{B}}$ is said to cover the belief simplex $\mathcal{B}$ with sample spacing δ if every belief in $\mathcal{B}$ is within distance δ of a point in $\tilde{\mathcal{B}}$. Pineau et al. selected such a set and developed an approximation to V^* that associated one α vector with each point in $\tilde{\mathcal{B}}$. They showed that the max-norm of the approximation error was linear in the 1-norm sample spacing.

We developed two novel bounds along the same lines. The first generalizes the result of Pineau et al. to include a certain type of non-uniform sample spacing. We showed that for discounted POMDPs, tight spacing is needed only near beliefs that can be reached from the initial belief b_0 within a few time steps. Specifically, we show that a weighted max-norm of the approximation error is linear in the sample spacing according to a weighted 1-norm. Importantly, the weighted max-norm approximation error can be related to the regret of the corresponding one-step lookahead policy.

The second bound relies on a transformation of the belief simplex that spreads out the curvature of V^*. We use uniform sampling in the transformed space, which corresponds to concentrating samples in high-curvature areas of the original space. We show that the max-norm of the approximation error is *quadratic* in the 2-norm sample spacing. Roughly speaking, this reduces the number of α vectors needed to achieve a given approximation error from n to $\sqrt{n}$.

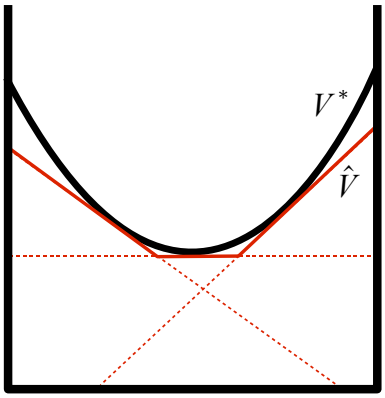


Figure 5.1: A max-planes approximation.

5.1 Technical Background

Since the optimal value function V^* of a POMDP is convex, it is natural to approximate it using the maximum of a set Γ of hyperplanes, also called α vectors, according to

$$V^*(b) \approx \hat{V}(b) = \max_{\alpha \in \Gamma} (\alpha \cdot b), \tag{5.1}$$

also written

$$\hat{V} = \max \Gamma. \tag{5.2}$$

We call this the *max-planes representation*.[1] Figure 5.1 shows an example max-planes approximation. The true value function V^* is shown as a thick curved line. The individual α vectors of Γ are shown as thin dashed lines. The approximation $\hat{V}$ is their upper surface, shown with thin solid lines.

Of course, there are many possible max-planes approximations to any given value function. We are most interested in approximations that optimize three properties:

- $\hat{V}$ should have small approximation error relative to V^*. For POMDP planning, perhaps the most relevant error measure is the policy regret induced by

[1] We use the term "max-planes representation" to specifically mean a set of α vectors. Max-planes representations can exactly represent a class of functions known as "piecewise linear and convex (PWLC) functions" or "convex polytopes". Those names are sometimes used interchangeably with our definition of "max-planes". However, we prefer the term "max-planes" to distinguish from other representations for the same function class, such as the convex hull representation (see §4.6.1).

the approximation error under the one-step lookahead policy $P\hat{V}$:

$$\text{regret}(P\hat{V}) := J\pi^*(b_0) - JP\hat{V}(b_0) \tag{5.3}$$

$$= JPV^*(b_0) - JP\hat{V}(b_0). \tag{5.4}$$

We will also relate the policy regret to other error measures that are easier to work with, including the max-norm $\|V^* - \hat{V}\|_\infty$ and a weighted max-norm.

- $|\Gamma|$ should be small. With fewer α vectors, time and space complexity are reduced.
- $\hat{V}$ should be a valid lower bound on V^* so that it can be used as the lower bound representation for focused value iteration. All of the approximations we discuss in this chapter are valid lower bounds.

Given a convex value function V, a value function error metric $D(\cdot, \cdot)$, and an approximation error $\epsilon \geq 0$, the set of *ϵ-approximating max-planes lower bounds* is defined to be

$$\text{MP}(V, \epsilon) := \{\Gamma \mid \max \Gamma \leq V \text{ and } D(V, \max \Gamma) \leq \epsilon\}. \tag{5.5}$$

Let $N(V, \epsilon)$ denote the size of the smallest ϵ-approximation

$$N(V, \epsilon) = \min_{\Gamma' \in \text{MP}(V,\epsilon)} |\Gamma'|. \tag{5.6}$$

We say a particular ϵ-approximation Γ is *minimal* if $|\Gamma| = N(V, \epsilon)$. We are interested in techniques for finding minimal or near-minimal approximations, particularly with respect to the policy regret error metric.

5.2 Fully Tangent Bounds and Belief Sampling

We say that a max-planes representation Γ is *fully tangent* to V if every α vector in Γ touches V at some point in the belief simplex. It is intuitively clear that under any reasonable error metric, among the minimal ϵ-approximating max-planes lower bounds for V, there must be some that are also fully tangent. This is because any valid lower bound that is not fully tangent can be made fully tangent without increasing its approximation error—we simply move any non-touching α vectors up until they touch V.

Based on this insight, we will restrict our attention to fully tangent representations constructed using the following process:

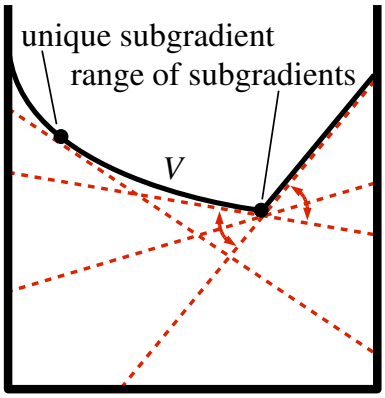


Figure 5.2: Subdifferentials of a convex function.

1. Select a finite set $\tilde{\mathcal{B}}$ of sample points from the belief simplex.

2. For each belief $b \in \tilde{\mathcal{B}}$, add to Γ a vector α which is a *subgradient* of V at b. That is, α must: (1) nowhere exceed V and (2) touch V at b. We denote the resulting set of vectors $\Gamma^{\tilde{\mathcal{B}}}$.

The set of subgradients of V at an interior point b is called the *subdifferential*, denoted $\mathcal{D}V(b)$. At points where V is differentiable there is a unique subgradient. Where V has a kink there is a range of subgradients. In general, the subdifferential is always a non-empty convex compact set. Figure 5.2 shows the subdifferential at two points for an example function.

In our asymptotic analyses it turns out not to be very important which subgradient is chosen at each belief in the sample set. For the sake of concreteness we specify an arbitrary selection rule. We use the *minimum subgradient* $\nabla^- V(b)$, which we define to be the first element of $\mathcal{D}V(b)$ in lexicographic order. $\nabla^- V(b)$ is a vector, and $\nabla_i^- V(b)$ denotes its ith component.

A *sample spacing measure* over $\mathcal{B}$ is a mapping $\delta : 2^{\mathcal{B}} \to \mathbb{R}^+$. Given a sample set $\tilde{\mathcal{B}} \subseteq \mathcal{B}$, $\delta(\tilde{\mathcal{B}})$ is its sample spacing. Small values of $\delta(\tilde{\mathcal{B}})$ correspond to denser coverage of $\mathcal{B}$ and thus (we hope) to a better approximation $\Gamma^{\tilde{\mathcal{B}}}$. For example, for any norm $\|\cdot\|_x$ on $\mathcal{B}$, there is an associated sample spacing measure δ_x, defined according to

$$\delta_x(\tilde{\mathcal{B}}) := \max_{b \in \mathcal{B}} \min_{b' \in \tilde{\mathcal{B}}} \left\| b - b' \right\|_x . \tag{5.7}$$

The value $\delta_x(\tilde{\mathcal{B}})$ is simply the maximum distance that one can get from any point in $\tilde{\mathcal{B}}$ while still remaining inside $\mathcal{B}$, with distances measured according to $\|\cdot\|_x$.

Thus we can speak of the "1-norm sample spacing" δ_1 or the "max-norm sample spacing" δ_∞.

Each sample spacing measure δ implicitly defines a *sampling strategy*, which is simply a way of spreading out any given number of sample points so as to minimize the sample spacing according to δ. If δ is based on a familiar norm such as the 1-norm, then the sample spacing is minimized when the points are more or less "evenly spread out". We informally call this *uniform sampling*. With a more exotic measure δ, the sample spacing may be minimized when points are concentrated in particular regions of $\mathcal{B}$.

The rest of this chapter studies the approximations achievable using three sampling strategies:

1. Uniform sampling of the belief simplex using δ_1. This is a brief review of some results from Pineau et al. (2006).

2. Concentrating by reachability. In discounted problems, value function precision is most important at beliefs that can be reached from b_0 within a few time steps. We generalize results from Pineau et al. to a non-uniform sampling, with sample spacing measured by a weighted 1-norm.

3. Concentrating samples by curvature. In low-curvature regions of V, a single α vector can approximate V well over a larger volume. We show how to use fewer α vectors by concentrating samples in high-curvature areas.[2]

5.3 Uniform Sampling

Pineau et al. (2006) derived the basic results about uniform sampling for POMDP value functions, relating 1-norm sample spacing of $\tilde{\mathcal{B}}$ to max-norm approximation error and policy regret of $\Gamma^{\tilde{\mathcal{B}}}$. They started by bounding the range of possible "slope" values of V^*.

For any convex function V defined over a convex set $\mathcal{B}$, let $\sigma(V)$ be the largest possible max-norm difference between any two subgradients of V over the interior of $\mathcal{B}$:

$$\sigma(V) := \max \left\{ \left\| \alpha - \alpha' \right\| \mid \alpha \in \mathcal{D}V(b), \alpha' \in \mathcal{D}V(b'), b, b' \in \text{int}(\mathcal{B}) \right\}. \quad (5.8)$$

We can upper bound $\sigma(V^*)$ when V^* is the optimal value function of a discounted POMDP. This is because each entry of an α vector subgradient to V^*

[2] Although we speak informally of "curvature", in fact our analysis uses robust techniques that apply to any convex function, even if it is not continuously twice differentiable.

must be an achievable total discounted reward value, starting from some state and following some policy. If we define

$$R_{\min} := \min_{s,a} R(s,a) \tag{5.9}$$

$$R_{\max} := \max_{s,a} R(s,a), \tag{5.10}$$

then each entry $\alpha(s)$ satisfies

$$\alpha(s) \leq R_{\max} + \gamma R_{\max} + \gamma^2 R_{\max} + \cdots \tag{5.11}$$

$$= \frac{R_{\max}}{1-\gamma}. \tag{5.12}$$

Similarly, $\alpha(s) \geq R_{\min}/(1-\gamma)$. Thus

$$\sigma(V^*) = \max_{\alpha,\alpha'} \left\|\alpha - \alpha'\right\|_\infty \tag{5.13}$$

$$= \max_{\alpha,\alpha',s} (\alpha(s) - \alpha'(s)) \tag{5.14}$$

$$\leq \frac{R_{\max} - R_{\min}}{1-\gamma}. \tag{5.15}$$

Pineau et al. then showed that the max-norm error of the $\Gamma^{\tilde{\mathcal{B}}}$ approximation is linear in the 1-norm sample spacing of $\tilde{\mathcal{B}}$:

Theorem 5.1. *(Paraphrased from Pineau et al. (2006).) Let V be a convex function defined over a compact set $\mathcal{B}$, let a sample set $\tilde{\mathcal{B}} \subseteq \mathcal{B}$ be chosen, and let $\hat{V} = \max \Gamma^{\tilde{\mathcal{B}}}$. Then*

$$\|V - \hat{V}\|_\infty \leq \delta_1(\tilde{\mathcal{B}})\sigma(V). \tag{5.16}$$

In order to relate this result to efficient POMDP value iteration, they introduced an approximate Bellman update operator $H_{\tilde{\mathcal{B}}}$ on value functions that consists of (1) applying the exact Bellman update H, then (2) forming the max-planes approximation with the sample set $\tilde{\mathcal{B}}$.[3] Unlike the exact Bellman update, $H_{\tilde{\mathcal{B}}}$ produces a value function whose representation has bounded size, at most $|\tilde{\mathcal{B}}|$ α vectors. Thus it can be computed efficiently.

Next they showed that because H is a contraction, no matter how many times the approximate version $H_{\tilde{\mathcal{B}}}$ is applied, the total error from all the approximations remains bounded:

[3] In §4.5.3, we described step (2) as "bounded pruning".

Theorem 5.2. *(Pineau et al., 2006) Let t be a positive integer and let $V_t^{\tilde{\mathcal{B}}}$ be the result of applying the $H_{\tilde{\mathcal{B}}}$ operator t times to $V_0^* = 0$. Then*

$$\|V_t^* - V_t^{\tilde{\mathcal{B}}}\|_\infty = \|H^t V_0^* - H_{\tilde{\mathcal{B}}}^t V_0^*\|_\infty \tag{5.17}$$

$$\leq \frac{(R_{\max} - R_{\min})\delta_1(\tilde{\mathcal{B}})}{(1-\gamma)^2}. \tag{5.18}$$

The max-norm approximation error can be related to policy regret using a bound from Williams and Baird (1993):

$$\text{regret}(P\hat{V}) \leq \frac{2\gamma}{1-\gamma}\|V^* - \hat{V}\|_\infty. \tag{5.19}$$

Finally, we can use the δ_1 analysis to get an upper bound on the size of the minimal ϵ-approximating max-planes lower bound, $N(V, \epsilon)$. We will make some simplifying assumptions (at the cost of loosening the resulting bound):

- We analyze convex functions over a class of domains in the form $\mathcal{X} = [0, L]^d$, where $L > 0$ is a size scale and d is the integer dimensionality. The POMDP belief simplex does not take this form, but any results we find can be applied to a POMDP belief simplex $\mathcal{B}$ using the fact that $\mathcal{B} \subseteq [0, 1]^{|\mathcal{S}|}$.
- We restrict ourselves to sampling based on tiling $\mathcal{X}$ with uniform hypercubes, where sample points are found at the centers of the cubes. This is not an ideal sampling strategy for the δ_1 spacing measure, but it is not too bad, and it has the virtue of being particularly easy to analyze.

These assumptions lead to the following bound.

Theorem 5.3. *Let $L > 0$ be a length scale, d an integer dimensionality, and $\mathcal{X} = [0, L]^d$. Let V be a convex function defined over $\mathcal{X}$. Let $\epsilon > 0$ be the maximum allowable approximation error. Then, measuring approximation error according to the max-norm, the number of α vectors needed is upper bounded by*

$$N(V, \epsilon) \leq \left\lceil \frac{1}{2}\frac{dL\sigma(V)}{\epsilon} \right\rceil^d. \tag{5.20}$$

Proof. We will cover $\mathcal{X}$ with uniform size hypercubes and place sample points at the centers of the hypercubes. Let the hypercubes have edge length W, so that the total number of hypercubes needed is

$$n = \lceil L/W \rceil^d. \tag{5.21}$$

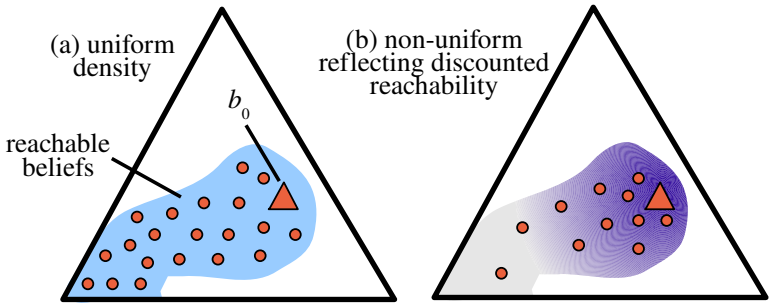


Figure 5.3: Uniform sampling vs. concentrating by reachability.

From Theorem 5.1, we know that the approximation error satisfies $\epsilon \leq \delta_1(\tilde{\mathcal{B}})\sigma(V)$. Therefore in order to ensure small enough approximation error we need to choose W so that

$$\delta_1(\tilde{\mathcal{B}}) \leq \epsilon/\sigma(V). \tag{5.22}$$

Within a hypercube, the points farthest from the center are at the corners. If the hypercube has edge length W, the distance from the center to a corner under the 1-norm is $\frac{1}{2}dW$, meaning we need

$$\frac{1}{2}dW \leq \epsilon/\sigma(V) \tag{5.23}$$

$$1/W \geq \frac{1}{2}\frac{d\sigma(V)}{\epsilon} \tag{5.24}$$

$$\tag{5.25}$$

Substituting the minimum valid value of $1/W$ into (5.21), we find the desired bound

$$n = \left\lceil \frac{1}{2}\frac{dL\sigma(V)}{\epsilon} \right\rceil^d. \tag{5.26}$$

□

5.4 Concentrating Samples By Reachability

In discounted problems, one can tolerate more approximation error at points that are only reachable from b_0 after many time steps. This section analyzes what happens when the sample spacing varies according to what we call *discounted reach-*

ability (Figure 5.3). We will see that there is a trade-off between sparse sampling and the convergence rate of value iteration.

The discounted reachability $\rho : \mathcal{B} \to \mathbb{R}$ is defined to be $\rho(b) = \gamma^L$, where L is the length of the shortest possible sequence of belief-state transitions from b_0 to b. ρ satisfies the property that $\rho(b') \geq \gamma\rho(b)$ whenever there is a single-step transition from b to b'. Based on ρ, we define a generalized sample spacing measure $\delta_{1,\rho^{-p}}$ (with $0 \leq p < 1$):

$$\delta_{1,\rho^{-p}}(\tilde{\mathcal{B}}) = \max_{b \in \bar{\Delta}} \min_{b' \in \tilde{\mathcal{B}}} \frac{\|b - b'\|_1}{\rho^{-p}(b)} \tag{5.27}$$

For the remainder of this section, we will use the shorthand δ_p for $\delta_{1,\rho^{-p}}$. In order to achieve a small δ_p value, $\tilde{\mathcal{B}}$ must have small 1-norm distance from all points in $\mathcal{B}$, but its distance from b can be proportionally larger if $\rho(b)$ is small.

When sample spacing is bounded in terms of δ_p, $H_{\tilde{\mathcal{B}}}$ does not have the error properties we want under the usual max-norm $\|\cdot\|_\infty$. We must define a new norm to reflect the fact that $H_{\mathcal{B}}$ induces larger errors where ρ is small. A weighted max-norm is a function $\|\cdot\|_\xi$ such that

$$\|V - \bar{V}\|_{\infty,\xi} = \max_b \frac{|V(b) - \bar{V}(b)|}{\xi(b)}, \tag{5.28}$$

where $\xi > 0$. Not surprisingly, $\|\cdot\|_{\infty,\rho^{-p}}$ is the norm we need; we will also refer to this norm using the shorthand $\|\cdot\|_p$ or simply as the "reachability norm". Note that when $p = 0$, δ_p reduces to the uniform 1-norm spacing measure and $\|\cdot\|_p$ reduces to the max-norm.

Our first result analyzes the behavior of H and $H_{\tilde{\mathcal{B}}}$ under the reachability norm. H is a contraction under the reachability norm, but with contraction factor γ^{1-p} rather than γ. Thus higher p values allow sparser sampling, but at the cost of a lower convergence rate.

Theorem 5.4. *The exact Bellman update H is a contraction under $\|\cdot\|_p$ with contraction factor γ^{1-p}.*

Proof. For any a, the mapping $V \mapsto a \otimes V$ has contraction factor γ^{1-p}:

$$\left\|a \otimes V - a \otimes \bar{V}\right\|_p = \max_b \frac{|(a \otimes V)(b) - (a \otimes \bar{V})(b)|}{[\rho(b)]^{-p}} \tag{5.29}$$

$$= \max_b \gamma \sum_{b'} \Pr(b' \mid b, a) \frac{|V(b') - \bar{V}(b')|}{[\rho(b)]^{-p}} \tag{5.30}$$

$$\leq \max_b \gamma \sum_{b'} \Pr(b'|b, a) \frac{|V(b') - \bar{V}(b')|}{[\gamma^{-1}\rho(b')]^{-p}} \tag{5.31}$$

$$\leq \max_b \gamma^{1-p} \sum_{b'} \Pr(b'|b, a) \ \left\|V - \bar{V}\right\|_p \tag{5.32}$$

$$= \gamma^{1-p} \left\|V - \bar{V}\right\|_p \tag{5.33}$$

Now choose an arbitrary $b \in \mathcal{B}$. Assume without loss of generality that $HV(b) \geq H\bar{V}(b)$. Choose a^* to maximize $(a^* \otimes V)(b)$, and $\bar{a}$ to maximize $(\bar{a} \otimes \bar{V})(b)$. It follows that $(a^* \otimes \bar{V})(b) \leq (\bar{a} \otimes \bar{V})(b) \leq (a^* \otimes V)(b)$, and

$$|HV(b) - H\bar{V}(b)| = |(a^* \otimes V)(b) - (\bar{a} \otimes \bar{V})(b)| \tag{5.34}$$

$$\leq |(a^* \otimes V)(b) - (a^* \otimes \bar{V})(b)| \tag{5.35}$$

$$\leq \max_a |(a \otimes V)(b) - (a \otimes \bar{V})(b)| \tag{5.36}$$

Dividing through by $\rho^{-p}(b)$ and maximizing over b yields

$$\|HV - H\bar{V}\|_p \leq \max_a \|(a \otimes V) - (a \otimes \bar{V})\|_p \tag{5.37}$$

$$\leq \gamma^{1-p}\|V - \bar{V}\|_p. \tag{5.38}$$

□

Next, we generalize Theorem 5.1 to the reachability norm.

Theorem 5.5. *Let V be a convex function defined over a compact set $\mathcal{B}$, and let a sample set $\tilde{\mathcal{B}} \subseteq \mathcal{B}$ be chosen. Then*

$$\|V - \max \Gamma^{\tilde{\mathcal{B}}}\|_p \leq \delta_p(\tilde{\mathcal{B}})\sigma(V). \tag{5.39}$$

Proof. The argument is analogous to the proof of Lemma 1 in Pineau et al. (2006). Necessary changes: (1) divide throughout by $\rho^{-p}(b')$, and (2) substitute γ^{1-p} for γ in the denominator to reflect the changed contraction properties of H under the new norm. □

Combining the two previous results, we can generalize Theorem 5.2. This establishes that when $\tilde{\mathcal{B}}$ is sampled according to δ_p, value iteration using the $H_{\tilde{\mathcal{B}}}$ update operator converges to an approximation of V^*, with bounded error under the reachability norm.

Theorem 5.6. *Let t be a positive integer and let $V_t^{\tilde{\mathcal{B}}}$ be the result of applying the $H_{\tilde{\mathcal{B}}}$ operator t times to $V_0^* = 0$. Then*

$$\|V_t^* - V_t^{\tilde{\mathcal{B}}}\|_p = \|H^t V_0^* - H_{\tilde{\mathcal{B}}}^t V_0^*\|_p \tag{5.40}$$

$$\leq \frac{(R_{\max} - R_{\min})\delta_p(\tilde{\mathcal{B}})}{(1-\gamma^{1-p})^2}. \tag{5.41}$$

Proof. The argument is analogous to Theorem 1 of Pineau et al. (2006). Necessary changes: (1) replace the max-norm with the reachability norm, and (2) replace γ with γ^{1-p}. □

Finally, we can generalize the policy regret bound from Williams and Baird to use the reachability norm.

Theorem 5.7. *Let $\hat{\pi} = P\hat{V}$. Then the regret from executing $\hat{\pi}$ rather than π^*, starting from b_0, is at most*

$$\frac{2\gamma^{1-p}}{1-\gamma^{1-p}}\|V^* - \hat{V}\|_p. \tag{5.42}$$

Proof. Choose $b \in \mathcal{B}$. We have

$$\begin{aligned}
|J\pi^*(b) - J\hat{\pi}(b)| &= |V^*(b) - (\hat{\pi} \otimes J\hat{\pi})(b)| \\
&= |V^*(b) - (\hat{\pi} \otimes \hat{V})(b) + (\hat{\pi} \otimes \hat{V})(b) - (\hat{\pi} \otimes J\hat{\pi})(b)| \\
&\leq |V^*(b) - (\hat{\pi} \otimes \hat{V})(b)| + |(\hat{\pi} \otimes \hat{V})(b) - (\hat{\pi} \otimes J\hat{\pi})(b)| \\
&\leq |HV^*(b) - H\hat{V}(b)| \\
&\quad + \gamma \sum_{b'} \Pr(b'|b, \hat{\pi}(b))|\hat{V}(b') - J\hat{\pi}(b')| \\
&\leq |HV^*(b) - H\hat{V}(b)| \\
&\quad + \gamma \sum_{b'} \Pr(b'|b, \hat{\pi}(b))\gamma^{-p}\rho^{-p}(b)\|\hat{V} - J\hat{\pi}\|_p \\
&\leq |HV^*(b) - H\hat{V}(b)| + \gamma^{1-p}\rho^{-p}(b)\|\hat{V} - J\hat{\pi}\|_p.
\end{aligned} \tag{5.43}$$

Dividing through by $\rho^{-p}(b)$ and maximizing over b gives

$$\begin{aligned}
\|J\pi^* - J\hat{\pi}\|_p &\leq \|HV^* - H\hat{V}\|_p + \gamma^{1-p}\|\hat{V} - J\hat{\pi}\|_p && (5.44)\\
&\leq \gamma^{1-p}\big(\, \|V^* - \hat{V}\|_p + \|\hat{V} - J\hat{\pi}\|_p \,\big) && (5.45)\\
&\leq \gamma^{1-p}\big(\, \|V^* - \hat{V}\|_p + \|\hat{V} - V^*\|_p \\
&\qquad + \|V^* - J\hat{\pi}\|_p \,\big) && (5.46)\\
&\leq \gamma^{1-p}\big(\, 2\|V^* - \hat{V}\|_p + \|V^* - J\hat{\pi}\|_p \,\big) && (5.47)\\
&= \gamma^{1-p}\big(\, 2\|V^* - \hat{V}\|_p + \|J\pi^* - J\hat{\pi}\|_p \,\big) && (5.48)\\
&= 2\gamma^{1-p}\|V^* - \hat{V}\|_p + \gamma^{1-p}\|J\pi^* - J\hat{\pi}\|_p. && (5.49)
\end{aligned}$$

Solving the recurrence,

$$\|J\pi^* - J\hat{\pi}\|_p \leq \frac{2\gamma^{1-p}}{1-\gamma^{1-p}}\|V^* - \hat{V}\|_p. \tag{5.50}$$

And since $\rho(b_0) = 1$, we have the desired regret bound:

$$J\pi^*(b_0) - J\hat{\pi}(b_0) \leq \frac{2\gamma^{1-p}}{1-\gamma^{1-p}}\|V^* - \hat{V}\|_p. \tag{5.51}$$

□

5.4.1 Reachability: Implications for Algorithm Design

The bias of our model toward beliefs with high discounted reachability describes current point-based value iteration algorithms more accurately than uniform sampling, at least to the extent that the algorithms perform a (typically shallow) forward exploration from the initial belief to generate $\mathcal{B}$.

The parameter p arose naturally during our analysis. $p = 0$ corresponds to uniform sampling and the usual max-norm. As p increases, samples grow less dense in areas with low reachability and the norm becomes correspondingly more tolerant. But the results show that there's no free lunch: the higher effective discount factor γ^{1-p} under the new norm means that more updates are required and the final error bounds are looser.

One natural way to apply these results is to use δ_p rather than δ_1 as the spacing measure used when selecting points to add to $\mathcal{B}$ in the belief set expansion phase of the PBVI-SSEA algorithm.[4]

[4] Pineau et al. (2006) presented several versions of the basic PBVI algorithm. PBVI-SSEA is the "standard" version that uses Stochastic Simulation with Exploratory Actions to expand the belief set.

After our original publication presenting the reachability metric, Izadi and Precup (2006) implemented this idea and compared the expected reward of policies output by PBVI-SSEA using δ_1 vs. δ_p, with various values of the p parameter. Their results showed that both versions eventually converged to a near-optimal policy, but did not provide any information about relative time performance. More recent experiments suggest that use of δ_p provides some time performance benefit on the *Tag* benchmark problem (Izadi, private communication).

5.5 Concentrating Samples By Curvature

Our second novel approach to max-planes approximation relies on a transformation of the belief simplex that spreads out the curvature of V^*. We use uniform sampling in the transformed space, which corresponds to concentrating samples in high-curvature areas of the original space. We show that the max-norm of the approximation error is *quadratic* in the 2-norm sample spacing. Roughly speaking, this reduces the number of α vectors needed to achieve a given approximation error from n to $\sqrt{n}$.

To simplify our analysis, we will again assume that the convex function of interest, V, has the domain $\mathcal{X} = [0, L]^d$. The smoothing transform is denoted $Z : \mathbb{R}^d \to \mathbb{R}^d$. For any belief b, $Z(b)$ is a vector whose components are $Z_i(b)$, defined according to[5]

$$Z_i(b) := \sqrt{b_i(\nabla_i^- V(b) - k_i)}, \tag{5.52}$$

where each k_i is a constant offset

$$k_i := \inf_{b \in \text{int}(\mathcal{B})} \nabla_i^- V(b). \tag{5.53}$$

The infimum is bounded under the assumption that $\sigma(V)$ is finite. This offset is designed to ensure that the value inside the square root is positive.

We use the smoothing transform to define a sample spacing measure

$$\delta_Z(\tilde{\mathcal{B}}) := \max_{b \in \mathcal{X}} \min_{b' \in \tilde{\mathcal{B}}} \|Z(b) - Z(b')\|_2. \tag{5.54}$$

Under δ_Z, distances between points are measured using the 2-norm in the trans-

The uniform spacing theorems discussed in §5.3 were originally presented as part of an argument that PBVI-SSEA converges.

[5] Here we find it convenient to refer to elements of the belief vector b using the notation b_i in place of our usual notation $b(s)$.

formed space. Z magnifies the distance between two points if they have very different subgradients and shrinks the distance if they have similar subgradients. Thus uniform sampling in the transformed space corresponds to concentrating samples in regions of the original space where the minimal subgradient is changing quickly, which one can informally think of as "high-curvature" regions.

Our first result shows that the max-norm approximation error is quadratic in the δ_Z sample spacing.

Theorem 5.8. *Let V be a convex function defined over $\mathcal{X}$, let a sample set $\tilde{\mathcal{B}} \subseteq \mathcal{X}$ be chosen, and let $\hat{V} = \max \Gamma^{\tilde{\mathcal{B}}}$. Then*

$$\|V - \hat{V}\|_\infty \leq \delta_Z^2(\tilde{\mathcal{B}}). \tag{5.55}$$

Proof. Choose $b \in \mathcal{X}$. We want to bound $|V(b) - \hat{V}(b)|$. Since $\hat{V}$ is a lower bound, $\hat{V}(b) \leq V(b)$. Let b' be the sample point from $\tilde{\mathcal{B}}$ that is closest to b according to the 2-norm in the transformed space. Let $\alpha = \nabla^- V(b)$ and $\alpha' = \nabla^- V(b')$. Since $\alpha' \in \Gamma^{\tilde{\mathcal{B}}}$, we have $\hat{V}(b) \geq \alpha' \cdot b$. Putting the two bounds on $\hat{V}(b)$ together, we have

$$|V(b) - \hat{V}(b)| \leq |V(b) - \alpha' \cdot b|. \tag{5.56}$$

Using the convexity of V and an analogy with the mean value theorem, one can show that

$$V(b) = V(b') + (b - b') \cdot u \tag{5.57}$$

for some vector u such that

$$(b - b') \cdot \alpha' \;\; \leq \;\; (b - b') \cdot u \;\; \leq \;\; (b - b') \cdot \alpha. \tag{5.58}$$

Then

$$V(b) = V(b') + (b - b') \cdot u \tag{5.59}$$
$$\alpha' \cdot b = V(b') + (b - b') \cdot \alpha'. \tag{5.60}$$

Subtracting,

$$|V(b) - (\alpha' \cdot b)| \leq |(b - b') \cdot (u - \alpha')| \tag{5.61}$$
$$\leq |(b - b') \cdot (\alpha - \alpha')| \tag{5.62}$$
$$\leq \sum_i |(b_i - b'_i)(\alpha_i - \alpha'_i)|. \tag{5.63}$$

In order to relate this to Z, we introduce the following inequality. Let

$x, x', y, y' \in \mathbb{R}^+$. Then

$$
\begin{aligned}
(x - x')(y - y') &= xy + x'y' - xy' - x'y && (5.64)\\
&\leq xy + x'y' - 2\sqrt{xy'x'y} && (5.65)\\
&= xy + x'y' - 2\sqrt{xyx'y'} && (5.66)\\
&= (\sqrt{xy} - \sqrt{x'y'})^2, && (5.67)
\end{aligned}
$$

where (5.65) relies on the arithmetic-geometric means inequality. Applying to each term in the sum above,

$$
\begin{aligned}
|V(b) - \hat{V}(b)| &\leq |V(b) - (\alpha' \cdot b)| && \text{[by (5.56)]} && (5.68)\\
&\leq \sum_i \left|(b_i - b'_i)(\alpha_i - \alpha'_i)\right| && \text{[by (5.63)]} && (5.69)\\
&= \sum_i \left|(b_i - b'_i)\left[(\alpha_i - k_i) - (\alpha'_i - k_i)\right]\right| && && (5.70)\\
&\leq \sum_i \left(\sqrt{b_i(\alpha_i - k_i)} - \sqrt{b'_i(\alpha'_i - k_i)}\right)^2 && \text{[by (5.67)]} && (5.71)\\
&= \sum_i (Z_i(b) - Z_i(b'))^2 && \text{[def. of } Z_i\text{]} && (5.72)\\
&= \|Z(b) - Z(b')\|_2^2 && && (5.73)\\
&\leq \delta_Z^2(\tilde{\mathcal{B}}). && && (5.74)
\end{aligned}
$$

□

Next, we use the δ_Z spacing measure to develop a tighter bound on the number of α vectors needed to achieve a given approximation error. We are able to use fewer vectors than with uniform sampling because the approximation error is quadratic in $\delta_Z(\tilde{\mathcal{B}})$ but only linear in $\delta_1(\tilde{\mathcal{B}})$. Roughly speaking, the number of vectors required is reduced from n to $\sqrt{n}$.

Theorem 5.9. *Let $L > 0$ be a length scale, d an integer dimensionality, and $\mathcal{X} = [0, L]^d$. Let V be a convex function defined over $\mathcal{X}$. Let $\epsilon > 0$ be the maximum allowable approximation error under the max-norm. Then the number of α vectors needed is upper bounded by*

$$
N(V, \epsilon) \leq \left\lceil \frac{1}{2}\sqrt{\frac{dL\sigma(V)}{\epsilon}} \right\rceil^d. \tag{5.75}
$$

Proof. Let $\mathcal{Y}$ be the image of $\mathcal{X}$ under the transform Z. Theorem 5.8 implies that the number of facets n needed to approximate V within ϵ is bounded by the number of 2-norm balls with radius $\sqrt{\epsilon}$ needed to cover $\mathcal{Y}$.

There are a number of methods to cover $\mathcal{Y}$ which give tighter or looser bounds. In the interest of developing a closed-form (if not particularly tight) bound, we will tile $\mathcal{Y}$ with a regular grid of hypercubes, then cover $\mathcal{Y}$ with balls that circumscribe the hypercubes.

If a d-hypercube has edge length W, the 2-norm distance from its center to any point is bounded by the distance to the corners, $\frac{1}{2}W\sqrt{d}$. In order to circumscribe the hypercube with a d-hypersphere of radius $\sqrt{\epsilon}$, we need $W \leq 2\sqrt{\epsilon/d}$.

$\mathcal{Y}$ is bounded along each dimension i. Choose $b \in \mathcal{X}$. On the low side, $b_i \geq 0$ and $(\nabla_i^- V(b) - k_i) \geq 0$, so $Z_i(b) \geq 0$. On the high side, $b_i \leq L$ and $\left\|\nabla_i^- V(b) - k_i\right\|_\infty \leq \sigma(V)$, so $Z_i(b) \leq \sqrt{L\sigma(V)}$.

To form the regular grid, each dimension is cut at uniform intervals of W into k pieces, meaning the number n of cells is

$$n = k^d \tag{5.76}$$

$$= \left\lceil \sqrt{L\sigma(V)}/W \right\rceil^d \tag{5.77}$$

$$= \left\lceil \frac{1}{2}\sqrt{\frac{dL\sigma(V)}{\epsilon}} \right\rceil^d . \tag{5.78}$$

□

We also develop a slightly tighter asymptotic bound using prior work on minimum density covers.

Theorem 5.10. *Let $\mathcal{X}, V, \epsilon$ be defined as in Theorem 5.9. As $\epsilon \to 0$,*

$$N(V, \epsilon) \lesssim \frac{\theta_d}{\kappa_d} \left(\frac{L\sigma(V)}{\epsilon} \right)^{d/2} , \tag{5.79}$$

where κ_d is the volume of the unit ball in $\mathbb{R}^d$ and θ_d is the minimum density of covering of $\mathbb{R}^d$ with unit balls under the 2-norm.

Remark 5.11. *There are elementary closed-form formulas for κ_d (Weisstein, 1999). Various bounds have been provided for θ_d; refer to Chapter 2 of Conway and Sloane (1999) for a recent review.*

Proof. To derive this bound, we again cover $\mathcal{Y}$ with balls of radius $\sqrt{\epsilon}$. This time we are using the minimum-density covering. Lemma 1 of Gruber (1993) showed

that for a Jordan measurable set J, the minimum number of balls of radius r needed to cover J as $r \to 0$ follows

$$n \sim \frac{\text{volume}(J)}{r^d \kappa_d / \theta_d}. \tag{5.80}$$

Intuitively, one can think of the denominator as the volume of each ball in the covering, using θ_d to correct for overlap.

$\mathcal{Y}$ is Jordan measurable with volume at most $(L\sigma(V))^{d/2}$, which gives the stated bound. □

5.5.1 Curvature: Implications for Algorithm Design

We have shown that we can approximate a convex function V within ϵ using fewer α vectors if we use tighter sample spacing in high-curvature areas of V and looser sample spacing elsewhere.

The reachability spacing approach selected a sample set using only the parameter p and the (static) structure of the POMDP. In contrast, the curvature spacing approach requires knowledge of the value function to be approximated. Unfortunately, if we are trying to approximate the optimal value function V^*, we cannot use curvature sampling to select an ideal sample set prior to calculating V^* itself.

However, there is a natural way to integrate curvature sampling into the PBVI-SSEA algorithm. At any belief set expansion phase, in place of the uniform δ_1 metric, we can use the δ_Z metric induced by the *current approximation* to the optimal value function. Under this approach, the quality of the δ_Z metric (and, in turn, the sample set) improves throughout execution along with the value function approximation. Since the size of PBVI-SSEA's sample set doubles at each expansion phase, samples chosen early on according to a poor spacing metric should be quickly be outnumbered by better spaced samples chosen later. We have yet to evaluate this approach experimentally.

5.6 Conclusions

With respect to the commonly used max-planes representation for POMDP value functions, we developed two new ways to use fewer α vectors to approximate the optimal value function V^* within ϵ. In the first approach, we focused on approximating V^* well in the neighborhood of beliefs that are quickly reachable from the initial belief b_0, requiring fewer α vectors in other parts of the belief simplex but still guaranteeing that policies based on the approximation have small regret. In the second approach, we found a way to smooth out the curvature of V^* so that each

α vector could effectively cover more volume. Roughly speaking, this approach reduced the number of α vectors needed from n to $\sqrt{n}$.

Chapter 6

Heuristic Search

Heuristic search can greatly increase the efficiency of MDP and POMDP solution algorithms by focusing effort on the most relevant parts of the search graph. This chapter presents two search strategies, Heuristic Search Value Iteration (HSVI) and Focused Real-Time Dynamic Programming (FRTDP), that fit into the focused value iteration framework of Chapter 3. Both HSVI and FRTDP can also be viewed as variants of the Trial-Based Real-Time Dynamic Programming (RTDP) algorithm (Barto et al., 1995).[1]

HSVI was developed as part of our first implementation of the focused value iteration framework, and was originally designed specifically to solve POMDPs. HSVI differs from RTDP in several ways:

- HSVI gains important benefits from fitting into the focused value iteration framework. Whereas RTDP maintains only an upper bound, focused value iteration maintains two-sided bounds on V^* that can be used to bound the regret of the current policy. This provides a natural way to monitor the rate of convergence so that search can be halted when a desired precision is reached.

- HSVI also uses the two-sided bounds to direct search more effectively, preferring to visit the states that contribute most to the uncertainty at s_0. Updating those states has the greatest potential to reduce the regret bound.

- If the MDP is discounted, HSVI adjusts how it prioritizes states, using the discounting to guarantee convergence even when the state space is infinite, as it is in the belief-MDP representation of a POMDP. (RTDP is not guaranteed to converge when applied to POMDPs.)

[1] Software implementations of most of the heuristic search algorithms described in this chapter are freely available as part of the ZMDP software package, which you can download at `http://www.cs.cmu.edu/~trey/zmdp/`.

We applied HSVI to several benchmark problems from the MDP and POMDP literature, comparing its performance to several other heuristic search algorithms. HSVI failed to converge over most of the MDP problems because they were not discounted (see §6.5 for a discussion of termination guarantees). However, compared to the state-of-the-art HDP+L algorithm, HSVI required less wallclock time to achieve the same regret bound over two of the three POMDPs, achieving 10x speedup in the best case.

FRTDP was developed later based both on RTDP and on our experience with HSVI. It differs in that:

- FRTDP, like HSVI, prefers to visit states that contribute to uncertainty at s_0. However, FRTDP is less myopic. It uses cached priority information to avoid fruitlessly revisiting states that resist improvement.
- FRTDP has a maximum depth termination criterion to abort trials that run too long. This technique, proposed by Barto et al. (1995) but never implemented, helps to cut off fruitless trials in "one-way door" problems where an irreversible early decision can make it much more difficult to reach a goal. The maximum depth increases adaptively from trial to trial as the search progresses.

FRTDP was tested under the same conditions as HSVI. It had the best wallclock time performance across six of the seven benchmark problems (all four MDPs and two of the three POMDPs), achieving up to 28x speedup relative to the best prior algorithm.

RTDP, HSVI, and FRTDP all explore the search graph by repeated trials. A trial starts at s_0 and explores forward. At each forward step, the current state is updated and a successor state is chosen via heuristics for *action selection* and *outcome selection* that vary depending on the search strategy. A trial finishes when the search strategy's *trial termination criterion* is satisfied. In an optional second phase of the trial called *round-trip updating*, the search retraces its path, updating the same states in reverse order in order to propagate information back to s_0. HSVI and FRTDP perform round-trip updating, but RTDP does not.

Figure 6.1 shows a trial. Circles are nodes of the search graph. The node at the top of the graph is the initial state s_0. Dark circles are states that satisfy the search strategy's trial termination condition (which varies between different search strategies). Arrows emanating from nodes are actions available to the agent. Thinner arrows emanating from the action arrows are possible outcomes of each action.

The thick solid arrow running down the figure shows the states visited and updated during the forward exploration phase of the trial. The corresponding dashed

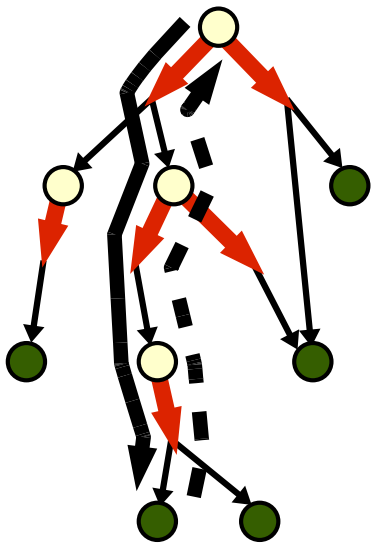


Figure 6.1: A search strategy trial.

	RTDP	HSVI	FRTDP
Bounds maintained	V^U only	V^U and V^L	V^U and V^L
Action selection (planning)	PV^U	PV^U	PV^U
Action selection (run-time)	PV^U	PV^L	PV^L
Outcome selection	stochastic	wtd. excess	wtd. priority
Trial termination	goal states	excess ≤ 0	$\Delta \leq 0$, max depth
Round-trip updating	no	yes	yes
Convergence conditions	R1-R3	H1-H2	R1-R3

Table 6.1: Comparison of search strategy features

arrow shows the round-trip updating phase of the trial, as performed by HSVI and FRTDP.

Table 6.1 provides an overview comparison of the search strategies. The rest of the chapter is organized as follows: §6.1 introduces classes of problems that we would like our algorithms to solve. §6.2 is a review of RTDP. §§6.3-6.4 present the HSVI and FRTDP algorithms, respectively. §6.5 analyzes conditions under which HSVI and FRTDP terminate. §6.6 compares the empirical performance of RTDP, HSVI, FRTDP, and some other search strategies, as applied to benchmark problems from the MDP and POMDP literature.

6.1 Problem Classes

The search strategies in this chapter can be applied to any MDP or POMDP, but we need additional problem structure to guarantee value function bound convergence and termination. We analyze the search strategies in the context of two classes of MDPs. The first class we call *RTDP-solvable MDPs*, which satisfy:

R1. $\mathcal{S}$ is finite.

R2. There exists at least one *proper policy*; that is, a policy that reaches a goal state $s \in \mathcal{G}$ from any starting state with probability 1.

R3. Every improper policy incurs infinite cost (that is, infinite negative reward) for at least one state.[2]

As the name suggests, these constraints guarantee convergence of the RTDP algorithm (Barto et al., 1995).

In an RTDP-solvable MDP, the agent has a single overriding task whose achievement is represented by reaching a goal state. Many interesting problems can be expressed in this form, but not all. For instance, if the problem has several tasks and the agent must accomplish as many as possible, it is more natural to give the agent a positive reward for accomplishing each single task, and there may not be any absorbing goal states. Also, the belief-MDP for a POMDP is not RTDP-solvable because its state space (that is, the belief space of the POMDP) is infinite.

The second problem class we call *discounted finite-branching MDPs*, which satisfy:

H1. The problem is discounted; $\gamma < 1$.

H2. The problem has a global branching factor j, meaning that from any state at most j successor states can be reached via any choice of action and any possible outcome.

The assumption of discounting is not appropriate for every problem; however, when applicable it is very powerful because it allows the value of a policy to be calculated to any desired precision while only examining the search graph to finite depth. This allows us to guarantee convergence of some search strategies even with

[2]Normally, RTDP-solvable problems are formulated such that all actions outside a goal state incur a non-zero cost, represented in our notation as a strictly negative reward. This non-zero cost property implies R3. However, R3 is also guaranteed under the weaker condition that *traversing any loop* in the search graph incurs strictly negative reward (leaving open the possibility that certain actions provide positive reward).

an infinite state space. Many problems with multiple tasks can be naturally represented within this class, and the belief-MDP for a discounted POMDP satisfies H1-H2 with $j = |\mathcal{A}||\mathcal{O}|$.

6.2 RTDP Review

The novel search strategies we present later in this chapter can be viewed as variants of the Real-Time Dynamic Programming (RTDP) algorithm of Barto et al. (1995). Barto et al. present multiple versions of RTDP; the most relevant version for our purposes is a form of Trial-Based RTDP in which s_0 is chosen as the start state of every trial. Note that Barto et al.'s presentation of RTDP spends considerable time discussing how to plan concurrently with system identification. In contrast, we are concerned only with off-line planning based on an exact model.

Algorithm 6.1 is an implementation of Trial-Based RTDP using notation compatible with our earlier discussion of focused value iteration. Although the structure of RTDP is broadly similar to focused value iteration, RTDP is not quite an instance of the algorithm schema because it does not keep a lower bound, and thus cannot monitor the regret bound as a condition for overall algorithm termination.

RTDP selects actions greedily according to the upper bound, the same approach used by HSVI and FRTDP (see §6.3.2 for a discussion). Given an action, RTDP's outcome selection rule stochastically selects a successor state according to the transition probabilities in the MDP model. This form of outcome selection has some good theoretical properties and is attractive due to its extreme simplicity, but it also means that RTDP often continues fruitlessly visiting states after their value has already converged. HSVI and FRTDP use more sophisticated outcome selection rules that try to mitigate this problem.

A state is *relevant* if it can be reached by at least one optimal policy. Under conditions R1-R3, RTDP's V^U value function is guaranteed, with probability 1, to converge to V^* over the set of relevant states. This implies that PV^U converges to an optimal policy with probability 1 (Barto et al., 1995).

RTDP was originally conceived as an algorithm for solving finite discrete MDPs using a tabular value function representation. Barto et al.'s convergence proof applies only in that case. Geffner and Bonet (1998) applied RTDP to POMDPs using a grid-based value function representation for V^U, an approach called RTDP-BEL. Unfortunately, although RTDP-BEL has been successfully applied to some problems, it is not guaranteed to converge in general.

As written, our version of RTDP leaves open the choice of representation for V^U. When used to solve POMDPs, it may be combined with any of the representations discussed in Chapter 4. This implementation of RTDP will be used later

Algorithm 6.1 Real-Time Dynamic Programming (RTDP), a search strategy (Barto et al., 1995).

```
1: uses implementation <UB> conforming to UPDATABLESIG
2:
3: function RTDP() :
4:    V^U ← <UB>.initialValueFunction()
5:    loop:
6:       trialRecurse(s_0)
7:
8: function trialRecurse(s) :
9:    if s ∈ G return
10:   V^U ← <UB>.update(V^U, s)
11:   a* ← arg max_a(a ⊗ V^U)(s)
12:   s' ← chooseSuccessorStochastically(s, a*)
13:   trialRecurse(s')
```

when we compare experimental performance of different search strategies.

6.3 Heuristic Search Value Iteration (HSVI)

HSVI was developed as part of our first implementation of focused value iteration, and was originally designed specifically to solve POMDPs. Relative to RTDP, HSVI uses the two-sided bounds of focused value iteration to guarantee the quality of its output policy, to more effectively guide search, and to guarantee convergence over the class of discounted finite-branching MDPs, which includes discounted POMDPs represented as belief-MDPs.[3]

HSVI is presented in Algorithm 6.2 as a complete algorithm that integrates heuristic search elements with the generic main loop of focused value iteration. Unfortunately, this formulation obscures the modularity discussed in Chapter 3, where we introduced the generic main loop as a separate component that could be reused with different heuristic search algorithms.

We could have written a functionally equivalent version of HSVI as a separate module explicitly conforming to the focused value iteration SEARCHSIG interface.

[3] We should note that in Smith and Simmons (2005), we used the names "HSVI1" and "HSVI2" to describe different complete implementations of focused value iteration. Both were based on the HSVI search strategy we describe here, but they used different value function representations for V^L and V^U.

However, due to the way that state selection is interleaved with value function updates, a modular implementation would have required that either (1) we replace the recursive implementation of HSVI trials with explicit stack data structures, increasing complexity and hiding the recursive structure, or (2) we use coroutine program semantics that allow simultaneous interleaving and recursion, but at the cost of making it harder to translate the implementation into programming languages that do not support coroutines. In the end we judged that it would be more useful to present an integrated implementation.

6.3.1 HSVI Bounds Intervals

Bounds intervals are central to HSVI. Recall that $(a \otimes V)(s)$ is the expected value of executing action a in state s, then receiving future reward according to the value $V(s')$ of the successor state s' after the state transition. We define the interval functions

$$\hat{V}(s) := [V^L(s),\ V^U(s)] \tag{6.1}$$

$$(a \otimes \hat{V})(s) := [(a \otimes V^L)(s),\ (a \otimes V^U)(s)] \tag{6.2}$$

$$H\hat{V}(s) := [HV^L(s),\ HV^U(s)] \tag{6.3}$$

$$\mathrm{width}([v_{\min}, v_{\max}]) := v_{\max} - v_{\min}. \tag{6.4}$$

$\hat{V}(s)$ is the range of possible values of the state s given the current bounds, $(a \otimes \hat{V})(s)$ is the range of possible values for s given that action a is selected, $H\hat{V}$ is the result of applying the Bellman update to both bounds, and the $\mathrm{width}()$ function measures the width of an interval.

Figure 6.2 graphically illustrates the identity

$$H\hat{V}(s) = [\max_a (a \otimes V^L)(s),\ \max_a (a \otimes V^U)(s)] \tag{6.5}$$

by showing, for a particular state s, the intervals $(a_i \otimes \hat{V})(s)$ and how they relate to $H\hat{V}(s)$.

Recall that when V^L is a uniformly improvable lower bound, Theorem 3.16 states that the long-term reward $JPV^L(s_0)$ achieved by the one-step lookahead policy PV^L is at least as good as the value $V^L(s_0)$ that V^L itself estimates. Since the long-term reward $V^*(s_0)$ of an optimal policy cannot exceed the upper bound $V^U(s_0)$, the regret from executing PV^L rather than an optimal policy cannot exceed $V^*(s_0) - V^L(s_0) \leq V^U(s_0) - V^L(s_0)$. In other words,

$$\mathrm{regret}(PV^L) \leq \mathrm{width}(\hat{V}(s_0)). \tag{6.6}$$

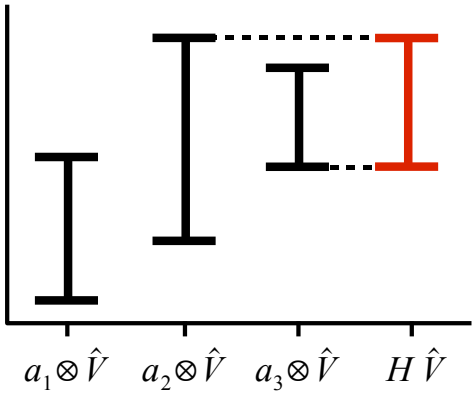


Figure 6.2: Relationship between $a_i \otimes \hat{V}$ and $H\hat{V}$.

Since our overall goal is to return a policy with small regret, HSVI is designed to prioritize the state updates that will most reduce the bounds interval $\text{width}(\hat{V}(s_0))$. We now show how this high-level design goal motivates both the action and outcome selection heuristics.

6.3.2 HSVI Action Selection

During planning, HSVI chooses actions greedily according to the upper bound, like RTDP. That is, from state s it chooses action a^* according to

$$a^* = \max_a (a \otimes V^U)(s) \tag{6.7}$$

The idea is that actions that currently seem to perform well are more likely to be part of an optimal policy; thus selecting such actions will lead HSVI to update states whose values are relevant to good policies. This is sometimes called the IE-MAX heuristic (Kaelbling, 1993).

One might ask why HSVI chooses the action that maximizes the upper bound rather than the lower bound. One way to think about it is that every time HSVI selects an action a from a state s, it in effect collects two kinds of evidence about a. The lower bound $(a \otimes V^L)(s)$ is a summary of the positive evidence that a performs well—larger values indicate more positive evidence. The upper bound $(a \otimes V^U)(s)$ is a summary of the negative evidence that a performs poorly—larger values indicate less negative evidence.

If HSVI's action selection heuristic tried to maximize the lower bound, it would tend to choose the same action every time, because further work on an action can

only increase the amount of positive evidence about it. By failing to investigate other actions, it might never discover the optimal action. Instead, HSVI tries to maximize the upper bound, choosing the action with the smallest amount of negative evidence, which encourages investigation of untried actions that have little negative evidence so far.

Another way to see why upper bound action selection helps guarantee convergence is to focus on trying to shrink the uncertainty at the current state s. If the action a^* is chosen to maximize the upper bound, the uncertainty in $H\hat{V}(s)$ after an update can never be larger than $(a^* \otimes \hat{V})(s)$. Figure 6.2 provides some intuition—note that the interval $H\hat{V}$ is smaller than $a_2 \otimes \hat{V}$ (maximum upper bound) but larger than $a_3 \otimes \hat{V}$ (maximum lower bound). In general,

$$\begin{aligned}
\mathrm{width}(H\hat{V}(s)) &= \max_a (a \otimes V^U)(s) - \max_a (a \otimes V^L)(s) && (6.8)\\
&= (a^* \otimes V^U)(s) - \max_a (a \otimes V^L)(s) && (6.9)\\
&\leq (a^* \otimes V^U)(s) - (a^* \otimes V^L)(s) && (6.10)\\
&= \mathrm{width}((a^* \otimes \hat{V})(s)). && (6.11)
\end{aligned}$$

Thus by choosing action a^* and shrinking the bounds interval $(a^* \otimes \hat{V})(s)$, HSVI is guaranteed to eventually shrink the interval $H\hat{V}(s)$.

Note that while HSVI's action selection heuristic maximizes the upper bound at planning time, the final policy PV^L it returns for use at run-time is based on the lower bound. This is due in part to the fact that PV^L has provably small regret

$$\mathrm{regret}(PV^L) \leq \mathrm{width}(\hat{V}(s_0)). \tag{6.12}$$

In contrast, the strongest result we are aware of for the quality of the policy PV^U based on the upper bound is

$$\mathrm{regret}(PV^U) \leq \frac{2\gamma}{1-\gamma} \left\| V^U - V^L \right\|_\infty, \tag{6.13}$$

which is weaker both because the leading coefficient is large for weakly discounted problems ($\gamma \approx 1$) and, more importantly, because in order for the result to be useful we would need $\mathrm{width}(\hat{V}(s))$ to be small *for every reachable state* s, not just for s_0.

Of course, if these theoretical bounds on regret are too loose, they may not be good indicators of the relative performance of PV^L and PV^U when applied to typical problems. However, empirical tests in §6.6 demonstrate that, at least in the cases we studied, PV^L outperforms PV^U in practice.

6.3.3 HSVI Outcome Selection

Recall that HSVI's primary goal is to shrink the regret bound by decreasing uncertainty at the initial state s_0. Its purpose in updating states during the later parts of a trial is to have the largest possible impact on uncertainty at states visited earlier in the trial and thus at s_0. Translating this into local decision-making, when moving forward from a state s, HSVI selects the outcome whose improvement has the greatest potential to impact the *excess uncertainty* at s.

The excess uncertainty of a state s encountered at depth d in a trial is defined to be[4]

$$\text{excess}(\hat{V}, s, d) := \text{width}(\hat{V}(s)) - \epsilon\gamma^{-d}. \tag{6.14}$$

The $\epsilon\gamma^{-d}$ term is motivated by the way discounting affects the flow of uncertainty back to s_0 from nodes deeper in the search tree. Suppose that for some state s, all the successors s' of s have uncertainty smaller than δ. It turns out that

$$\begin{aligned}
\text{width}(H\hat{V}(s)) &\leq \gamma \sum_{s'} T(s, a^*, s')\text{width}(\hat{V}(s')) && (6.15)\\
&\leq \gamma \sum_{s'} T(s, a^*, s')\delta && (6.16)\\
&= \gamma\delta. && (6.17)
\end{aligned}$$

In other words, the uncertainty at the successors s' "flows back" to the parent s after an update, but decreased by the discount factor γ.

One can show inductively that if all the states at depth d have uncertainty at most $\epsilon\gamma^{-d}$, then after enough updates to propagate that information back to s_0, we will have $\text{width}(\hat{V}(s_0)) \leq \epsilon$, which is the termination condition for the overall algorithm. Thus we have defined excess uncertainty to be any uncertainty above and beyond the amount $\epsilon\gamma^{-d}$ that HSVI can permit while still guaranteeing eventual termination.

From a state s, having selected an action a^*, HSVI's outcome selection heuristic chooses the successor state s^* that contributes most to the excess uncertainty of s

$$s^* := \max_{s'} T(s, a^*, s')\ \text{excess}(\hat{V}, s', d+1), \tag{6.18}$$

[4] We omit the $\hat{V}$ argument to excess() when it is obvious what pair of value function bounds is being used.

which is motivated by the property that, after s is updated,

$$\text{excess}(H\hat{V}, s, d) \leq \gamma \sum_{s'} T(s, a^*, s') \, \text{excess}(\hat{V}, s', d+1). \tag{6.19}$$

In effect, the excess uncertainty at s flows back from its successors s'. In order to have the largest potential impact on the excess uncertainty at s, HSVI works on decreasing the uncertainty flowing from the successor s' that makes the largest contribution to the sum.

The forward exploration phase of an HSVI trial ends if, d steps into the trial, HSVI reaches a state s that is *finished at depth* d, meaning it has no excess uncertainty:

$$\text{excess}(\hat{V}, s, d) \leq 0. \tag{6.20}$$

The fact that HSVI outcome selection uses the same excess uncertainty heuristic as trial termination is key. It means that the outcome selection heuristic guides forward exploration away from finished states, keeping HSVI from entering an endless loop in which each trial visits the same sequence of states ending in the same finished state.

6.3.4 Running HSVI in Anytime Fashion

The version of HSVI(ϵ) presented above assumes that we know in advance that we want a policy with regret bounded by ϵ. In practice, however, we often do not know what a reasonable ϵ is for a given problem—we just want the algorithm to do the best it can in the available time.

To support this alternate mode of operation, we present a driver routine called ConvergentHSVI, Algorithm 6.3. ConvergentHSVI repeatedly calls HSVI(ϵ) with smaller values of the regret bound ϵ so that the regret converges to zero in the limit. ConvergentHSVI can be used in anytime fashion. If interrupted at the end of any trial, its resulting policy PV^L has regret at most $V^U(s_0) - V^L(s_0)$.

The outer loop of ConvergentHSVI adjusts the ϵ argument to HSVI. It is initialized to ϵ_0 and multiplied by a constant factor $k_\epsilon < 1$ every time the previous value of ϵ is achieved. The parameters k_ϵ and ϵ_0 are tunable. Limited empirical testing suggests that performance is not very sensitive to the parameter values, and the following give reasonable performance:

$$k_\epsilon = 0.95 \tag{6.21}$$

$$\epsilon_0 = k_\epsilon (V_0^U(s_0) - V_0^L(s_0)), \tag{6.22}$$

Algorithm 6.2 Heuristic Search Value Iteration (HSVI), a search strategy.

```
1: uses implementation <LB> conforming to UPDATABLESIG
2: uses implementation <UB> conforming to UPDATABLESIG
3:
4: function HSVI(ε) :
5:    V^L ← <LB>.initialValueFunction()
6:    V^U ← <UB>.initialValueFunction()
7:    while V^U(s_0) − V^L(s_0) > ε :
8:       trialRecurse(s_0, d = 0)
9:    return V^L
10:
11: function trialRecurse(s, d) :
12:    if excess(s, d) ≤ 0 :
13:       return
14:    update(s)
15:    a* ← arg max_a (a ⊗ V^U)(s)
16:    s* ← arg max_{s'∈N(s,a*)} T(s, a*, s') excess(s', d + 1)
17:    trialRecurse(s*, d + 1)
18:    update(s)
19:
20: function update(s) :
21:    V^L ← <LB>.update(V^L, s)
22:    V^U ← <UB>.update(V^U, s)
23:
24: function excess(s, d) :
25:    return (V^U(s) − V^L(s)) − εγ^{−d}
```

where V_0^L and V_0^U are the initial lower and upper bounds.

6.4 Focused Real-Time Dynamic Programming (FRTDP)

FRTDP, Algorithm 6.4, was developed based both on RTDP and on our experience with HSVI. FRTDP is less myopic than HSVI because its outcome selection heuristic uses cached priority information to avoid fruitlessly revisiting states that resist improvement. FRTDP also has a maximum depth termination criterion to abort trials that run too long, helping it cut off fruitless trials in problems with "one-way doors", where an irreversible early decision can make it much more dif-

Algorithm 6.3 ConvergentHSVI, a driver for running HSVI in anytime fashion.

```
1: uses implementation <LB> conforming to UPDATABLESIG
2: uses implementation <UB> conforming to UPDATABLESIG
3:
4: function ConvergentHSVI() :
5:     V^L ← <LB>.initialValueFunction()
6:     V^U ← <UB>.initialValueFunction()
7:     ε ← ε_0
8:     loop:
9:         while V^U(s_0) − V^L(s_0) > ε :
10:            trialRecurse(s_0, d = 0)
11:        ε ← k_ε ε
12:
13: [other functions defined as for HSVI, see Algorithm 6.2]
```

ficult to reach a goal.

As with HSVI, when FRTDP is invoked with a non-zero regret bound ϵ, the algorithm focuses on forcing the excess uncertainty at s_0 to 0. However, FRTDP uses a slightly different definition of excess uncertainty that does not incorporate the discount factor.[5] The *FRTDP excess uncertainty* of a state s is

$$\Delta(s) := V^U(s) - V^L(s) - \epsilon/2, \tag{6.23}$$

and a state s is said to be *FRTDP-finished* if $\Delta(s) \leq 0$. (The $\epsilon/2$ term is motivated later.)

6.4.1 Search Graph Expansion

Recall that HSVI's heuristics for action and outcome selection were based on reasoning about the relationship between excess uncertainty at a state s and the excess uncertainty of its immediate successors s'. FRTDP uses a less myopic version of the same logic—it reasons about the relationship between excess uncertainty at a state and its more distant successors on the fringe of the *explicit graph*.

[5] Unfortunately, the convergence proof for FRTDP would not go through with HSVI's definition of excess uncertainty for technical reasons. The key difficulty is the fact that the HSVI notion of the "depth" of a state in the search graph depends on the path followed to reach the state. If a state were visited via multiple paths with different lengths and FRTDP excess uncertainty depended on depth, the priority information cached by FRTDP could be inconsistent from trial to trial.

Algorithm 6.4 Focused RTDP (FRTDP), a search strategy.

uses implementation <LB> **conforming to** UPDATABLESIG
uses implementation <UB> **conforming to** UPDATABLESIG

function FRTDP(ϵ) **:**
  $D \leftarrow D_0$
  $V^L \leftarrow$ <LB>.initialValueFunction()
  $V^U \leftarrow$ <UB>.initialValueFunction()
  while $V^U(s_0) - V^L(s_0) > \epsilon$ **:**
    $(q_{\text{prev}}, n_{\text{prev}}, q_{\text{curr}}, n_{\text{curr}}) \leftarrow (0, 0, 0, 0)$
    trialRecurse($s_0, W = 1, d = 0$)
    if $(q_{\text{curr}}/n_{\text{curr}}) + \zeta \geq (q_{\text{prev}}/n_{\text{prev}})$ **then** $D \leftarrow k_D D$
  return V^L

function trialRecurse(s, W, d) **:**
  $(a^*, s^+, \delta) \leftarrow$ update(s)
  trackUpdateQuality($\delta W, d$)
  if $\Delta(s) \leq 0$ **or** $d \geq D$ **then return**
  trialRecurse($s^+, \gamma T(s, a^*, s^+)W, d + 1$)
  update(s)

function update(s) **:**
  $a^* \leftarrow \arg\max_a (a \otimes V^U)(s)$
  $u \leftarrow V^U(s)$
  $V^L \leftarrow$ <LB>.update(V^L, s)
  $V^U \leftarrow$ <UB>.update(V^U, s)
  $\delta \leftarrow |V^U(s) - u|$
  $p_0, s^+ \leftarrow \max, \arg\max_{s' \in \mathcal{N}(s,a^*)} \gamma T(s, a^*, s')\, p(s')$
  $p(s) \leftarrow \min(\Delta(s), p_0)$
  return (a^*, s^+, δ)

function trackUpdateQuality(q, d) **:**
  if $d > D/k_D$ **then** $(q_{\text{curr}}, n_{\text{curr}}) \leftarrow (q_{\text{curr}} + q, n_{\text{curr}} + 1)$
  else $(q_{\text{prev}}, n_{\text{prev}}) \leftarrow (q_{\text{prev}} + q, n_{\text{prev}} + 1)$

function initNode(s) **:**
  [implicitly called the first time each state s is touched]
  $p(s) \leftarrow \Delta(s)$

function $\Delta(s)$ **:**
  return $(V^U(s) - V^L(s)) - \epsilon/2$

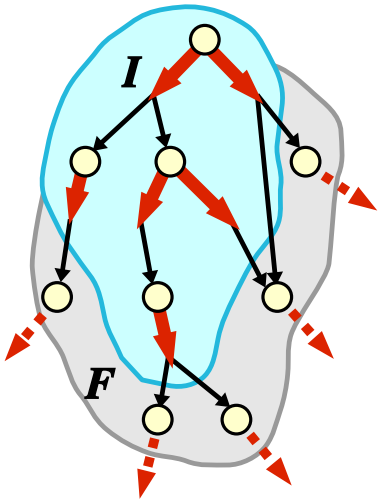


Figure 6.3: The explicit graph interior $\mathcal{I}$ and fringe $\mathcal{F}$.

The explicit graph is the explored part of the MDP search graph. It contains all of the states that the search algorithm has touched so far. Our planning algorithms assume the presence of a complete MDP model, but they can be applied regardless of how the model is represented—the MDP search graph may even be infinite. The only requirement is that the model specifies the initial state s_0 and provides a way to expand a state, discovering its immediate successors. The explicit graph can be thought of as a finite data structure that caches all the information the search algorithm has learned so far from the model via state expansions.

Expanding a state means generating its outgoing immediate rewards, transition probabilities, and successor states and adding them to the explicit graph. Furthermore, whenever a new successor state s is added, the search algorithm gets access to its initial bounds information $\hat{V}(s)$.

Explicit graph nodes are either *internal nodes*, which have already been expanded, or *fringe nodes*, which have not. Let $\mathcal{I}$ denote the set of internal nodes and $\mathcal{F}$ the set of fringe nodes. Figure 6.3 shows an example explicit graph partway through the search process. The dashed arrows emanating from the fringe nodes represent their outgoing transitions, which are not yet known to the search algorithm.

We can now recast the problem of designing action and outcome selection heuristics. Given a particular explicit graph, we want the heuristics to guide forward exploration towards the fringe state whose expansion has the largest potential impact on the excess uncertainty $\Delta(s_0)$ at the initial state s_0, and thus on the regret

bound of the policy PV^L that FRTDP returns. To do this, we need a framework for assessing the relative contributions of the fringe states to the uncertainty at s_0.

We can start by defining a statistic for the relevance of a state to a particular policy π. If we consider the distribution of possible execution traces for π starting from s_0 and interpret the discount γ in terms of trace termination (that is, execution terminates at any given time step with probability $1 - \gamma$), then the *occupancy*, $W^\pi(s_0, s)$, is the expected number of time steps per execution that π spends in state s before passing beyond the fringe into the unknown part of the graph.

Formally, occupancy is defined as the solution to the following simultaneous equations ($s' \in \mathcal{I} \cup \mathcal{F}$):

$$W^\pi(s_0, s') = \delta(s_0, s') + \gamma \sum_{s \in \mathcal{I} - \mathcal{G}} W^\pi(s_0, s) T(s, \pi(s), s') \tag{6.24}$$

where $\delta(x, y) = 1$ if $x = y$ and 0 otherwise.[6]

Let π be a policy that defines what action to take from any internal node in the search graph. Let π^+ be an *optimal extension* of π, a policy that uses π to select actions so long as execution stays within $\mathcal{I}$, but uses an optimal policy to select actions afterwards. We can calculate upper and lower bounds on the quality $J\pi^+(s_0)$ of policy π^+ as follows. Define

$$J^L\pi(s_0) := J^{\mathcal{I}}\pi(s_0) + \sum_{s \in \mathcal{F}} W^\pi(s_0, s) V^L(s) \tag{6.25}$$

$$J^U\pi(s_0) := J^{\mathcal{I}}\pi(s_0) + \sum_{s \in \mathcal{F}} W^\pi(s_0, s) V^U(s) \tag{6.26}$$

$$\hat{J}\pi(s_0) := \left[J^L\pi(s_0), J^U\pi(s_0) \right], \tag{6.27}$$

where $J^{\mathcal{I}}\pi(s_0)$ is the expected reward from executing π up to the point where it reaches a fringe node. Then $J\pi^+(s_0)$ must fall in the interval $\hat{J}\pi(s_0)$.

By breaking up the contributions to the value of π^+ from the known and unknown parts of the search graph, we can identify where uncertainty in the value of

[6] If $\gamma = 1$ and π has loops, the occupancy of some states may diverge. However, the occupancy converges for the problems and policies that interest us.

π^+ comes from:

$$\begin{aligned}\text{width}(\hat{J}\pi(s_0)) &= J^U\pi(s_0) - J^L\pi(s_0) && (6.28)\\ &= \sum_{s\in\mathcal{F}} W^\pi(s_0,s)V^U(s) - \sum_{s\in\mathcal{F}} W^\pi(s_0,s)V^L(s) && (6.29)\\ &= \sum_{s\in\mathcal{F}} W^\pi(s_0,s)\text{width}(\hat{V}(s)). && (6.30)\end{aligned}$$

The uncertainty at s_0 is seen to be a weighted sum over the fringe states, and the contribution of each fringe state s is $W^\pi(s_0,s)\text{width}(\hat{V}(s))$, "the occupancy times the uncertainty".

We now have a framework for assessing the contribution of a fringe state to the uncertainty *for a particular policy*. But recall that our real goal is to shrink the excess uncertainty $\Delta(s_0)$ that drives the regret bound. Intuitively, since V^L and V^U are bounds on the value of an optimal policy π^*, we would expect $\Delta(s_0)$ to be related to the occupancy of π^*, along the lines of the analysis above. Unfortunately, FRTDP does not have an optimal policy in hand, since that is exactly what it is trying to calculate!

However, recall from the discussion of HSVI that

$$\begin{aligned}\text{width}(H\hat{V}(s)) &\leq \text{width}((a^*\otimes\hat{V})(s)) && (6.31)\\ &= \sum_{s'} T(s,a^*,s')\text{width}(\hat{V}(s')). && (6.32)\end{aligned}$$

where a^* is chosen to optimize $(a\otimes V^U)(s)$. Abstractly, this inequality bounds the uncertainty about the quality of the (unknown) optimal policy in terms of the uncertainty about the greedy policy PV^U that follows the upper bound.

For our analysis of FRTDP, we rely on a relationship that can be viewed as a transitive extension of (6.32). Assume for the moment that the bounds functions V^L and V^U have been consistently extended from the fringe states into the interior of the explicit graph, locally satisfying the Bellman equation:

$$\begin{aligned}V^L(s) &= HV^L(s) && (6.33)\\ V^U(s) &= HV^U(s), && (6.34)\end{aligned}$$

for all $s\in\mathcal{I}$. This means that V^L would be the optimal value function on the interior of the explicit graph if we had $V^* = V^L$ on the fringe; a similar relation

holds for V^U. Thus, the corresponding one-step lookahead policies are optimal:

$$J^L PV^L(s) = \max_\pi J^L \pi(s) \tag{6.35}$$

$$J^U PV^U(s) = \max_\pi J^U \pi(s). \tag{6.36}$$

Furthermore, we know that performing one-step lookahead with an optimal value function achieves expected reward consistent with the value function:

$$J^L PV^L(s) = V^L(s) \tag{6.37}$$

$$J^U PV^U(s) = V^U(s). \tag{6.38}$$

Putting this together,

$$\begin{aligned}
\mathrm{width}(\hat{V}(s_0)) &= V^U(s_0) - V^L(s_0) && (6.39)\\
&= J^U PV^U(s_0) - J^L PV^L(s_0) && (6.40)\\
&\leq J^U PV^U(s_0) - J^L PV^U(s_0) && (6.41)\\
&= \mathrm{width}(\hat{J}PV^U(s_0)) && (6.42)\\
&= \sum_{s \in \mathcal{F}} W^{PV^U}(s_0, s)\mathrm{width}(\hat{V}(s)). && (6.43)
\end{aligned}$$

Thus the uncertainty about an optimal policy is related to the uncertainty about PV^U, and (6.43) is analogous to (6.32).

It also implies that if $\mathrm{width}(\hat{V}(s)) \leq \epsilon$ for all states $s \in \mathcal{F}$, then after enough updates to propagate this information back to s_0, we would have $\mathrm{width}(\hat{V}(s_0)) \leq \epsilon$, which is the overall algorithm termination criterion. Thus one might consider a state finished when its bounds width drops below ϵ.

However, we suggest that it may be easier to push *most* of the states on the fringe to a tighter bound of $\epsilon/2$, leaving some other fringe states to have somewhat wider bounds and still achieving the target of ϵ at s_0. This motivates the $\epsilon/2$ term in the definition of $\Delta(s)$.

An identity analogous to (6.43) also holds for the excess uncertainty:

$$\Delta(s_0) \leq \sum_{s \in \mathcal{F}} W^{PV^U}(s_0, s)\Delta(s), \tag{6.44}$$

which suggests that the fringe node with greatest potential to impact $\Delta(s_0)$ can be

chosen according to

$$s^* = \arg\max_{s \in \mathcal{F}} W^{PV^U}(s_0, s)\Delta(s). \tag{6.45}$$

In subsequent sections we will use the shorthand W^U to refer to W^{PV^U}.

6.4.2 FRTDP Outcome Selection

FRTDP tries to reach the s^* that maximizes (6.45) by choosing the right successor at each forward step of exploration. Assume for the moment that the search graph is a tree. Then, at any step of forward exploration, each immediate successor is the root of a subtree, and the problem reduces to selecting the successor whose subtree contains s^*.

This section defines the priority $p(s)$ of a state s. We show that (1) the priority can be calculated and cached recursively in parallel with value function updates, and (2) in the case that the search graph is a tree, if we choose the successor state according to weighted priority, the corresponding subtree contains s^*.

For any immediate successor s', let $\mathcal{F}^{s'}$ indicate the subset of fringe states contained in the subtree rooted at s'. Then if forward exploration is currently at state s, the path leading to s^* must go through the immediate successor s^+ satisfying

$$s^+ = \arg\max_{s' \in \mathcal{N}(s,a)} \max_{s'' \in \mathcal{F}^{s'}} W^U(s_0, s'')\Delta(s''). \tag{6.46}$$

In a tree there is only one path to reach any given fringe state s'', and if $s'' \in \mathcal{F}^{s'}$, that path must lead through s and s', leading to the identity

$$\begin{aligned} W^U(s_0, s'') &= W^U(s_0, s)W^U(s, s')W^U(s', s'') && (6.47)\\ &= W^U(s_0, s)\left[\gamma T(s, PV^U(s), s')\right]W^U(s', s''). && (6.48)\\ & && (6.49) \end{aligned}$$

Define the *priority* of a state s' to be

$$p(s') := \max_{s'' \in \mathcal{F}^{s'}} W^U(s', s'')\Delta(s''). \tag{6.50}$$

This allows us to decompose (6.46) into

$$s^+ = \arg \max_{s' \in \mathcal{N}(s,a)} \max_{s'' \in \mathcal{F}^{s'}} W^U(s_0, s'')\Delta(s'') \tag{6.51}$$

$$= \arg \max_{s' \in \mathcal{N}(s,a)} \max_{s'' \in \mathcal{F}^{s'}} W^U(s_0, s) \left[\gamma T(s, PV^U(s), s')\right] W^U(s', s'')\Delta(s'') \tag{6.52}$$

$$= \arg \max_{s' \in \mathcal{N}(s,a)} W^U(s_0, s) \left[\gamma T(s, PV^U(s), s')\right] \max_{s'' \in \mathcal{F}^{s'}} W^U(s', s'')\Delta(s'') \tag{6.53}$$

$$= \arg \max_{s' \in \mathcal{N}(s,a)} W^U(s_0, s) \left[\gamma T(s, PV^U(s), s')\right] p(s') \tag{6.54}$$

$$= \arg \max_{s' \in \mathcal{N}(s,a)} \gamma T(s, PV^U(s), s')p(s'), \tag{6.55}$$

where we can drop the factor $W^U(s_0, s)$ in the last step because it does not depend on s' and must be strictly positive since s was visited. With similar reasoning about decomposition one can show that the priority satisfies the recursion

$$p(s) = \begin{cases} \max_{s' \in \mathcal{N}(s,a)} \gamma T(s, PV^U(s), s')p(s') & s \in \mathcal{I} \\ \Delta(s) & s \in \mathcal{F}. \end{cases} \tag{6.56}$$

Every time FRTDP performs a point-based update at s it also calculates and caches the value of $p(s)$ according to the following slightly modified formula,[7] which reduces to (6.56) when the search graph is a tree but has better properties for general graphs:

$$p(s) = \begin{cases} \min(\Delta(s), \max_{s' \in \mathcal{N}(s,a)} \gamma T(s, PV^U(s), s')p(s')) & s \in \mathcal{I} \\ \Delta(s) & s \in \mathcal{F}. \end{cases} \tag{6.57}$$

Recall that FRTDP performs round-trip updating as the last phase of each trial, retracing and updating the states it visited during forward exploration. This ensures that, under the assumption that the graph is a tree, at the end of each trial all states in the interior have $p(s)$ values that are up to date with respect to their successors. Likewise, during forward exploration, when s^+ is being selected, all immediate successors s' have up-to-date $p(s')$ values. Thus, despite considering only immediate successors during priority updates and s^+ selection, FRTDP is guaranteed to reach the fringe state s^* that globally maximizes (6.45).

However, in general graphs, FRTDP forward exploration is no longer guaran-

[7] In practice, priority values can be very small. Our implementation of FRTDP calculates $\log(p(s))$ rather than $p(s)$ in order to avoid floating-point underflow.

teed to reach s^*, for two main reasons:

1. $W^U(s_0, s'')$ is the expected number of times execution reaches s'', *adding up the likelihood over all possible paths* from s_0 to s''. Calculating $p(s)$ according to the recursion (6.57) in effect prioritizes each fringe state s'' according to the likelihood of the *single most likely path* to s''.

   In a tree the total occupancy and single-path occupancy are equal. In a general graph, especially one with a large amount of outcome uncertainty, the occupancy of a state deep in the search graph is likely to be spread over an exponential number of distinct paths, making the single-path occupancy much smaller than the total. However, we expect that in most problems drawn from real applications the same fringe states should tend to have high occupancy according to both measures, even if the measures give very different values numerically.

2. In a general graph, some $p(s)$ values may not be up to date with respect to their successors at the end of the trial. This is because a single state updated during forward exploration may have many immediate predecessors, and round-trip updating visits only the predecessors that were on the path followed by forward exploration. However, this type of inconsistency is typically temporary, as the other predecessors are visited and have their priorities updated during later trials.

Thus, when FRTDP uses local priority calculations to explore general graphs it is no longer guaranteed to reach the highest impact fringe state s^*, but its heuristic choices are still likely to be reasonable. Furthermore, it is still guaranteed to converge (see §6.5.2).

6.4.3 Adaptive Maximum Depth Termination

With the excess uncertainty trial termination alone, FRTDP is a usable search algorithm. However, as with RTDP, poor outcome selection early in a trial could lead into a quagmire of irrelevant states that takes a long time to escape.

FRTDP's *adaptive maximum depth* (AMD) trial termination criterion mitigates this problem by cutting off long trials. FRTDP maintains a *current maximum depth* D. A trial is terminated if it reaches depth $d \geq D$. FRTDP initializes D to a small value D_0, and increases it for subsequent trials. The idea is to avoid overcommitting to long trials early on, but retain the ability to go deeper in later trials, in case there are relevant states deeper in the search graph.

FRTDP performance for any particular problem depends on how D is adjusted, so it is important that whatever technique is used be relatively robust across prob-

	RTDP-solvable	Discounted finite-branching
RTDP	converges, no rate specified	probably fails in some cases
HSVI	known to fail in some cases	term., regret $\leq \epsilon$, time bound
FRTDP-AMD	term., regret $\leq \epsilon$, no time bound	probably fails in some cases

Table 6.2: Search strategy termination conditions

lems without manual parameter tuning. We chose to adjust D adaptively, using trial statistics as feedback. After each trial, FRTDP chooses whether to keep the current value of D or to increase it, multiplying by a factor of $k_D > 1$.

The feedback mechanism is fairly ad hoc. Each update in a trial is given an *update quality score* $q = \delta W$ that is intended to reflect how useful the update was. δ measures how much the update changed the upper bound value $V^U(s)$.[8] W is a single-path estimate of the occupancy of the state being updated under the current greedy policy. After each trial, D is increased by default, but remains the same if the average update quality near the end of the trial ($d > D/k_D$) is smaller than the average in the earlier part of the trial, with the difference exceeding a small margin of error $\zeta > 0$.[9] Refer to the pseudo-code of Algorithm 6.4 for details. We used the parameter values $D_0 = 10$, $k_D = 1.1$, and $\zeta = 10^{-5}$ for all the experiments reported here.

6.5 Theoretical Results

Although they were presented as integrated implementations including both heuristic search state selection and a main loop, the HSVI and FRTDP algorithms can be viewed as heuristic search modules fitting into the focused value iteration algorithm schema. Thus for both algorithms Theorem 3.30 applies. If the algorithm terminates, its returned policy PV^L has regret bounded to at most ϵ.

The rest of this section is devoted to analyzing the conditions required for HSVI and FRTDP termination. The results are summarized in Table 6.2. Note that the algorithms have somewhat different notions of termination and convergence.

RTDP is designed to be run as an anytime algorithm. Applied to an RTDP-

[8] The change in the regret could also be used, but the change in the upper bound is slightly faster to calculate.

[9] The version of AMD presented in Smith and Simmons (2006) did not include the margin of error ζ, in effect setting $\zeta = 0$. We now use a small positive value for ζ to ensure that the maximum depth continues to increase in near-equilibrium situations where the average update quality is very small throughout the run and round-off may be an issue. This minor change does not appear to affect performance in practice.

solvable MDP, its value function converges to V^* over the subset of relevant states, but the convergence proof in Barto et al. (1995) does not estimate the rate of convergence, nor is there a built-in way to monitor convergence during execution so that the algorithm can be terminated when it reaches a desired precision.

In contrast, HSVI and FRTDP include built-in monitoring of convergence and terminate when the regret of the returned policy is bounded to a specified ϵ. In addition, our proof of HSVI convergence for discounted finite-branching problems provides an *a priori* upper bound on the number of updates required for the regret bound to reach a particular precision ϵ.

6.5.1 HSVI Termination

The following analysis of HSVI leads up to the main result, Theorem 6.8, which shows that HSVI(ϵ) terminates in bounded time when applied to a discounted finite-branching problem. Theorem 6.9 is a corresponding negative result, showing that HSVI is *not* guaranteed to terminate when applied to an undiscounted RTDP-solvable problem.

Lemma 6.1. *Let K be a strong point-based update operator, let V^L and V^U be uniformly improvable value functions, and let $a^* = \arg\max_a (a \otimes V^U)(s)$. Then*

$$\operatorname{width}(K_s \hat{V}(s)) \leq \gamma \sum_{s'} T(s, a^*, s') \operatorname{width}(\hat{V}(s')). \tag{6.58}$$

Proof. We have

$$\begin{aligned}
&\operatorname{width}(K_s \hat{V}(s)) && && (6.59)\\
&= \operatorname{width}(H \hat{V}(s)) && [K_s \text{ is strong}] && (6.60)\\
&= HV^U(s) - HV^L(s) && [\text{def. of } H\hat{V}] && (6.61)\\
&= \max_a (a \otimes V^U)(s) - \max_a (a \otimes V^L)(s) && [\text{def. of } H] && (6.62)\\
&= (a^* \otimes V^U)(s) - \max_a (a \otimes V^L)(s) && [\text{def. of } a^*] && (6.63)\\
&\leq (a^* \otimes V^U)(s) - (a^* \otimes V^L)(s) && && (6.64)\\
&= \gamma \sum_{s'} T(s, a^*, s') \, (V^U(s) - V^L(s')) && [\text{def. of } \otimes] && (6.65)\\
&= \gamma \sum_{s'} T(s, a^*, s') \operatorname{width}(\hat{V}(s')). && [\text{def. of } \hat{V}] && (6.66)
\end{aligned}$$

□

Lemma 6.2. *Under the conditions of Lemma 6.1,*

$$\text{excess}(K_s\hat{V}, s, d) \leq \gamma \sum_{s'} T(s, a^*, s') \text{ excess}(\hat{V}, s', d+1) \tag{6.67}$$

Proof. We have

$$\begin{aligned}
&\text{excess}(K_s\hat{V}, s, d) && && (6.68)\\
&= \text{width}(K_s\hat{V}(s)) - \epsilon\gamma^{-d} && \text{[def. of excess]} && (6.69)\\
&\leq \gamma \sum_{s'} T(s, a^*, s') \text{ width}(\hat{V}(s')) - \epsilon\gamma^{-d} && \text{[Lemma 6.1]} && (6.70)\\
&= \gamma \left[\sum_{s'} T(s, a^*, s') \text{ width}(\hat{V}(s')) - \epsilon\gamma^{-(d+1)} \right] && && (6.71)\\
&= \gamma \sum_{s'} T(s, a^*, s') \ \left[\text{width}(\hat{V}(s')) - \epsilon\gamma^{-(d+1)} \right] && \left[\sum_{s'} T(s, a^*, s') = 1 \right] && (6.72)\\
&= \gamma \sum_{s'} T(s, a^*, s') \text{ excess}(\hat{V}, s', d+1). && \text{[def. of excess]} && (6.73)
\end{aligned}$$

□

Lemma 6.3. *Under the conditions of Lemma 6.1, if all successors $s' \in \mathcal{N}(s, a^*)$ are finished at depth $d+1$, then after applying a strong update operator K_s, state s will be finished at depth d.*

Proof. We have

$$\begin{aligned}
&\text{excess}(K_s\hat{V}, s, d) && && (6.74)\\
&\leq \gamma \sum_{s'} T(s, a^*, s') \cdot \text{excess}(\hat{V}, s', d+1) && \text{[Lemma 6.2]} && (6.75)\\
&\leq \gamma \sum_{s'} T(s, a^*, s') \cdot 0 && \text{[def. of finished]} && (6.76)\\
&= 0. && && (6.77)
\end{aligned}$$

□

Lemma 6.4. *HSVI always chooses an unfinished successor if there is one. Specifically, if HSVI encounters a state s at depth d and there is a state $s' \in \mathcal{N}(s, a^*)$*

that is not finished at depth $d+1$, then the successor

$$s^* := \arg\max_{s'} T(s, a^*, s') \operatorname{excess}(\hat{V}, s', d+1) \tag{6.78}$$

chosen by HSVI is not finished at depth $d+1$.

Proof. The heuristic maximized when s^* is chosen is positive only for successor states that are unfinished at depth $d+1$. □

Lemma 6.5. *Let $\epsilon > 0$ and let HSVI(ϵ) be applied to a problem satisfying conditions H1-H2. Let V^L, V^U be bounded value functions. Then all states are finished at every depth $d \geq d_{\max}$, where*

$$d_{\max} = \lceil \log_\gamma(\epsilon / \left\| V^U - V^L \right\|_\infty) \rceil. \tag{6.79}$$

Proof. Let s be a state, let $d \geq d_{\max}$, and define the shorthand $W = \left\| V^U - V^L \right\|_\infty$. We have

$$\begin{aligned}
d &\geq \lceil \log_\gamma(\epsilon/W) \rceil && && (6.80)\\
d &\geq \log_\gamma(\epsilon/W) && && (6.81)\\
\gamma^d &\leq \epsilon/W && [x \mapsto \gamma^x \text{ monotone decreasing}] && (6.82)\\
W - \epsilon\gamma^{-d} &\leq 0 && && (6.83)\\
\operatorname{width}(\hat{V}(s)) - \epsilon\gamma^{-d} &\leq 0 && [W = \max_s \operatorname{width}(\hat{V}(s))] && (6.84)\\
\operatorname{excess}(\hat{V}, s, d) &\leq 0. && [\text{def. of excess}] && (6.85)
\end{aligned}$$

□

Corollary 6.6. *When HSVI(ϵ) is applied under the conditions of Lemma 6.5, every HSVI trial reaches a finished state and terminates after visiting at most $d_{\max}$ states. Each state visited in the trial is updated once during forward exploration and again during round-trip updating, so a trial requires at most $2d_{\max}$ point-based updates.*

Lemma 6.7. *If the HSVI main loop has not yet terminated, then executing a complete HSVI trial will cause at least one state that is not finished to become finished. Specifically, for some d, a state that was not finished at depth d will become finished at depth d.*

Proof. Suppose HSVI has not yet terminated and a trial is executed. Let the final two states encountered during forward exploration be s and s', encountered at depth d and $d+1$, respectively. Because forward exploration terminated at s' but did not

terminate at s, it must be the case that before the trial s' was finished at depth $d+1$ but s was not finished at depth d.

Because the finished state s' was chosen by the outcome selection heuristic, we can infer that all successors in $\mathcal{N}(s, a^*)$ were finished at depth $d+1$, since by Lemma 6.4 HSVI always chooses an unfinished successor if there is one. Applying Lemma 6.3, state s will be finished at depth d after it is updated.

Thus executing the trial causes state s, which was not finished at depth d, to become finished at depth d. □

Theorem 6.8. *Let $\epsilon > 0$ and let HSVI(ϵ) be applied to an MDP satisfying H1-H2 with branching factor j. Let the initial bounds V_0^L, V_0^U be globally bounded and uniformly improvable. Let*

$$d_{\max} = \lceil \log_\gamma(\epsilon / \left\| V_0^U - V_0^L \right\|_\infty) \rceil. \tag{6.86}$$

as in Lemma 6.5, and let

$$\textit{nodeCount}(j, d) := \frac{j^{d+1} - 1}{j - 1}. \tag{6.87}$$

Then HSVI will terminate after at most nodeCount$(j, d_{\max} - 1)$ trials, each of which involves at most $2d_{\max}$ point-based updates.

Proof. Begin with the case that the search graph is a tree, meaning there is only one path to reach any given state, and each state has a fixed depth in the tree. In that case, one can say that a state s that appears at fixed depth d is "finished" as shorthand for "finished at depth d".

Applying Lemma 6.5, only states that appear at depth $d < d_{\max}$ can be initially unfinished. Since the tree has branching factor j, the number of states at depth d is j^d, and the total number of states at all depths up to $d_{\max}$ is

$$\text{nodeCount}(j, d_{\max} - 1) = \sum_{d=0}^{d_{\max}-1} j^d = \frac{j^{d_{\max}} - 1}{j - 1}. \tag{6.88}$$

Lemma 6.7 states that at least one unfinished state becomes finished after each trial. Because the value function bounds are uniformly improvable and are modified only by point-based updates, the bounds interval at each state can only grow narrower and finished states never become unfinished. Thus the initial state must become finished after at most nodeCount$(j, d_{\max} - 1)$ trials, causing HSVI to terminate.

In the general case the search graph may not be a tree, but this can only help HSVI. The formula for nodeCount$(j, d_{\max} - 1)$ counts the number of paths through

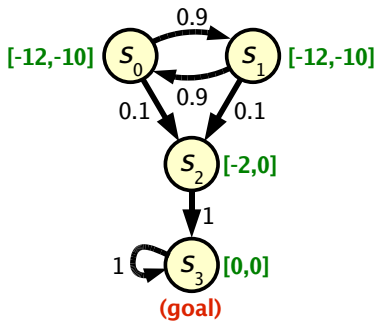


Figure 6.4: An undiscounted MDP which causes HSVI to enter an infinite loop.

the graph that start at s_0 and have length $d < d_{\max}$. If the graph is a tree, each distinct path leads to a distinct state, but in a general graph distinct paths may lead to the same state, effectively reducing the total number of states that need to be finished. □

Theorem 6.9. *In the absence of conditions H1-H2, HSVI trials are* not *guaranteed to terminate, even if the RTDP convergence conditions R1-R3 are met, $\epsilon > 0$, and the initial bounds V^L, V^U are globally bounded and uniformly improvable.*

Proof. We provide a specific MDP and initial bounds satisfying the conditions of the theorem, and show that a trial of HSVI will fall into an infinite loop.

Consider an undiscounted MDP with four states $s_0, \ldots, s_3$ and one action a, shown in Figure 6.4. Let s_0 be the initial state and s_3 be a goal state. For every non-goal state s, the action a incurs cost $R(s, a) = -1$. From state s_0, the possible transitions are $T(s_0, a, s_1) = 0.9$ and $T(s_0, a, s_2) = 0.1$. From state s_1, the possible transitions are $T(s_1, a, s_0) = 0.9$ and $T(s_1, a, s_2) = 0.1$. From state s_2, the only possible transition is $T(s_2, a, s_3) = 1$.

The initial bounds are set up as follows. Let $\hat{V}(s_0) = \hat{V}(s_1) = [-12, -10]$, $\hat{V}(s_2) = [-2, 0]$, and $\hat{V}(s_3) = [0, 0]$. One can verify that updating state s_2 would cause its bounds to collapse to $[-1, -1]$, but updating any other single state would leave its bounds unchanged. In no case would an update weaken the lower or upper bound, so these bounds are uniformly improvable.

We show that a trial of HSVI(ϵ) with $\epsilon = 0.1$ beginning at the initial state s_0 will fall into an infinite loop between states s_0 and s_1. Since neither state is finished and an update at either leaves the bounds unchanged, we only need to show that, given the specified bounds, forward exploration from state s_0 will visit state s_1, and vice versa.

For this MDP, HSVI's action selection heuristic is irrelevant since the MDP has only one action. From state s_0, only states s_1 and s_2 are reachable. Since $\gamma = 1$, $\text{excess}(\hat{V}, s, d)$ is independent of d. We have

$$T(s_0, a, s_1) = 0.9 \tag{6.89}$$

$$\text{excess}(\hat{V}, s_1, d) = (-10 - (-12)) - 0.1 = 1.9 \tag{6.90}$$

$$T(s_0, a, s_2) = 0.1 \tag{6.91}$$

$$\text{excess}(\hat{V}, s_2, d) = (0 - (-2)) - 0.1 = 1.9, \tag{6.92}$$

so from state s_0, the outcome selection heuristic based on weighted excess uncertainty will prefer state s_1 to s_2. States s_0 and s_1 are symmetric, so the same argument shows that from state s_1 forward exploration will choose s_0. □

6.5.2 FRTDP Termination

This section discusses FRTDP termination. In Smith and Simmons (2006), we claimed that FRTDP was guaranteed to converge over the class of RTDP-solvable MDPs. Unfortunately, we later realized that this result was somewhat overstated. More detailed study showed that there was a gap in our analysis of the adaptive maximum depth (AMD) termination criterion. At this point, we can show guaranteed termination only when maximum depth termination is not used. We call this variant FRTDP-AMD, as opposed to the usual variant FRTDP+AMD. Nonetheless, we continue to conjecture (though we cannot yet prove) that FRTDP+AMD always converges.

The results in this section build up to Theorem 6.16, which shows that FRTDP-AMD terminates when applied to an RTDP-solvable problem. The section ends with a discussion of the gap that remains in proving termination for FRTDP+AMD.

Lemma 6.10. *Under the conditions of Lemma 6.1, after an update at s,*

$$\Delta(s) \leq \gamma \sum_{s'} T(s, a^*, s')\ \Delta(s') \tag{6.93}$$

Proof. We have

$$\Delta(s) \tag{6.94}$$
$$= \text{width}(K_s \hat{V}(s)) - \epsilon/2 \qquad \text{[def. of } \Delta\text{]} \tag{6.95}$$
$$\leq \gamma \sum_{s'} T(s, a^*, s') \text{ width}(\hat{V}(s')) - \epsilon/2 \qquad \text{[Lemma 6.1]} \tag{6.96}$$
$$\leq \gamma \left[\sum_{s'} T(s, a^*, s') \text{ width}(\hat{V}(s')) - \epsilon/2 \right] \tag{6.97}$$
$$= \gamma \sum_{s'} T(s, a^*, s') \left[\text{width}(\hat{V}(s')) - \epsilon/2 \right] \quad \left[\sum_{s'} T(s, a^*, s') = 1 \right] \tag{6.98}$$
$$= \gamma \sum_{s'} T(s, a^*, s') \, \Delta(s'). \qquad \text{[def. of } \Delta\text{]} \tag{6.99}$$

□

Lemma 6.11. *Under the conditions of Lemma 6.1, if all successors $s' \in \mathcal{N}(s, a^*)$ are FRTDP-finished, then after applying a strong update operator K_s, state s will be FRTDP-finished.*

Proof. Parallels the proof of the corresponding result for HSVI, Lemma 6.3, replacing "finished" with "FRTDP-finished". □

Lemma 6.12. *The priority of a state s is negative if and only if s is FRTDP-finished.*

Proof. To prove equivalence we must prove that the implication is true in both directions:

1. $\Delta(s) \leq 0$ *implies* $p(s) \leq 0$.

   The value $p(s)$ can be set in either of two ways:

   $$p(s) \leftarrow \min(\Delta(s), p_0) \qquad \text{[Alg. 6.4 line 28]} \tag{6.100}$$
   $$p(s) \leftarrow \Delta(s). \qquad \text{[Alg. 6.4 line 37]} \tag{6.101}$$

   In both cases, $p(s) \leq \Delta(s)$, so the implication holds.

2. $p(s) \leq 0$ *implies* $\Delta(s) \leq 0$.

   Proof by backwards induction from the frontier. When a state s on the frontier is first touched, the implication holds because $p(s) = \Delta(s)$.

Given an interior state s, the induction hypothesis is that the implication holds for every successor $s' \in \mathcal{N}(s, a^*)$. Suppose $p(s) \leq 0$ after s is updated. From FRTDP lines 27,28 we see that $p(s)$ was set by

$$p(s) \leftarrow \min(\Delta(s), \arg \max_{s' \in \mathcal{N}(s,a^*)} \gamma T(s, a^*, s')\, p(s')). \tag{6.102}$$

If $\Delta(s)$ is the smaller argument to $\min$, then $\Delta(s) = p(s) \leq 0$ and the implication holds.

Otherwise the second argument to $\min$ is equal to $p(s)$ and is therefore negative. Since γ and the $T(\cdot)$ transition probabilities are strictly positive, we find that all the $p(s')$ values must likewise be negative. By the induction hypothesis, all the successors must satisfy $\Delta(s') \leq 0$, so applying Lemma 6.11, after the update $\Delta(s) \leq 0$ and the implication holds.

□

Lemma 6.13. *FRTDP always chooses an FRTDP-unfinished successor if there is one. Specifically, if FRTDP encounters a state s and there is a state $s' \in \mathcal{N}(s, a^*)$ that is FRTDP-unfinished, then the successor*

$$s^+ := \arg \max_{s'} T(s, a^*, s')\, p(s') \tag{6.103}$$

chosen by FRTDP is FRTDP-unfinished.

Proof. Applying Lemma 6.12, the heuristic maximized when s^+ is chosen is positive only for successor states that are FRTDP-unfinished. □

Lemma 6.14. *If the main loop has not yet terminated, then executing a complete trial of FRTDP-AMD(ϵ) causes at least one state that was not FRTDP-finished to become FRTDP-finished.*

Proof. Parallels the proof of the corresponding result for HSVI, Lemma 6.7. □

Lemma 6.15. *Let $\epsilon > 0$ and let FRTDP-AMD(ϵ) be applied to an MDP satisfying RTDP convergence conditions R1-R3. Suppose the initial bounds V_0^L, V_0^U are uniformly improvable. Then every trial terminates.*

Proof. Proof by contradiction. Suppose there is a trial that never ends. Let $\mathcal{Y} = \{s_0, a_0, s_1, a_1, \ldots\}$ be the sequence of states and actions chosen by FRTDP forward exploration.

Since $\mathcal{S}$ is finite by R1 but $\mathcal{Y}$ is infinite, some subset $\mathcal{S}^I \subseteq \mathcal{S}$ of the states must appear infinitely often in $\mathcal{Y}$. Furthermore, for each state $s \in \mathcal{S}^I$, some subset $U(s) \subseteq \mathcal{A}$ of the actions must be applied from state s infinitely often in $\mathcal{Y}$.

States not in $\mathcal{S}^I$ appear only finitely often in $\mathcal{Y}$, so after some number of steps all their appearances must be exhausted. Similarly, after some number of steps all appearances of actions $a_t \notin U(s_t)$ must be exhausted. Therefore, we can find a time T such that for all $t \geq T$, $s_t \in \mathcal{S}^I$ and $a_t \in U(s_t)$.

Since FRTDP forward exploration terminates on reaching an FRTDP-finished state and all goal states are FRTDP-finished, there must not be any goal states in $\mathcal{S}^I$. However, by R2 there is a proper policy, so there must be at least one "escape route" that leads out of $\mathcal{S}^I$ and eventually reaches a goal. In other words, there must be states $s_x \in \mathcal{S}^I$, $s_y \notin \mathcal{S}^I$ and an action a_x such that $T(s_x, a_x, s_y) > 0$.

For at least one of these escape routes, the action a_x must be chosen infinitely often from s_x. Otherwise, forward exploration would be selecting actions according to a policy with no chance of reaching a goal; that is, an improper policy. Applying R3, the improper policy would incur infinite cost for at least one state, and the $V^U(s)$ values for states $s \in \mathcal{S}^I$ would diverge to $-\infty$. But this contradicts the invariant $V^U(s) \geq V^*(s)$, which is guaranteed by the fact that V^U is uniformly improvable and modified only by conservative updates.

Thus there must be s_x, a_x, s_y defined as above such that the escape route action a_x is chosen infinitely often from s_x. The rest of the proof analyzes FRTDP outcome selection to show that under our earlier assumption that exploration never escapes $\mathcal{S}^I$ (meaning outcome s_y is never chosen), we arrive at a contradiction.

Define $p_t(s)$ to be the FRTDP priority of state s at time t, and define the total priority

$$F_t = \sum_{s \in \mathcal{S}^I} p_t(s). \tag{6.104}$$

At time t, FRTDP updates the priority of s_t according to[10]

$$p_{t+1}(s_t) = \gamma T(s_t, a_t, s_{t+1}) p_t(s_{t+1}). \tag{6.105}$$

It follows that

$$F_{t+1} = F_t - p_t(s_t) + \gamma T(s_t, a_t, s_{t+1}) p_t(s_{t+1}). \tag{6.106}$$

Regrouping terms to facilitate cancellation, define

$$G_t = F_t - p_t(s_t). \tag{6.107}$$

The behavior of G_t depends on whether $s_t = s_{t+1}$. If so, $G_{t+1} = G_t$. If not, since

[10]Note that although most of the complexity in this proof comes from making it work in the undiscounted case ($\gamma = 1$), the argument goes through unmodified for discounted problems.

state s_{t+1} is not updated at time t, we have $p_{t+1}(s_{t+1}) = p_t(s_{t+1})$ and

$$\begin{aligned} G_{t+1} &= F_{t+1} - p_{t+1}(s_{t+1}) && (6.108)\\ &= F_{t+1} - p_t(s_{t+1}) && (6.109)\\ &= F_t - p_t(s_t) + \gamma T(s_t, a_t, s_{t+1}) p_t(s_{t+1}) - p_t(s_{t+1}) && (6.110)\\ &= G_t + \gamma T(s_t, a_t, s_{t+1}) p_t(s_{t+1}) - p_t(s_{t+1}) && (6.111)\\ &= G_t - (1 - \gamma T(s_t, a_t, s_{t+1})) p_t(s_{t+1}). && (6.112) \end{aligned}$$

Since priorities are positive for FRTDP-unfinished states, we see that G_t monotonically decreases.

Suppose for some $t > T$ and some state $s^* \in \mathcal{S}^I$, $s^* \neq s_x$, we have $s_t = s_x$, $a_t = a_x$, and $s_{t+1} = s^*$. Define $\delta = 1 - \gamma T(s_x, a_x, s^*)$. $T(s_x, a_x, s_y) > 0$ implies $\delta > 0$. Since FRTDP outcome selection prefers s^* to s_y, we must have

$$\begin{aligned} T(s_x, a_x, s^*) p_t(s^*) &\geq T(s_x, a_x, s_y) p_t(s_y) && (6.113)\\ p_t(s^*) &\geq \frac{T(s_x, a_x, s_y)}{T(s_x, a_x, s^*)} p_t(s_y). && (6.114) \end{aligned}$$

But $s_y \notin \mathcal{S}^I$, so s_y is not updated after time T and its priority $p_t(s_y) = p_T(s_y)$ remains constant. Putting it all together,

$$\begin{aligned} G_{t+1} &= G_t - (1 - \gamma T(s_t, a_t, s_{t+1})) p_t(s_{t+1}) && (6.115)\\ &= G_t - (1 - \gamma T(s_x, a_x, s^*)) p_t(s^*) && (6.116)\\ &= G_t - \delta p_t(s^*) && (6.117)\\ &\leq G_t - \delta \frac{T(s_x, a_x, s_y)}{T(s_x, a_x, s^*)} p_T(s_y) && (6.118) \end{aligned}$$

Thus, each time the sequence s_x, a_x, s^* appears in $\mathcal{Y}$ at $t > T$, G_t decreases by at least the constant value in the second term of (6.118), which is strictly greater than zero.

The rest of the argument breaks down into two cases.

1. If there are at least two states in $\mathcal{S}^I$, there must be a state $s^* \neq s_x$ as above such that the sequence s_x, a_x, s^* appears infinitely frequently in $\mathcal{Y}$. This would imply that the modified total priority G_t diverges to $-\infty$, which is impossible, so we have reached a contradiction.

2. In the simpler case where there is only one state $s_x \in \mathcal{S}^I$, one can verify that $p_t(s_x)$ shrinks exponentially until it drops below zero, leading to an escape from $\mathcal{S}^I$ and a contradiction.

□

Theorem 6.16. *Let $\epsilon > 0$ and let FRTDP-AMD(ϵ) be applied to an MDP satisfying RTDP convergence conditions R1-R3. Suppose the initial bounds V_0^L, V_0^U are uniformly improvable. Then FRTDP-AMD(ϵ) terminates after at most $|\mathcal{S}|$ trials.*

Proof. By Lemma 6.15, every trial terminates. By Lemma 6.14, each trial causes at least one state that was not FRTDP-finished to become FRTDP-finished. Therefore it takes at most $|\mathcal{S}|$ trials to FRTDP-finish all the states. When the initial state s_0 is FRTDP-finished, the overall algorithm terminates. □

Lemma 6.17. *Using FRTDP+AMD under the conditions of Theorem 6.16, the maximum depth value D increases without bound as long as FRTDP+AMD continues running.*

Proof. AMD increases D unless, at the end of the trial, the average value of $W\delta$ over the "shallow states" near the beginning of the trial ($d < D/k_D$) exceeds the average over deep states later in the trial by at least ζ.

Both W and δ are always positive, so the average $W\delta$ over the deep states is at least zero, meaning that in order for D not to be increased, the average $W\delta$ over the shallow states must be at least ζ. Since the estimated occupancy W for any state is always less than 1, this implies that the average δ over the shallow states must be at least ζ, and thus at least one of the shallow states must have $\delta \geq \zeta$.

But for any state s, δ measures the decrease in $V^U(s)$ caused by updating s. Since V^U is uniformly improvable, $V^U(s)$ monotonically decreases and is bounded below by $V^*(s)$. This means the number of times s can have have a decrease $\delta \geq \zeta$ is bounded by

$$n(s) = \frac{V_0^U(s) - V^*(s)}{\zeta}, \tag{6.119}$$

where $V_0^U(s)$ is the initial upper bound at s. In turn, the total number of times that a decrease $\delta \geq \zeta$ can occur is bounded by

$$N = \sum_{s \in \mathcal{S}} n(s), \tag{6.120}$$

which is finite since $\mathcal{S}$ is finite. Thus FRTDP+AMD can perform at most N trials without increasing the maximum depth D. □

Conjecture 6.18. *Under the conditions of Theorem 6.16 FRTDP+AMD is guaranteed to terminate.*

Proof sketch. FRTDP+AMD trials fall into two classes: *normally terminated trials* that end because $\Delta(s) \leq 0$, and *AMD-terminated trials* that end because $d \geq D$. By Lemma 6.14, every normally terminated trial causes at least one state to become FRTDP-finished. However, AMD-terminated trials do not necessarily finish any states. Therefore it suffices to show that it is impossible to have an infinite number of consecutive AMD-terminated trials.

Lemma 6.15 showed that when AMD is not used, every trial eventually terminates. However, it does not say anything about the maximum number of states a trial can visit. Without this information we cannot tell whether trials will be AMD terminated or normally terminated.

Lemma 6.17 showed that the current maximum depth D increases without bound as long as FRTDP+AMD continues running. Intuitively, this suggests that after enough trials the maximum depth D will eventually surpass the depth at which an FRTDP-finished state would be encountered, triggering normal termination.

The only alternative is that successive AMD-terminated trials modify the bounds values in such a way that the depth at which normal termination would occur *also* grows without bound. We conjecture that this is impossible, and in particular that more careful bookkeeping in Lemma 6.15 would yield a bound on the maximum possible depth a trial can reach before it terminates normally—specifically, a single bound that applies to all trials.

If this conjecture holds, as D increases from trial to trial, eventually it would exceed the maximum depth for normal termination. From that point on all trials would be normally terminated, so the behavior of FRTDP+AMD would be identical to that of FRTDP-AMD and it would terminate by Theorem 6.16. □

6.6 Experimental Results

We evaluated the performance of the various search strategies as applied to benchmark problems from the MDP and POMDP literature. Both HSVI and FRTDP showed substantial performance improvements over prior state-of-the-art algorithms with respect to some of the benchmark problems.

6.6.1 MDP Results

Our MDP performance evaluation used problems drawn from the popular racetrack discrete MDP domain of Barto et al. (1995). Instances of racetrack are RTDP-solvable, so FRTDP should terminate, and HSVI may not.

States of racetrack are integer vectors in the form $(x, y, \dot{x}, \dot{y})$ that represent the discrete position and speed of the car in a 2D grid. The actions available to the car

are integer accelerations $(\ddot{x}, \ddot{y})$ where both $\ddot{x}$ and $\ddot{y}$ are drawn from $\{-1, 0, 1\}$. The car starts in one of a set of possible start states. The goal is to maneuver the car into one of a set of goal states. Some cells in the grid are marked as obstacles; if the car's path intersects one of these cells, it is reset back to one of the start states with zero velocity. Uncertainty in this problem comes from "skidding". Each time the agent takes an acceleration action, with probability p the car skids: the commanded action is replaced with $(\ddot{x}, \ddot{y}) = (0, 0)$.

Because HSVI and FRTDP focus on outcome selection, we also wanted to study increasing the amount of uncertainty in the problem. We did this by increasing the number of possible outcomes from an error. We call this the "wind" problem variant (marked by adding a suffix of `-w`). In the wind variant, with probability $p = 0.1$ an additional acceleration is added to the commanded acceleration. The additional acceleration is drawn from a uniform distribution over 8 possible values: $\{(-1,-1), (-1,0), (-1,1), (0,-1), (0,1), (1,-1), (1,0), (1,1)\}$. The idea is that instead of skidding, a "gust of wind" provides additional acceleration in a random direction.

We selected two racetrack problems whose maps have been published: `large-b` from Barto et al. (1995) and `large-ring` from Bonet and Geffner (2003b). With two versions (normal and "wind") for each problem, our results cover a total of four problems.

We selected three heuristic search algorithms to compare with HSVI and FRTDP: RTDP, LRTDP, and HDP. In addition, we implemented a modified version of HDP that maintains a lower bound and uses that as the basis for its output policy. We call this algorithm HDP+L.

Following Bonet and Geffner (2003b), all algorithms were provided with the same initial upper bound V_0^U, calculated by a domain-independent relaxation in which the best possible outcome of any action is assumed to always occur. Formally, the Bellman update is replaced by

$$V(s) \leftarrow \max_a \left[R(s,a) + \gamma \max_{s' \in \mathcal{N}(s,a)} V(s') \right] \tag{6.121}$$

The time required to calculate V_0^U is not included in the reported wallclock times, since it is the same for all search strategies. However, we do include the time required to expand the transition dynamics for each state the first time it was updated by the search strategy; this expansion included checking for collisions with the walls.

There is no trivial way to calculate a useful uniformly improvable lower bound for a racetrack problem. See McMahan et al. (2005) for discussion of of a general technique for efficiently calculating an informative lower bound. For convenience,

rather than implementing this general technique, we manually specified $V_0^L(s) = -1000$, which is a gross underestimate of the actual $V^*(s)$ values for the specific problem instances we used.

Our experiments were run on a 3.2 GHz Pentium-4 processor with 1 GB of RAM. We implemented all search strategies in C++. We applied each search strategy to each racetrack instance, running until one of two conditions occurred, either (1) approximately 100 seconds of wallclock time elapsed or (2) for strategies that have a way to monitor regret, the regret dropped below $\epsilon = 10^{-3}$.

Figures 6.5 and 6.6 report progress vs. wallclock time for each combination of problem and search strategy. Each column is devoted to one of the problems. The top frame of each column plots the progress of the upper bound value $V^U(s_0)$ and the middle frame plots the lower bound $V^L(s_0)$ for those search strategies that maintain a lower bound (HDP+L, HSVI, and FRTDP).

The bottom frame plots the progress of the mean reward $J\pi(s_0)$ received by executing the best policy so far. The policy used is PV^L for search strategies that keep two-sided bounds (HDP+L, HSVI, FRTDP), or PV^U for those that keep only the upper bound (RTDP, LRTDP, HDP). Each data point in the $J\pi(s_0)$ plots was generated by interrupting the algorithm and taking the mean total discounted reward of its current policy over 10^4 simulation trials. Each simulation trial was terminated after at most 251 steps if the goal had not been reached. The approximate size of the 95% confidence intervals for these estimates is provided in the title of the plot.

For all but one of the racetrack instances (the exception was `large-ring`), HSVI failed to terminate because one of its trials entered an apparently infinite loop. We handled this by plotting any data points gathered up to the point where HSVI entered its final trial. Plots for other algorithms generally ended before 100 seconds elapsed because they achieved the regret bound $\epsilon = 10^{-3}$. RTDP is the exception as it does not have a built-in way to monitor the regret bound (and it usually converged more slowly in any case).

These plots are large data sets that we would like to reduce to a few scores that we can use to rank performance. Optimal MDP planning is usually defined as optimizing the true policy quality $J\pi(s_0)$ as estimated in the bottom frame plots, so the first idea for scoring is to somehow summarize the bottom frame.

However, in a real application, we might require not only a good policy but a *guaranteed* good policy, in which case it may not be particularly helpful for the policy to achieve high $J\pi(s_0)$ before the quality guarantee "catches up"—the algorithm would need to continue until it had a strong guarantee in any case.

Furthermore, in ranking algorithms we prefer to report the relative amount of time required to achieve a fixed quality, which is easy to understand across multiple problems, rather than the relative quality achieved at a fixed time, which is more

problem-specific since the magnitude of difference in quality that constitutes an "important" change is subjective and varies according to the problem.

Calculating the relative amount of time required to achieve a fixed quality in effect means running a horizontal line across the graph and reading off the time value where the horizontal intersects the quality plot for each algorithm.

Unfortunately, this is problematic for the $J\pi(s_0)$ vs. time plots, which are in general not invertible functions. Quality of the current policy can increase and decrease erratically as the algorithm progresses, and is measured by a noisy sampling process. Depending on the method used to calculate the point of intersection, the reported time could be confounded by changes in both the precision of policy quality measurements and the frequency at which they are taken.

For these reasons, our reduced scores instead focus on the performance with respect to the regret bound $\mathrm{width}(\hat{V}(s_0))$, which is invertible since it decreases monotonically. In order to include algorithms which do not maintain a lower bound, we also calculate scores based on the monotonically decreasing upper bound $V^U(s_0)$. After discussing these reduced scores we will look at cases where their implied ordering of the algorithms differs from the qualitative ordering observed in the $J\pi(s_0)$ plot.

For each problem, let $V^U_{\min}$ be the minimum value such that every search strategy achieves a value of $V^U(s_0)$ at or below $V^U_{\min}$ by the end of its run. Similarly, let $\epsilon_{\min}$ be the minimum regret value such that every search strategy that keeps two-sided bounds achieves a value of $\mathrm{width}(\hat{V}(s_0))$ at or below $\epsilon_{\min}$. These values can be thought of as "least common denominators" that allow time comparisons to be made across all search strategies.

Table 6.3 reports the effort each search strategy required to reach the common denominator upper bound value $V^U_{\min}$.[11] The left side columns give the number of point-based updates performed, and the right side columns give the corresponding wallclock time. HSVI was not included in the analysis for problems that caused it to enter an infinite loop.

We see that FRTDP requires the fewest updates to reach $V^U_{\min}$ for all problems except `large-ring`, the one problem for which HSVI successfully terminated. In that case, FRTDP was edged out slightly by HSVI. Excluding HSVI, FRTDP's speedup relative to the best other search strategy, measured in number of updates, ranges from 1.6x to 3.1x.

The wallclock time measurements give a more mixed picture. Algorithms that keep two-sided bounds (HDP+L, HSVI, and FRTDP) incur extra overhead on each update for maintaining the lower bound. FRTDP also incurs overhead for maintaining priority values. Relative to LRTDP updates, FRTDP updates take about

[11] $V^U_{\min}$ values for the problems were: LB, -23.25; LBW, -24.44; LR, -16.17; LRW, -16.51.

twice as long; this is sufficient to put LRTDP ahead in terms of wallclock time for `large-b` and `large-b-w`, but not for `large-ring` and `large-ring-w`.

Table 6.4 reports the effort each search strategy required to reach the common denominator regret value $\epsilon_{\min}$.[12] Again, the left side columns give the number of point-based updates performed, and the right side columns give the corresponding wallclock time.

The ordering for effort to reach $\epsilon_{\min}$ is broadly similar to the previous table, with FRTDP requiring far fewer updates than HDP+L, but much less dominant in terms of wallclock time. Regrettably, our tests did not include a variant of LRTDP that keeps two-sided bounds ("LRTDP+L"); it might well have outperformed FRTDP in terms of regret wallclock time on the `large-b` variants.

Comparing the upper bound and regret scores to the $J\pi(s_0)$ plots, a few things stand out. First, we notice that (especially for the `large-b` variants) RTDP quickly reaches a high policy quality despite the fact that its upper bound convergence is the slowest among all the algorithms. As far as we can tell, this is mostly a "last mile" issue. RTDP quickly reaches a reasonable policy, but takes a long time to converge for the last few relevant states because of its stochastic outcome selection. Unfortunately, this also suggests that the relative timing results in the tables are fairly sensitive to the choice of the target value $V^U_{\min}$ or $\epsilon_{\min}$—if we had used looser values RTDP would have looked better.

Second, as judged by eyeballing the $J\pi(s_0)$ plots, the performance did not vary too much among all the algorithms (with the exception of HSVI's failure to terminate). In every case, the policy quality was within, say, $\epsilon = 1$ of the optimal policy relatively quickly. The relative wallclock times required by all the algorithms were within an order of magnitude of each other.

Third, on `large-ring` and especially `large-ring-w`, HSVI and FRTDP show $J\pi(s_0)$ behavior that is qualitatively different from the other search/problem combinations. Their policy quality increases smoothly rather than steeply, reaching reasonable values relatively early on. We attribute this to outcome selection differences.

RTDP selects outcomes stochastically, so over the course of a long trial, it is unlikely to stay on the "nominal" path that always selects the most likely outcome. LRTDP and HDP variants have bushier trials—in effect, they always select all outcomes, but as a result their early trials involve far more updates than the other search strategies.

In contrast, FRTDP and HSVI choose a single outcome in a systematic way. This means that the first trial they roll out is likely to stay on a path that is nominal in some sense. If the problem is structured so that the nominal path quickly reaches

[12] Due to the way the algorithms were terminated, the $\epsilon_{\min}$ value was 10^{-3} for all problems.

	Millions of updates				Wallclock (s)			
	LB	LBW	LR	LRW	LB	LBW	LR	LRW
RTDP	8.26	70.26	3.86	62.03	9.7	84.9	6.2	83.2
LRTDP	0.92	1.75	1.30	2.56	**3.3**	**13.4**	4.7	20.0
HDP	1.83	3.36	2.40	3.30	4.6	17.2	6.4	21.2
HDP+L	1.83	3.36	2.40	3.30	4.6	18.3	6.6	21.7
HSVI	-	-	**0.42**	-	-	-	3.0	-
FRTDP	**0.57**	**0.85**	0.44	**0.82**	4.0	14.3	**2.9**	**13.9**

Table 6.3: Effort required to achieve the upper bound value $V^U_{\min}$ for MDPs.

	Millions of updates				Wallclock (s)			
	LB	LBW	LR	LRW	LB	LBW	LR	LRW
HDP+L	1.83	3.36	2.40	3.43	4.6	18.3	6.6	22.3
HSVI	-	-	**0.42**	-	-	-	3.0	-
FRTDP	**0.58**	**0.96**	0.44	**0.99**	**4.1**	**14.7**	**2 .9**	**14.8**

Table 6.4: Effort required to achieve the regret value $\epsilon_{\min}$ for MDPs.

a goal, this property can lead to quick discovery of a good core policy, with further updates tending to improve the policy by making it more robust to falling off the nominal path.

6.6.2 POMDP Results

We also compared the performance of the search strategies applied to the same large sparse POMDP benchmark problems used in Chapter 4. The experimental procedure for POMDP testing was in most respects the same as was used for MDPs; we mention only the differences here.

In order to apply the search strategies to POMDPs, we needed to choose an appropriate representation for the POMDP value function bounds. We selected the `mask/prune` lower bound representation and the `mask` upper bound representation based on the performance evaluation in Chapter 4. We also extended the maximum wallclock time allotted for each run from 100 to 1000 seconds, since the POMDP problems are far more difficult.

Table 6.5 reports the effort required for each search strategy to achieve the upper bound value $V^U_{\min}$.[13] The left side of the table gives the number of updates required. Several algorithms have nearly the same performance, with HSVI

[13] $V^U_{\min}$ values for the problems were: Tag, -2.22; RS57, 26.29; LS1, 106.84.

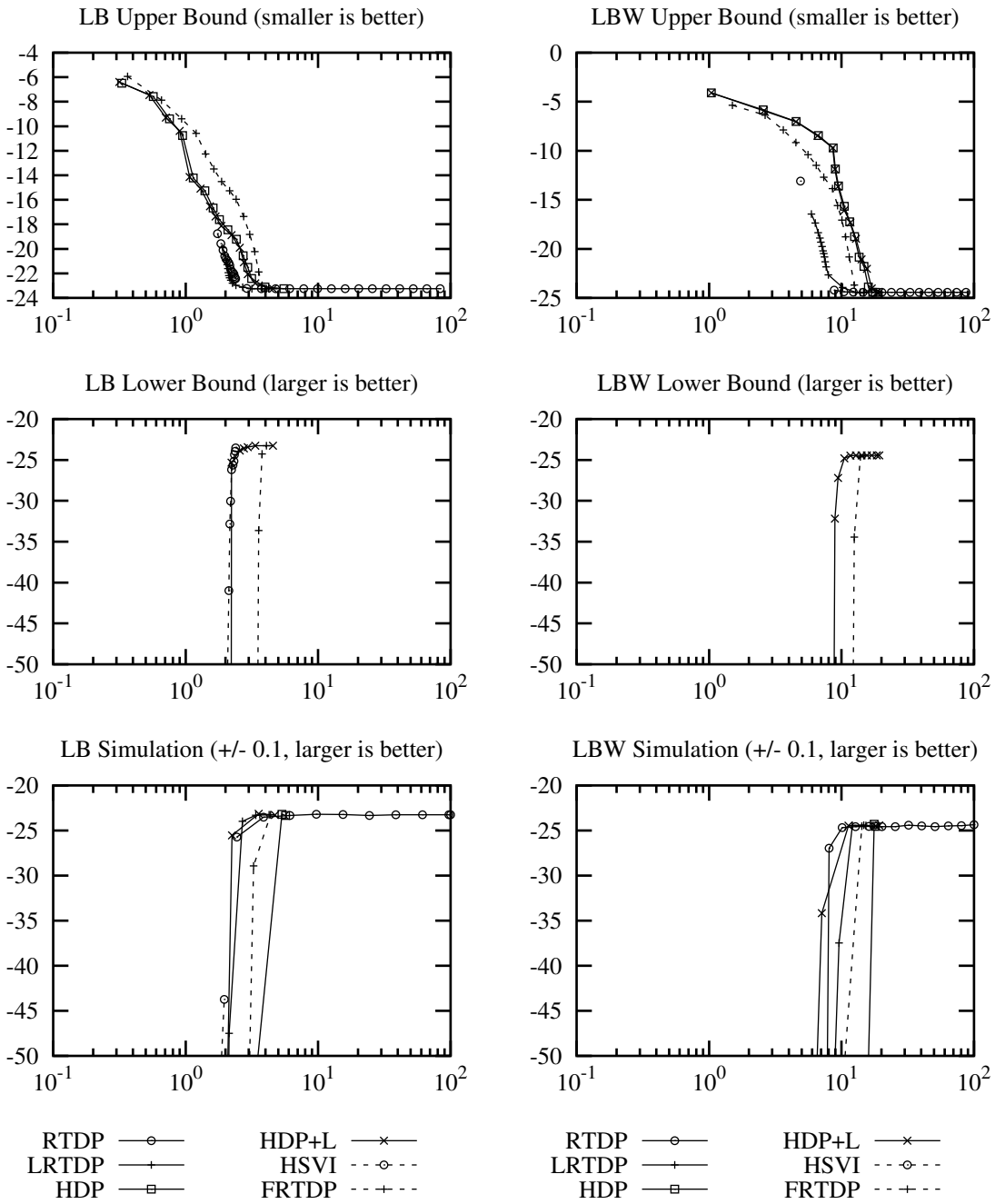


Figure 6.5: Progress vs. wallclock (s) for the `large-b` (“LB”) and `large-b-w` (“LBW”) MDPs.

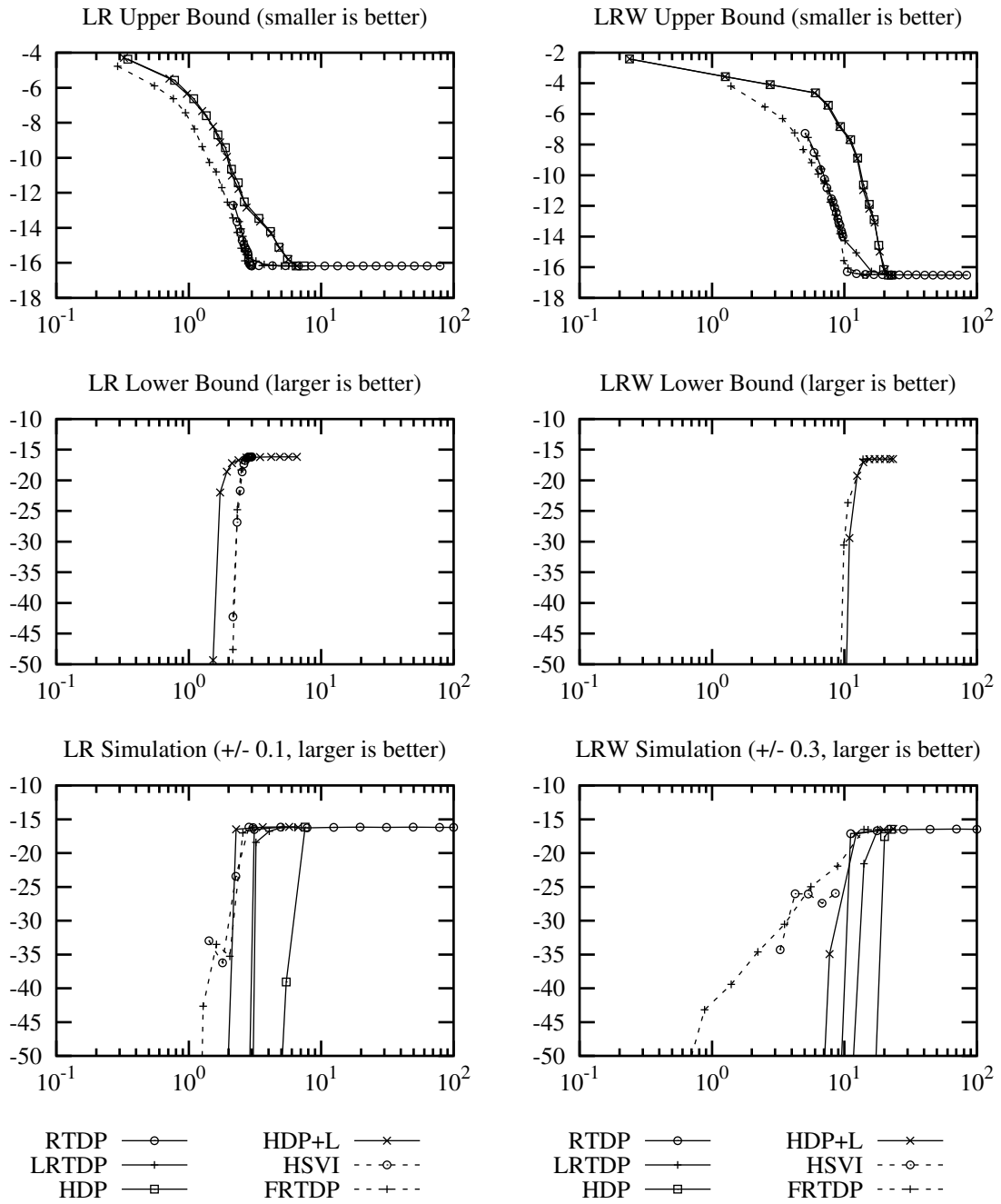


Figure 6.6: Progress vs. wallclock (s) for the `large-ring` ("LR") and `large-ring-w` ("LRW") MDPs.

performing best by a narrow margin on *Tag* and *RockSample[5,7]*, and RTDP on *LifeSurvey1*.

The right side of the table gives the corresponding wallclock time required. We find that while RTDP performs best on *LifeSurvey1*, HDP is the surprise winner on both the *Tag* and *RockSample[5,7]* problems, outperforming HSVI by a wide margin. This indicates that individual HDP updates were much faster than HSVI updates, which can be explained by two main factors.

First, since HDP did not maintain a lower bound, it did not incur extra overhead from lower bound updates. When we examined discrete MDPs with tabular bounds representations, lower bound overhead had relatively little impact on overall update time. In contrast, for POMDPs, using more complex updatable representations, lower bound updates turned out to be much slower than upper bound updates, more than doubling the update time. (However, the extra overhead FRTDP incurs to calculate a priority value turns out to be insignificant compared to value function updates.)

Second, for these problems HDP updated each belief it expanded more often. A useful statistic to examine is the *update ratio*, or mean number of belief updates per distinct belief expanded. For *Tag*, HDP's update ratio was 10.6 vs. 3.2 for HSVI. For *RockSample[5,7]*, HDP's update ratio was 12.8 vs. 3.2 for HSVI. Because HDP spent more time updating beliefs it had already expanded rather than looking at new beliefs, its bounds representations tended to contain fewer entries and thus could be updated faster.

We are unsure why HDP's update ratio was higher, but it may be explained by the fact that each bushy HDP trial included more updates than a single-path HSVI trial. Actions selected early in a trial in effect commit an algorithm to exploring a particular part of the search graph for the rest of the trial. Each such commitment lasted longer with HDP's longer trials, which could have led HDP to visit fewer distinct beliefs.

Table 6.6 reports the effort required for each search strategy to achieve the regret value $\epsilon_{\min}$.[14] The left side of the table gives the number of updates required. HSVI required the fewest updates on *Tag*, narrowly edging out HDP+L with FRTDP well behind, and FRTDP performed best by a wide margin on the other problems.

We are not sure why different algorithms fare better from problem to problem. We can say that the *Tag* is qualitatively different from the other two problems in that it (1) has far fewer states, and (2) includes motion errors that continually inject fresh uncertainty into the problem. In contrast, for both *RockSample[5,7]* and *LifeSurvey1*, uncertainty starts high and decreases throughout execution as obser-

[14] $\epsilon_{\min}$ values for the problems were: Tag, 3.87; RS57, 4.26; LS1, 15.21.

vations are collected.

The right side of the table gives the corresponding amount of wallclock time required. FRTDP is still best on *RockSample[5,7]* and *LifeSurvey1*, but HDP+L manages to beat HSVI on *Tag* due to its faster individual updates. Both HSVI and HDP+L incur lower bound overhead, but HDP+L gains an advantage from its higher update ratio (the same update ratio as HDP, since HDP and HDP+L choose the same beliefs to update).

Note an apparent anomaly—according to the table, FRTDP updates appear to take less time on average than HDP+L updates on the *RockSample[5,7]* problem. This is because for any algorithm the time spent per update tends to grow as the algorithm progresses due to the increasing size of the bounds representations. Since FRTDP reached the desired regret bound after fewer updates, its reported number of updates and wallclock time represent the average time per update for only the early part of the update sequence.

Figures 6.7 and 6.8 report progress vs. wallclock time for each combination of problem and search strategy. As with the MDP performance plots, each column is devoted to one of the problems, and from top to bottom the frames plot $V^U(b_0)$, $V^L(b_0)$, and the estimated value of $J\pi(b_0)$ from simulation trials.

The plots of upper bound and lower bound progress do not contain any particular surprises given our earlier discussion of relative effort for the different search strategies.

However, the plots of policy quality $J\pi(b_0)$ are qualitatively different from the corresponding plots for MDPs. Search strategies that maintained two-sided bounds and used PV^L as their policy (HDP+L, HSVI, and FRTDP) had quality increasing smoothly throughout the run. On the other hand, search strategies that used PV^U (RTDP, LRTDP, and HDP) had policy quality that varied erratically up and down without any obvious increase over the run. For POMDPs, the regret guarantee implied by the uniformly improvable lower bound is evidently a more significant issue.

6.7 Conclusions

We presented two novel search strategies, HSVI and FRTDP, that are designed to be used within the framework of focused value iteration. Both search strategies use two-sided uniformly improvable bounds to both guarantee the quality of the output policy and guide outcome selection. FRTDP goes beyond HSVI in that its outcome selection is less myopic; however, to achieve this it sacrifices some simplicity.

FRTDP converges over the class of RTDP-solvable MDPs, which most naturally represent problems with a finite state space where the task is to reach a goal

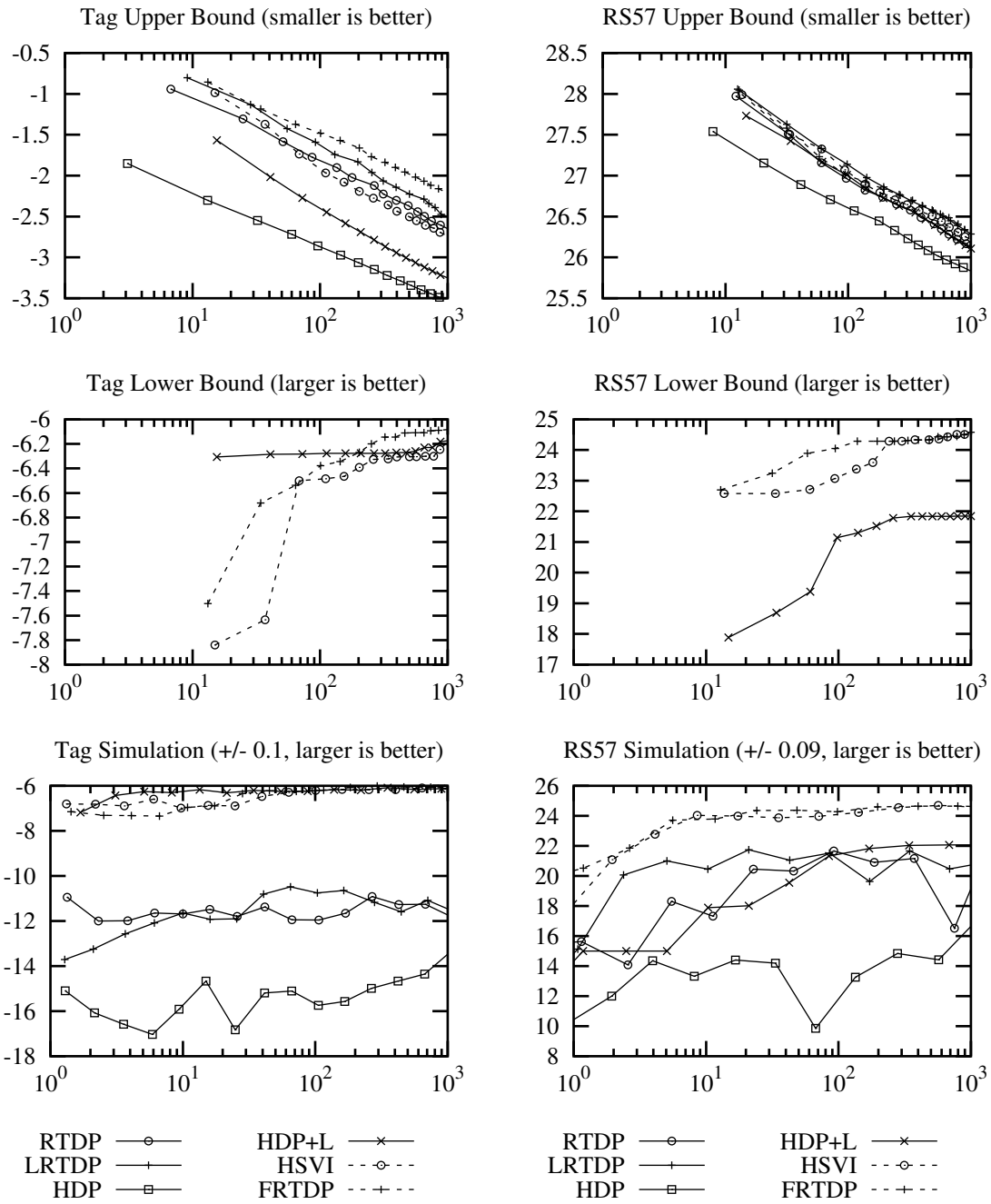


Figure 6.7: Progress vs. wallclock (s) for the *Tag* and *RockSample[5,7]* ("RS57") POMDPs.

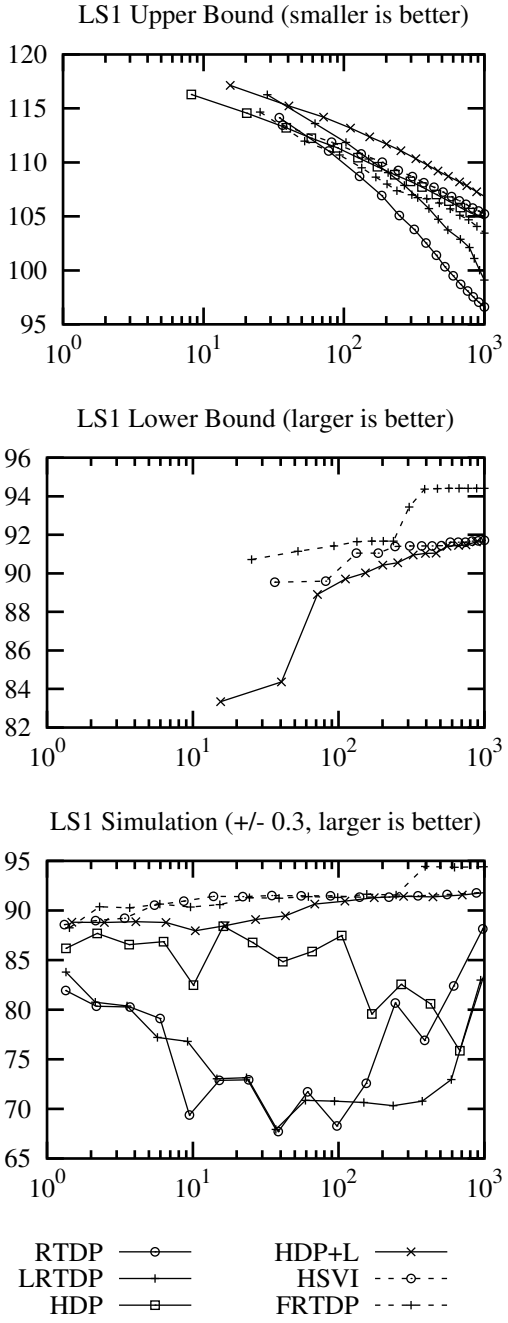


Figure 6.8: Progress vs. wallclock (s) for the *LifeSurvey1* ("LS1") POMDP.

	Thousands of updates			Wallclock (s)		
	Tag	RS57	LS1	Tag	RS57	LS1
RTDP	39.9	49.2	**65.0**	306.1	653.5	**188.8**
LRTDP	35.3	42.4	67.7	465.0	1000.7	327.2
HDP	18.8	34.1	175.8	**10.0**	**261.8**	483.1
HDP+L	18.8	34.1	175.8	65.1	644.4	1002.6
HSVI	**14.1**	**27.2**	132.9	218.5	827.3	586.1
FRTDP	43.0	31.8	65.9	974.8	1000.0	346.0

Table 6.5: Effort required to achieve the upper bound value $V^U_{\min}$ for POMDPs.

	Thousands of updates			Wallclock (s)		
	Tag	RS57	LS1	Tag	RS57	LS1
HDP+L	27.6	47.0	175.8	**101.9**	1000.0	1002.6
HSVI	**21.9**	5.6	133.5	403.9	79.6	588.8
FRTDP	43.0	**3.6**	**51.8**	974.8	**34.8**	**252.4**

Table 6.6: Effort required to achieve the regret value $\epsilon_{\min}$ for POMDPs.

state with minimum effort. HSVI, on the other hand, converges over the class of discounted finite-branching MDPs, which can naturally express tasks with multiple competing goals and costs; this class also includes the belief-MDP representations of discounted POMDPs that have a finite state space. Unfortunately, neither algorithm has guaranteed convergence over both classes of MDPs.

Our experiments with RTDP-solvable MDPs drawn from the racetrack domain show that HSVI often fails to converge. As for FRTDP, it always converged, usually with fewer updates than other competing search strategies, but its overall time performance was slower in some cases (by the $V^U_{\min}$ measure) because its individual updates were more expensive. Our advice for practitioners:

1. HSVI is clearly not suitable for this type of RTDP-solvable MDP due to its frequent failure to converge.

2. Although RTDP was competitive with the other algorithms in terms of quickly producing an acceptable policy, it does not appear to have any compelling advantage, and should probably be avoided due to its lack of a built-in way to monitor convergence.

3. Any of the other algorithms we studied will probably provide acceptable performance on small problems if time constraints are not too tight and extremely precise convergence is not necessary. With more severe require-

ments, it may be worthwhile to test a few of the algorithms and compare performance. FRTDP is a reasonable first choice. We should also note that some potentially strong algorithms were not included in our experimental comparison (see §2.9.5).

4. One should keep two-sided bounds if it is important for the output policy to have strong quality guarantees.

We also compared search strategy performance on large sparse POMDPs. Although we cannot guarantee that FRTDP will converge for these problems, it did not have any difficulties in practice. No single algorithm dominated over all the problems. By the $V^U_{\min}$ measure, many search strategies required nearly the same number of updates, and those that did not incur extra overhead from keeping a lower bound had much better time performance. By the $\epsilon_{\min}$ measure, which seems to be better connected to policy quality in simulation, HSVI and FRTDP required fewer updates, but the superior time performance of HDP+L on the *Tag* problem showed that for POMDP search it is also important to take into account the update ratio. Our advice for practitioners:

1. For these large POMDPs, keeping a lower bound appears to be critical. Search strategies that relied on PV^U had policy quality that was highly erratic, without any obvious increase over time.

2. FRTDP performed well for these POMDPs, but it may not converge in all cases, particularly because the POMDP belief space is not finite.[15] For some uses this reliability concern may outweigh its evident performance benefits.

3. Among algorithms that keep a lower bound, there is wide variation in convergence time with no clear winner across all problems. This suggests trying multiple algorithms to see which is best suited to your type of problem. HSVI is a reasonable first choice.

[15] It is possible that the generalization provided by some POMDP value function representations ensures FRTDP termination even though the belief simplex is infinite. Unfortunately, this kind of question is not easy to resolve using the theoretical tools we have developed so far.

Chapter 7

POMDP State Abstraction

The state of a POMDP can often be factored into a tuple of n state variables. The corresponding unfactored or "flat" model, with size exponential in n, may be intractably large. This issue is important because most existing POMDP solvers operate on a flat model representation, with only a few exceptions (Hansen and Feng, 2000; Poupart and Boutilier, 2004).

However, in some problems there are efficient ways to identify irrelevant variables that cannot affect the solution. In that case the irrelevant variables can be abstracted away, exponentially shrinking the state space in the flat model (Boutilier and Dearden, 1994). If the overall task can be hierarchically decomposed into subtasks, one can take a finer-grained approach and temporarily abstract away variables that are relevant overall but irrelevant within a particular subtask (Pineau et al., 2003c). When interleaving planning and execution, the amount of abstraction may also vary at different planning horizons (Baum and Nicholson, 1998).

We present an alternate method called conditionally irrelevant variable abstraction (CIVA) for losslessly reducing the size of the factored model. A state variable is said to be *conditionally irrelevant* for a given partial assignment to other state variables if certain conditions are satisfied that guarantee it can be temporarily abstracted away without affecting policy optimality. Figure 7.1 shows how CIVA fits into the overall planning process. Our method considers only factored state, although factored actions and observations can also be useful (Guestrin et al., 2001; Feng and Hansen, 2001).

We applied CIVA to previously intractable POMDPs from a robotic exploration domain (motivated in §8.4). We were able to abstract, expand, and approximately solve POMDPs that had up to 10^{24} states in the uncompressed flat representation. The resulting policies outperformed hand-tuned heuristic policies both in simulation and in testing onboard a robot in a controlled outdoor environment.

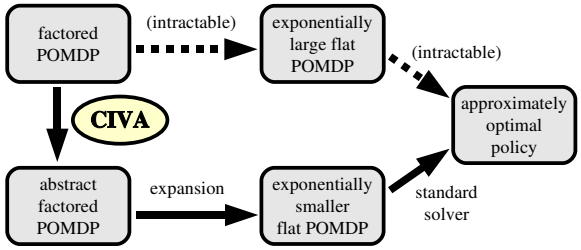


Figure 7.1: CIVA Process Diagram.

7.1 Example Problem

Our primary testing domain for CIVA was the *LifeSurvey* robotic exploration problem. We will use *MiniLifeSurvey*, a simplified version of *LifeSurvey*, to provide intuition about conditional irrelevance. In *MiniLifeSurvey*, a robot is moving through a one-dimensional map from west to east. The robot has sensors for detecting life en route, but it must balance the cost of using these sensors against the expected value of the resulting data. The robot has three available actions:

1. `move`: Moves the robot one cell to the east, with a cost of -1. Always returns a `null` observation.

2. `scan`: Applies the robot's long-range sensor, providing noisy information as to whether life is present in the cell just ahead of the robot, with a cost of -2. Returns either a `positive` or `negative` observation.

3. `sample`: Applies the robot's short-range sensor to the current cell, with a cost of -10. Always returns a `null` observation (the sensor data is returned to scientists, but not analyzed onboard the robot). If the cell contains detectable life, the robot receives a reward of +20.

The variables in *MiniLifeSurvey* are:

1. X_1: The position of the robot, ranging from 1 to k. The position is always known and advances deterministically when the `move` action is applied.

2. $Y_1, \ldots, Y_k$: Each Y_i has the value `L` or `N` ("life" or "no life") depending on whether cell i of the map contains detectable life or not.

The robot starts in cell 1 and has remote sensing data that provides independent prior probabilities for the Y_i variables. The problem ends when the robot applies the `move` action in the rightmost cell.

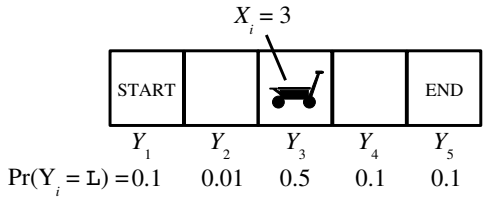


Figure 7.2: The *MiniLifeSurvey* problem.

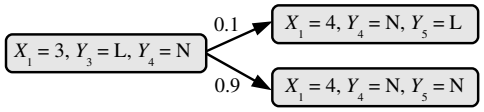


Figure 7.3: A state transition in the abstract model.

Figure 7.2 shows an instance of *MiniLifeSurvey* with $k = 5$. The priors $\Pr(Y_i = \texttt{L})$ for each Y_i are shown below the map. The robot is shown at position $X_1 = 3$. From this position, the observation returned by the `scan` action depends on Y_4, and the reward from the `sample` action depends on Y_3.

The key insight underlying CIVA is that in a structured problem like *MiniLifeSurvey* the robot needs to consider joint assignments to only a few of the state variables at any one time. In position $X_1 = 3$, only the variables Y_3 and Y_4 are immediately relevant in the sense that they can affect the rewards or observations in the next time step. Because the robot moves only forward, variables Y_1 and Y_2 can have no further effect on the system. Variable Y_5 will be important later, but nothing can be learned about its value through any of the other Y_i variables or the robot's next action, so in considering the next action to take the robot can temporarily disregard Y_5 and reconstruct its probability distribution later as needed. (These concepts will be formalized later.)

Figure 7.3 shows one example state transition for the `move` action in the abstract model produced by CIVA. Each abstract state in the reduced model corresponds to an equivalence class of states in the original model. For example, the abstract state on the left corresponds to the set of all states with $X_1 = 3$, $Y_3 = \texttt{L}$, $Y_4 = \texttt{N}$, and any value for the other Y_i variables. The arrows in the diagram are labeled with transition probabilities.

As we will explain later, the abstract states in the abstract model specify values only for the Y_i variables that are conditionally relevant given the value of X_1. With $X_1 = 3$, Y_3 and Y_4 are conditionally relevant; with $X_1 = 4$, Y_4 and Y_5 are conditionally relevant.

Abstracting away variables in the factored model results in an exponentially smaller flat model. In the uncompressed *MiniLifeSurvey* model with map length k, there are k possible values for X_1 and k binary-valued variables Y_i, so there are $k \times 2^k$ states. In the CIVA-compressed model, only the position and two of the Y_i variables need to be tracked at a time, so there are just $4k$ abstract states.

7.2 POMDP Review

Recall that a POMDP policy is a mapping from histories to actions in the form

$$a_t = \pi(a_0, o_0, a_1, o_1, \ldots, a_{t-1}, o_{t-1}). \tag{7.1}$$

Given a system model and the initial belief b_0, the agent can use Bayesian updates to calculate the posterior belief b_t corresponding to any history and rewrite the policy in the form $a_t = \pi(b_t)$. It is a theorem that every POMDP P has an optimal policy π^*, which among all policies π maximizes the long-term expected reward

$$J^P\pi(b) = E_{\pi, b_0=b}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right] \tag{7.2}$$

for all beliefs b. (We use the superscript P in J^P to denote the value of the policy with respect to the specific POMDP model P.) Solving the POMDP exactly means finding such an optimal policy, but most practical algorithms find a policy that is only guaranteed to be near-optimal for a given starting belief b_0.

We say that two models $P = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma, \mathcal{O}, O, b_0 \rangle$ and $P' = \langle \mathcal{S}', \mathcal{A}', T', R', \gamma', \mathcal{O}', O', b_0' \rangle$ are *policy-compatible* if $\mathcal{A} = \mathcal{A}'$ and $\mathcal{O} = \mathcal{O}'$. Policy compatibility ensures that any policy for P' can be applied to P, since policies for the two models have the same functional form according to equation (7.1), although they may represent system state and beliefs differently.

We say that P and P' are *policy-equivalent* if for every policy π the long-term expected reward of the initial belief is the same for the two models

$$J^P\pi(b_0) = J^{P'}\pi(b_0'). \tag{7.3}$$

Policy equivalence ensures that the two models have the same optimal policies.[1] Applying CIVA to a model P produces a policy-equivalent model P'.

[1] These definitions of policy compatibility and equivalence are novel as far as we know.

7.3 Conditional Relevance

In this section we formally define conditional relevance and present an approach for identifying conditionally irrelevant variables so that CIVA can abstract them away. CIVA is based on the assumption that the state can be factored into a tuple of discrete variables $\langle X_1, \ldots, X_j, Y_1, \ldots, Y_k \rangle$, where $X_1, \ldots, X_j$ are *upstream variables* and $Y_1, \ldots, Y_k$ are *downstream variables*. We denote the tuple of upstream values $X = \langle X_1, \ldots, X_j \rangle$, and similarly $Y = \langle Y_1, \ldots, Y_k \rangle$. We use ϕ to denote the value of particular variables; for example, $\phi_{X_i}(s)$ is the value of variable X_i in state s, and $\phi_X(s)$ is the joint value of all upstream variables in state s.

Upstream variable values are always known to the agent, transition deterministically, and are independent of the downstream variables (though they may depend on other upstream variables). In *MiniLifeSurvey*, the known and deterministically transitioning position X_1 is an upstream variable, but the uncertain Y_i variables representing the presence of life are downstream variables.

Upstream variable dynamics are already specified along with the other variables as part of the transition function T, but it is also convenient to define special notation that reflects the additional structure. We define the upstream transition function U such that $x' = U(x, a)$, where x is the value of the upstream variables at one time step, and x' is the upstream value at the next time step after taking action a.

An upstream value x is *reachable* if, starting from the known initial upstream value x_0, there is a sequence of upstream transitions that reaches x. The set of all reachable upstream values can easily be generated by forward recursion from x_0.

We will build up the definition of conditional relevance in several steps. A downstream variable Y_i is *immediately relevant* at x, written $Y_i \in f^{\text{IR}}(x)$, if it is possible for Y_i to have an "immediate effect" on the problem. That is, $Y_i \in f^{\text{IR}}(x)$ unless all of the following constraints hold:

I1. Y_i has no immediate effect on reward. For any state s, let s/E denote the set of states that agree with s over all variables other than Y_i. Let a be an action, let s be a state with $\phi_X(s) = x$, and let $s' \in s/E$. Then we must have $R(s, a) = R(s', a)$.

I2. Y_i has no immediate effect on observations. Let a be an action, o be an observation, let s be a state with $\phi_X(s) = x$, and let $s' \in s/E$. Then we must have $O(a, s, o) = O(a, s', o)$.

I3. Y_i has no immediate effect on the transitioning of other variables. Let a be an action, let s be a state with $\phi_X(s) = x$, let $s' \in s/E$, and let s'' be an

arbitrary state. Then we must have

$$\sum_{\sigma \in s''/E} T(s, a, \sigma) = \sum_{\sigma \in s''/E} T(s', a, \sigma). \tag{7.4}$$

The idea is that immediately relevant variables tend to become "entangled" with the rest of the system, so they typically cannot be abstracted away without losing policy-equivalence. In *MiniLifeSurvey*, when $X_1 = i$, Y_i is immediately relevant because it influences reward under the `sample` action (condition I1), and Y_{i+1} is immediately relevant because it influences the observation under the `scan` action (condition I2).

A downstream variable Y_i is *a-predictable* at x, written $Y_i \in f^{\mathrm{P}}(x, a)$, if it is possible to reconstruct the distribution over possible values of Y_i after applying action a in a state with $\phi_X(s) = x$, given only knowledge of x, a, and b_0. In other words, given x, a, and b_0, the distribution over possible values for Y_i after the transition must be conditionally independent of all other state information, including the preceding value of Y_i and other downstream variables.

The idea is that we can temporarily "forget" probabilistic information about the value of a variable, even one that is going to be relevant later, if at that later point the information can be reconstructed. In *MiniLifeSurvey*, when $X_1 = i$, the variable Y_j is a-predictable for all actions if $j \geq i + 2$. This is because the robot has not yet had an opportunity to learn anything about variables beyond its sensor range—all available information can be reconstructed from the initial belief.

A downstream variable Y_i is *conditionally relevant* at x, written $Y_i \in f^{\mathrm{R}}(x)$, if either:

C1. The variable Y_i is immediately relevant at x, or

C2. For some action a, (i) Y_i is conditionally relevant at $x' = U(x, a)$, and (ii) Y_i is not a-predictable at x.

The idea is that the agent needs to keep track of probabilistic information about a variable if it is either immediately relevant or there is a future point where it is both relevant and we have no way to reconstruct the information. In *MiniLife-Survey*, when $X_1 = i$, only the immediately relevant variables Y_i and Y_{i+1} are conditionally relevant. The relevance determination algorithms discussed below help develop intuition as to why this is the case.

7.4 Relevance Determination

We assume that the factored model provided to CIVA specifies which variables are upstream versus downstream, as this is easy to determine manually. However, we still need a way to determine the upstream values where downstream variables are conditionally irrelevant so we can abstract them away. We call this problem *relevance determination*.

A relevance determination algorithm is *exact* if for any upstream value x it calculates the exact set $f^{\mathrm{R}}(x)$ as defined above. In contrast, it is *conservative* if for any x it calculates a superset of $f^{\mathrm{R}}(x)$. In other words, a conservative algorithm errs only on the side of tagging variables relevant when they are not. Conservative relevance determination may result in a compressed model that is larger than necessary, but it retains the key property that the original model and compressed model are policy-equivalent.

Our approach to relevance determination is conservative. It is a three-step process. We (1) find immediately relevant variables, (2) find predictable variables, and (3) find conditionally relevant variables.

7.4.1 Finding Immediately Relevant Variables

The first step in relevance determination is to calculate the immediately relevant variables $f^{\mathrm{IR}}(x)$ for every reachable upstream value x. To ensure that the overall conditional relevance determination is conservative, the immediate relevance determination also needs to be conservative. That is, when in doubt it must tag a variable as immediately relevant.

Checking the immediate relevance constraints I1-I3 for $Y_i \in f^{\mathrm{IR}}(x)$ by brute force is often intractable, since each constraint involves enumeration over the set of all states s with $\phi_X(s) = x$; this set has size exponential in the number of downstream variables. Thus tractable immediate relevance determination depends on leveraging the structure of the factored model.

CIVA is a general approach that is not tied to any particular factored representation. Possible representations for the R, O, and T functions include decision trees (Boutilier et al., 2000) and algebraic decision diagrams (ADDs) (St. Aubin et al., 2000), among others. Immediate relevance determination can be performed over any representation, but the choice of representation affects its computational complexity.

For concreteness, we describe an efficient exact immediate relevance determination algorithm for a particular decision tree representation. Let the functions R, O, and T be represented as decision trees with the following variable ordering: (1) first branch on the action, (2) then on the observation (for O only), (3) then on

upstream state variables in an arbitrary fixed order, (4) then on downstream state variables in an arbitrary fixed order. The T function takes two state arguments s and s'; all state variables of s are ordered before all state variables of s' in the decision tree. Let n_R, n_O, and n_T be the number of nodes in the decision tree representations of R, O, and T respectively.

One can determine if $Y_i \in f^{\text{IR}}(x)$ using the following procedure:

1. Check I1. Restricting the function R to a particular action a and upstream value x corresponds to selecting a particular subtree of the decision tree. If for every action a the corresponding subtree does not contain a node branching on Y_i, then I1 is satisfied. Overall, this check can be performed in order n_R time (and usually much faster, since portions of the tree relating to upstream values other than x can be ignored).

2. Check I2. Similar to the check of I1, this time iterating over all combinations of actions and observations in O. This check can be performed in order n_O time.

3. Check I3. Restricting T to a particular action a and upstream value $\phi_X(s) = x$ corresponds to selecting a particular subtree. Then in order to check I3 we must (1) sum out the Y_i variable of s' within the subtree, (2) recursively canonicalize the subtree by eliminating branch nodes that have identical subtrees on either branch, and (3) check if the subtree now contains a node branching on the Y_i variable of s. This procedure can be performed for all actions in order n_T time.

Thus, for each upstream value x and variable Y_i, we can check if $Y_i \in f^{\text{IR}}(x)$ in order $n_R + n_O + n_T$ time, which is relatively efficient if the model is compact. The procedure is exact under the condition that R, O, and T decision trees provided to it are canonical, meaning they contain no unnecessary branch nodes.

Note that not all problems with factored structure can be compactly represented with this type of decision tree. We expect that efficient conservative immediate relevance determination algorithms exist for ADDs and under certain relaxations of the variable ordering constraints, but this is the only case we have worked out in detail.

7.4.2 Finding Predictable Variables

The second step in relevance determination is to calculate the a-predictable variables $f^{\text{P}}(x, a)$ for every reachable upstream value x and action a. Recall that $Y_i \in f^{\text{P}}(x, a)$ if, after applying action a in a state with $X = x$, it is possible

to reconstruct the probability distribution of Y_i given only knowledge x, a, and b_0. Because of the way predictability relates to conditional relevance, we say that a predictability determination algorithm is conservative if it errs only on the side of tagging variables *not* predictable.

We do not know of any tractable algorithm for exact predictability determination in the general case. Predictability of Y_i at x depends on whether the agent is able to gain information about Y_i on the way from the initial state to a state with $X = x$, and whether that information is still pertinent when it arrives. Since these considerations can in general depend on the path that the agent takes through the state space, it might be very difficult to check that a variable is predictable over all paths.

However, if the goal is *conservative* predictability determination, there are several types of structure that make it easy to prove that a variable is predictable. For example:

- If applying action a in a state with $X = x$ overwrites all previous information about Y_i, then $Y_i \in f^{\mathrm{P}}(x, a)$. For example, if action a flips a coin, the agent knows that there is a 50% chance the coin shows heads after the state transition, regardless of what information it might have had before.

- If one can show that any path leading to a state with $X = x$ must pass through a state with $X = x'$, and entering a state with $X = x'$ fixes a known and permanent value for Y_i, then $Y_i \in f^{\mathrm{P}}(x, a)$. For example, suppose the only way to get hold of a fire extinguisher is to break its glass case. Then if the agent has the fire extinguisher, it can reconstruct the fact that the glass is broken.

In our robotic exploration domain, we rely on yet another type of special structure that implies predictability. A variable Y_i is *untouched* at x, written $Y_i \in f^{\mathrm{U}}(x)$ if it satisfies:

U1. Y_i is independent of other variables in the initial belief,

U2. The value of Y_i does not change over time,

U3. Y_i is not immediately relevant at x, and

U4. For every predecessor upstream value x' such that $x = U(x', a)$, Y_i is untouched at x'.

If a variable is untouched at x, we can be sure that its probability distribution is unchanged from what it was in b_0. This makes it a-predictable at x for every action a. For example, if the agent starts out believing there is a utility closet upstairs

with 50% probability, and the agent has not yet had enough time to go upstairs and check, its current belief about the utility closet is just its initial belief.

We can identify untouched variables using forward recursion. First we mark as touched every variable that violates U1-U3. Then we perform local updates to enforce the consistency of U4; if Y_i is touched at x, then Y_i is marked as touched at all successors $x' = U(x, a)$ of x. Local updates are propagated forward until U4 is globally satisfied.

7.4.3 Finding Conditionally Relevant Variables

The final step in relevance determination is to calculate the conditionally relevant variables $f^{\mathrm{R}}(x)$ for every reachable upstream value x. The reader may wish to review the definition of conditional relevance, conditions C1 and C2 above.

With $f^{\mathrm{IR}}(x)$ and $f^{\mathrm{P}}(x)$ in hand, it is straightforward to calculate $f^{\mathrm{R}}(x)$ by backward recursion. First we mark every immediately relevant variable as conditionally relevant to satisfy C1. Then we perform local updates to enforce the consistency of C2. If Y_i is conditionally relevant at x, then it is marked as conditionally relevant for all predecessors x' such that $x = U(x', a)$ and $Y_i \notin f^{\mathrm{P}}(x', a)$. Local updates are propagated backwards until C2 is globally satisfied.

7.5 Model Abstraction

This section defines the form of the abstract model produced by CIVA. First we define the *predictability transformed* version of the transition function T. Let x be an upstream value and a be an action such that $Y_i \in f^{\mathrm{P}}(x, a)$, and let s be a state consistent with x. The predictability of Y_i means that its probability distribution after the state transition can be calculated given only knowledge of x, a, and b_0. There are two ways this can happen:

1. For prior states with $X = x$, the value of Y_i after applying a depends only on a. In this case no change needs to be made.

2. The value of Y_i after applying a formally depends on some downstream variable Y_m, but in fact all reachable beliefs with $X = x$ have probability distributions for Y_m that lead to the same prediction of Y_i.[2] In this case, we can rewrite the transition function so that, independent of the value of Y_m, the posterior probability distribution of Y_i is its reconstructed value as an a-predictable variable.

[2]When we say a belief with $X = x$ we mean a belief in which only states with $X = x$ have non-zero probabilities.

The result of performing this rewrite wherever possible is denoted $\tilde{T}$.

Conditional relevance defines an equivalence relation E on states, such that for two states s, s', we have $E(s, s')$ if s and s' both (i) share the same upstream value x and (ii) agree on the values of the conditionally relevant downstream variables $f^{\mathrm{R}}(x)$. E induces a partition of $\mathcal{S}$ into equivalence classes of similar states. Let $s/E = \{s' \mid E(s, s')\}$ denote the class containing state s.

We will abuse notation by writing versions of $\tilde{T}$, O, and R that take equivalence classes as arguments. We define

$$\tilde{T}(s, a, s'/E) = \sum_{\sigma \in s'/E} T(s, a, \sigma), \tag{7.5}$$

and we define $\tilde{T}(s/E, a, s'/E) = q$ if for all $\sigma \in s/E$, we have $\tilde{T}(\sigma, a, s'/E) = q$. Otherwise $T(s/E, a, s'/E)$ is not well defined. $R(s/E, a)$ and $O(a, s'/E, o)$ can be well-defined or not in a similar way.

It turns out that with conditional relevance defined as presented earlier, for all s, s', a, o, we have that $\tilde{T}(s/E, a, s'/E)$, $R(s/E, a)$ and $O(a, s'/E, o)$ are well-defined. Thus equivalence classes in the original model can be used as states in the reduced model, and the equivalence-class versions of $\tilde{T}$, R, and O define the reduced system dynamics. The fact that the reduced system dynamics are well-defined implies that the abstract model is policy-equivalent to the original. We include only the equivalence classes corresponding to reachable upstream values in the reduced model.

7.6 Application to *MiniLifeSurvey*

Now we tie some of the formal concepts back to the *MiniLifeSurvey* domain introduced earlier. When the robot is at position $X_1 = 3$, the immediately relevant variables are $f^{\mathrm{IR}}(x) = \{Y_3, Y_4\}$. Cell 3 is the current cell, so Y_3 affects the reward when applying the `sample` action. Cell 4 is the cell just ahead of the robot, so Y_4 affects the observation when applying the `scan` action.

Recall that in general all untouched variables are a-predictable. In *MiniLife-Survey*, the converse happens to be true as well. With $X_1 = 3$, the only untouched variable is $f^{\mathrm{U}}(x) = \{Y_5\}$. As the robot moves from west to east, all other downstream variables have already had a chance to affect observations.

With $X_1 = 3$, the conditionally relevant variables are $f^{\mathrm{R}}(x) = \{Y_3, Y_4\}$. Y_1 and Y_2 are irrelevant because there is no way they can affect future observations or rewards. Y_5 is irrelevant because, even though it will become immediately relevant when $X_1 = 4$, it is currently untouched.

Figure 7.3 shows one example state transition for the `move` action in the abstract model. The value of Y_4 remains the same across the transition, since all the Y_i variables are static. This could be inferred directly from the original transition function T. On the other hand, the value of Y_5 was not specified before the transition. The distribution over Y_5 values after the transition is inferred from the prior probability information $\Pr(Y_5 = \texttt{L}) = 0.1$ from the initial belief. This information from b_0 was effectively folded into the transition function when T was transformed into $\tilde{T}$.

7.7 Application to *LifeSurvey*

The full *LifeSurvey* problem places the robot in a two-dimensional map. The robot must move from the west edge of the map to the east edge, but within the bounds of the map it can choose its own path. Figure 7.4 shows an example prior map. Regions of the map are indicated as contiguous groups of cells with the same shading. Each region has its own per-cell prior probability of containing detectable life. In addition, the robot receives reward only the first time it samples a cell with evidence of life in any given region. *LifeSurvey* is motivated and described in detail in §8.4.

In the interest of expediency we developed a special-purpose version of CIVA for *LifeSurvey*. The uncompressed system dynamics were expressed procedurally, rather than in a declarative representation like a decision tree or ADD. As the immediate relevance structure for *LifeSurvey* was fairly simple, we found it easiest to provide CIVA with a hard-coded conservative labeling of immediately relevant variables rather than writing a general-purpose routine to check I1-I3. Determination of predictability and conditional relevance used constraint propagation as described earlier.

The *LifeSurvey* domain is well suited to CIVA. The downstream variables in *LifeSurvey* include both per-cell variables (presence or absence of life), and per-region variables (has life been sampled in this region yet?). Only the robot's current cell and cells just ahead are relevant, and regions that the robot has permanently left behind or has yet to encounter are irrelevant.

The instance shown in Figure 7.4, called *LifeSurvey1*, had 3.5×10^{24} states in the unreduced flat representation versus 7,001 with the flat representation after CIVA. We can get better insight about the abstraction by looking at the contributions from different state variables (see detailed variable descriptions in §8.4.2). Decomposing:

- `position`. This is an upstream variable; upstream variables are never abstracted.

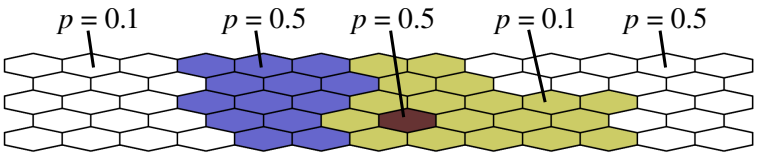


Figure 7.4: *LifeSurvey1* prior map.

- `lastMoveDirection`. Upstream variable.
- `usedScanInThisCell`. Upstream variable.
- $\texttt{lifeInCell}_c$ ($c = 1, \ldots, k$). Only (at most) the three cells immediately ahead of the rover are conditionally relevant. With k total cells in the map, abstraction reduces the number of joint assignments to the cells from 2^k to approximately 2^3. For *LifeSurvey1*, with $k = 63$, this provides the majority of the compression, a factor of about 5.8×10^{17}.
- $\texttt{rewardInRegion}_r$ ($r = 1, \ldots, n$). The only conditionally relevant regions are those that (1) might already have been sensed, given the rover's current position and (2) are still reachable from the current position.

  The amount of abstraction achievable depends on the shape of the regions; in the worst case, if regions were arranged such that the robot could drive from any region to any other, all per-region variables might be relevant simultaneously, which would lead to very little compression. However, *LifeSurvey1* is relatively benignly structured, such that in many positions only the current region is relevant.

  With n regions in the map, in the best case, abstraction reduces the number of joint assignments from 4^n to 4. For *LifeSurvey1*, with $n = 5$, this gives a best case compression factor of 256. The observed compression factor is about 200.

Combining the $\texttt{lifeInCell}_c$ and $\texttt{rewardInRegion}_r$ abstractions gives an approximate best-case compression factor of $2^{k-3} \times 4^{n-1}$. For *LifeSurvey1*, these two abstractions reduced the size of the model to 29,953 states. Finally, after discarding unreachable states, only 7,001 states remained.

7.8 Conclusions

We presented CIVA, a novel approach for losslessly compressing POMDPs with appropriate factored structure. When applied to the *LifeSurvey* robotic exploration

domain, we were able to abstract, expand, and approximately solve POMDPs whose uncompressed flat representation had up to 10^{24} states.

Effective use of CIVA relies on strong assumptions about problem structure. There must be deterministic state variables to use as upstream variables. If the *LifeSurvey* model included position uncertainty, position could not have been used as an upstream variable.

The existence of conditionally irrelevant variables tends to rely on a sense of "forward progress" through the system. If the *LifeSurvey* robot was able to move backward through the map, cells it passed by would no longer be irrelevant. If the problem was cyclical in nature, there would typically be no untouched state variables after the first cycle. (Although some variables might still be a-predictable due to other types of structure.)

Irrelevance also requires a certain amount of independence between downstream variables. If downstream variables are correlated in the b_0 prior, then they cannot be considered untouched according to our current definition (although the requirements could be relaxed in some circumstances). Similarly, downstream variables can become entangled if they affect each other's transitions or they jointly affect observations.

For these reasons, we expect that only a small proportion of interesting POMDP problems would gain significant benefit from the full CIVA compression algorithm described here. However, many more problems have approximate conditional irrelevance structure which could lend itself to lossy compression extensions of CIVA.

Overall, the reader may wish to think of this chapter less as a description of an integrated algorithm and more as a conceptual toolkit. Depending on the problem, some or all of the CIVA concepts may be applicable. For instance, the idea of rewriting the transition function based on some form of reachability analysis in order to fold in information from the initial belief and remove dependencies on the prior state may work with other types of problem structure that we have not considered.

Chapter 8

Science Autonomy

"Science autonomy" refers to exploration robotics technologies involving onboard science analysis of collected data (Castaño et al., 2003b). We demonstrated the relevance of science autonomy and POMDP planning to robotic exploration in the context of the Limits of Life in the Atacama project (Figure 8.1), a three-year effort to study techniques for robotic exploration and map the distribution of extremophile life in the Atacama Desert of Chile (Cabrol et al., 2007). In order to simulate Mars-like operational constraints, during Atacama field operations a team of geologists and biologists commanded the Zoë rover once per day from Pittsburgh (Wettergreen et al., 2005).

We performed two main science autonomy experiments (Figure 8.2). Our first experiment deployed an automatic life detection system in the field, providing us with valuable experience integrating science autonomy technology with onboard software systems and with the command cycle of remote science operations. Our second experiment tested POMDP planning technology for efficiently mapping the distribution of life in a controlled outdoor environment. Together, these experiments address different challenges that will be relevant to future field deployment of POMDP planning for robotic exploration, and for science autonomy in particular.

As we will see in our discussion of the *LifeSurvey* problem (§8.4), POMDP planning is well suited to robotic exploration domains. Unknown features of the rover's environment can be cleanly modeled as POMDP state uncertainty, and noisy sensors can be modeled with POMDP probabilistic observations. However, realistically scaled problems remain intractable. We were able to approximately solve instances of *LifeSurvey* only by combining nearly all of the POMDP planning improvements covered in this thesis. By some measures, these POMDPs are among the largest ever solved.

Figure 8.1: The Zoë rover platform in the Atacama desert.

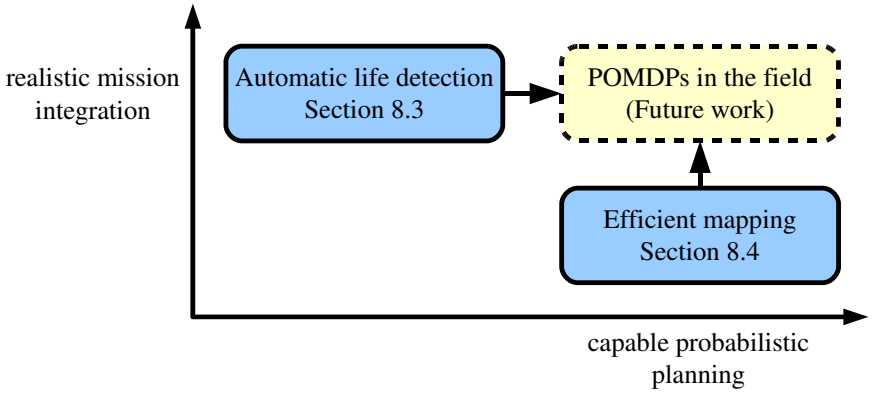


Figure 8.2: Our experiments build towards field deployment of POMDP planning for robotic exploration.

8.1 Related Work on Science Autonomy

There has been considerable previous work on science autonomy; we briefly touch on relevant research in several areas.

8.1.1 Onboard Science Data Analysis and Selective Data Return

Science autonomy systems often need to identify interesting targets for potential sensor measurements. Several groups have used stereo images to find rocks based on their height above the ground plane (Gor et al., 2000; Fox et al., 2002; Pedersen,

2002). These methods can reliably find large rocks on flat ground, but they cannot usually identify small or partially buried rocks and their range is limited to around 10 m. A second method reduces rock detection to the simpler problem of finding shadows (Gulick et al., 2001). This strategy finds a point on the rock's surface but does not find the outline of the rock, which can make subsequent classification more difficult. Several other methods detect rocks based on their contrast with the background in terms of albedo, color, or texture. For example, Castaño et al. (2004) look for closed shapes with an edge detector. This technique works best for finding a small number of rocks with high accuracy.

One can think of images as consisting of multiple channels of information, such as intensity, shading and stereo disparity. Rocks that are clearly distinct from background in one channel may not be in another. Thompson et al. (2005a) developed a detection system that segmented distinct regions on several channels, then passed all the regions to a Bayes net that classified them as rocks, shadows, or non-rocks. The classifier returned meaningful probabilities that could be used to trade off precision vs. recall.

Given rock outlines, one can automatically calculate shape measures like circular variance and Diepenbrök roundness. Dunlop (2006) suggests that these measures correlate well with Crofts' roundness and sphericity measurements, which in turn are useful indicators of how rocks were created and transported. Spectral properties also offer mineralogical information for geologic classification. The Robotic Antarctic Meteorite Search (Pedersen et al., 2001) integrated spectroscopy and visual imagery to find and classify meteorites during robotic traverse on the Elephant Moraine plateau. Pedersen (2001) demonstrates statistical models of the local environment that use contextual cues to improve classification performance by the explorer robot. More recently, Bornstein and Castaño (2005) have demonstrated automatic classification of carbonate minerals using spectral characteristics.

A natural extension of autonomous rock detection and classification is the generation of automatic geologic signatures—profiles describing the distribution of rock classes at a site (Thompson et al., 2005b). These geologic signatures reveal subtle geologic trends, the borders between geologic units, and sudden changes compared to neighboring locales. All of these onboard analyses can inform adaptive measurement or return decisions. An experiment analyzed a traverse consisting of five neighboring locales in the Atacama Desert. It suggested that site profiles based on autonomous rock detection correlated with the principal distinctions in surface composition identified by a human at the site. Later, Thompson et al. (2006) detected changes in geological profiles over a long traverse and used them to prioritize novel images for data return.

8.1.2 Scientist Priorities

Scientist priorities specify which types of features or datasets are most interesting and thereby determine the goals of a science autonomy system. Chien et al. (2005) describes a simple preference model that does not distinguish the relative priorities of different feature classes. This strategy was used for the Autonomous Sciencecraft Experiment conducted aboard the Earth Observing One (EO-1) orbiter. Transient phenomena such as floods, fires, and volcanic activity were captured using autonomous retargeting of a high-resolution imager, where manual retargeting would have been too slow.

Castaño et al. (2003b) proposed three more expressive models for science value. Scientists using the target signatures strategy assign priority scores to particular classes of features. If the rover downlink budget is oversubscribed, the rover can use the scores to return the most informative subset of the collected data. This approach has the disadvantage that scientists can only prioritize classes of features that they can anticipate in advance. Using the representative sampling strategy, the rover builds a statistical model for observed features on the fly. It groups the data into clusters with the object of returning a data set containing examples from each significant group of observed features. Using the novelty detection strategy, the rover prioritizes features that do not fall into any known class, presuming that these features are interesting precisely because they are unusual.

Smith et al. (2005) proposed an integrated approach encompassing all three of these strategies. It worked by supplementing prespecified target signatures with new target signatures for unanticipated classes. The new target signatures were prioritized based on scientist interest in representative sampling and novelty detection.

8.1.3 Science Autonomy Planning Systems

There has been considerable research on planning systems for planetary surface rovers (Estlin et al., 2002; Urmson et al., 2003; Tompkins, 2005), achieving notable success with the MAPGEN and GESTALT planners used by the Mars Exploration Rovers (Bresina et al., 2005; Maimone et al., 2006). Our discussion will focus narrowly on controllers that either use probabilistic planning or have the ability to react to science observations collected on the fly.

The Antarctic meteorite search of Pedersen (2001) selected sensing actions greedily in order to reduce the uncertainty in rock/meteorite classification according to a Bayesian network model. Although its action selection was myopic, this system was notable for its principled use of a probabilistic model to handle uncertainty.

Moorehead (2001) studied prioritized coverage planning for rover exploration. The planner started with a partial map and executed a coverage pattern by greedily selecting motion actions to maximize information gain according to specified metrics, for instance determining traversability or observing as much terrain of a specific type as possible.

Estlin et al. (2003) developed a planner for inserting opportunistic science actions into a realistic rover command sequence. Their CASPER planner is sophisticated enough to reason about such issues as detailed path planning, instrument warm-up periods, and instantaneous power constraints that prevent rover subsystems from operating simultaneously. It successfully inserted rock measurement actions into a rover traverse as part of an integrated demonstration of the OASIS system in a controlled outdoor environment. CASPER was also used to insert opportunistic science actions into observation plans onboard the EO-1 spacecraft (Chien et al., 2003).

Dearden et al. (2003) generated contingent plans that were robust with respect to uncertainty about the resources that actions would consume. One could use the same contingent plan representation to build in slack for opportunistic science.

Smith (2004) developed a system for optimally dropping and reordering actions when rover resources are oversubscribed. Pedersen et al. (2006) described an integrated demonstration using Smith's oversubscription planner to insert new science tasks on the fly as they were requested by the human members of a human-robot team.

8.2 Robotic Investigation

Beginning in 2005, Carnegie Mellon's "Science on the Fly" project investigated science autonomy technologies for use in planetary science applications. It focused on capabilities for science operations that involve long over-the-horizon traverses, including autonomous detection and classification of rocks, autonomous spectroscopy, science goal representations and planning. A partnership with the "Life in the Atacama" (LITA) project, funded by NASA's ASTEP astrobiology program, provided a unique opportunity to test autonomous science technologies as an integrated part of an exploration robotics field campaign.

The LITA project aimed to survey the distribution of extremophile life and habitats in the Atacama, a Mars-analog desert in Chile. The three Atacama field campaigns visited six field sites (Figure 8.3): sites A, B and D were located in the "humid zone" near the Coastal Range; sites C and F were in the arid core of the desert; site E was deeper into the central depression. Scientists explored the desert remotely using an autonomous rover commanded from North America. Re-

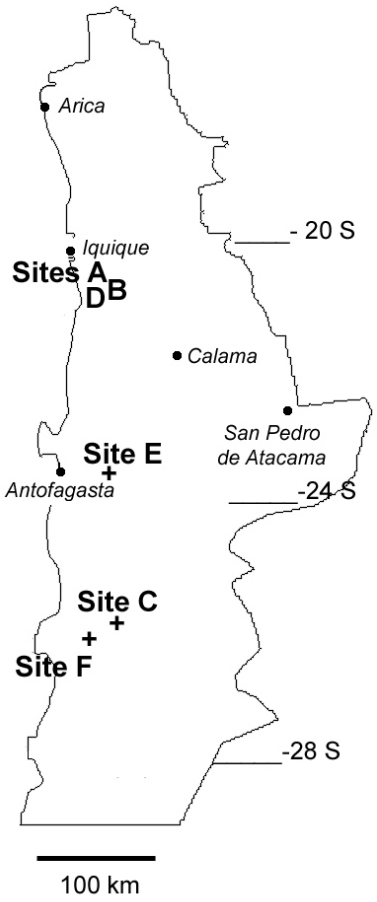


Figure 8.3: Map showing the location of LITA field campaign sites. From Cabrol et al. (2007).

mote operations enforced space-relevant constraints such as limited bandwidth and a single daily command cycle. This permitted research into extremophile habitats while simultaneously testing operational strategies to exploit navigational autonomy in fast rovers (Cabrol et al., 2007).

The project utilized Zoë, a solar-powered rover capable of single-command autonomous navigation of multiple kilometers (Figure 8.1). Its instrument suite included a mast-mounted pan-tilt unit with narrow-field panoramic cameras (pancams) and fixed forward-facing navigation cameras (navcams). It also carried

Figure 8.4: The microscopic fluorescence imager deployed and spraying underneath the robot.

an integrated microscopic imager capable of detecting chlorophyll or bacterial colonies. An onboard sprayer deposited fluorescent dyes which would bond to living organisms after a short incubation period (Figure 8.4). The microscopic imaging apparatus would induce fluorescence with a flashlamp. Subtracting ambient light left only the fluorescence from bonded dyes, an *in situ* image of any organisms that were present. Finally, the science team used orbital imagery from ASTER (Abrams, 2000), Hyperion (Pearlman et al., 2001), and IKONOS (Dial et al., 2003) as an integral part of planning over-the-horizon traverses. Details about science instrument payload are given by Cabrol et al. (2007) and Waggoner et al. (2007).

The science goals of the LITA project were particularly amenable to science autonomy. The scientists attempted to complete a wide-area biogeologic survey in a short time by visiting new environmental units whenever possible. The data collected during these traverses involved areas that scientists had not seen previously. However, they expected that extremophile habitats would be distributed in isolated patches and oases of life that could be targeted with a "follow-the-water" strategy.

Because of the sparsity of microhabitats, data collected during a long traverse were expected to contain a few positive images containing life mixed with a large number of negative images. The extensive measurements required to verify life at any single location were expensive in terms of rover time and limited amounts of experimental dye resources. It was hypothesized that science autonomy could manage these resources more efficiently by performing life-detection experiments selectively in response to more expedient observations.

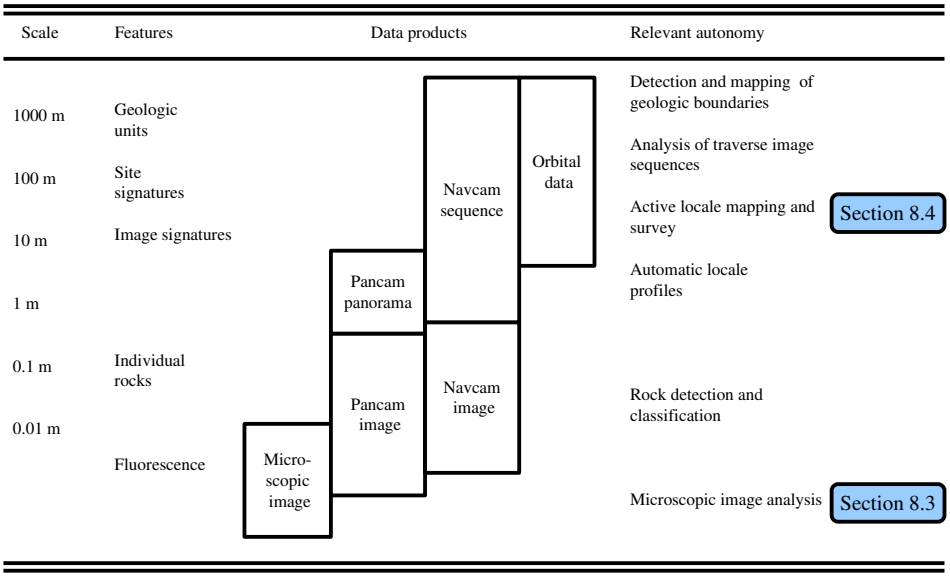


Figure 8.5: Science autonomy experiments in the LITA project dealt with instruments and features at multiple scales.

We present experiments in two specific autonomous science domains (Figure 8.5). §8.3 demonstrates autonomously responding to evidence of life with followup measurements, specifically reacting to chlorophyll fluorescence detected using a microscopic imager. §8.4 details probabilistic planning to efficiently map the distribution of life. Here, an active mapping system moves the rover to new locations in response to collected data.

8.3 Autonomously Responding to Evidence of Life

The LITA campaign strategy for unambiguously detecting life was to search for spatially correlated fluorescence signals from several distinct types of biomolecules using the robot's onboard microscopic fluorescence imager (FI). The full protocol of FI measurements at a locale was resource-intensive, taking approximately 23 minutes and using fluorescent dye from a limited onboard stock.

However, since the emphasis was on finding locales with positive signal from as many distinct biomolecules as possible, we were able to improve overall efficiency by terminating the protocol early if initial signals came back negative. In this way we saved rover resources without sacrificing high-value unambiguous de-

tections of life.

We divided the full protocol into two segments: (1) Check quickly for natural chlorophyll fluorescence; (2) Only if chlorophyll was detected through onboard image analysis, autonomously respond by performing followup measurements with dyes to detect other biomarkers. In the common case that the result from step (1) was negative, the modified protocol took only eight minutes to complete and did not consume any dye. Our field evaluation confirmed that the improved protocol significantly improved overall efficiency.

8.3.1 The Fluorescence Imager (FI) Instrument

Figure 8.4 shows the FI, a down-pointing camera mounted on the bottom of the Zoë rover (Waggoner et al., 2007). It has a 10 cm field of view and transverse resolution of 210 μm. During autonomous response experiments, the sampling location under the FI was chosen by stopping the rover at fixed distances along its traverse, and the camera was deployed and auto-focused using z-axis motion.

The FI could detect either the reflectance or fluorescence of a sample in various channels. A xenon flashlamp provided illumination. Servos could select one of six optical interference filters for the excitation path between the flashlamp and sample, and one of ten filters for the emission path between the sample and CCD.

The FI captured reflectance under a combination of sunlight and flashlamp illumination with no excitation filter. In RGB color mode, separate images with red, green, and blue emission filters combined to create a visual color image. In fluorescence mode the FI captured a greyscale intensity image with excitation and emission channel pair selected to respond to the fluorescence of the molecule under study, either naturally fluorescent chlorophyll or an artificial marker dye whose fluorescence increased when bound with a biomolecule. Different marker dyes responded to DNA, proteins, lipids, and carbohydrates. An automatic sprayer could spray the sample with water to enhance chlorophyll fluorescence under dry conditions. It could also spray a solution that contained all four marker dyes along with acid and detergent to aid dye penetration.

8.3.2 Chlorophyll Detection Experimental Procedure

The FI protocol for each sample had two phases: (1) The FI sprayed water and captured several images, including a chlorophyll image used to determine if followup was warranted. (2) It sprayed the dye mixture and captured several images, including dye fluorescence images. Execution times for the phases were approximately 8 and 15 minutes, respectively.

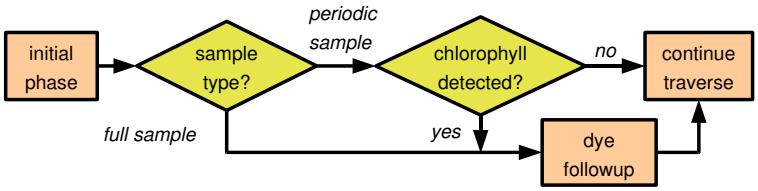


Figure 8.6: Sample protocol flowchart.

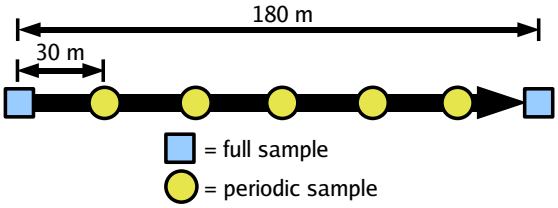


Figure 8.7: Periodic sampling traverse.

Scientists could command two types of sample. A full sample always included both phases of the protocol. A periodic sample always included phase 1, but continued to phase 2 only if chlorophyll was autonomously detected (Figure 8.6).

Scientists used the autonomous followup capability during 180 m periodic sampling transects (Figure 8.7). At each endpoint of the transect, the rover stopped and took a full sample. At 30 m intervals within the transect, the rover stopped and took a periodic sample. The rover executed each traverse autonomously within a single command cycle (including both driving and FI samples).

In keeping with the LITA high-mobility exploration strategy, the science team decided to add a followup quota ensuring that the rover would not dwell too long on a particular transect. At most three periodic samples per transect could trigger followups. After filling this quota the rover would not spray dye on subsequent samples even if it detected chlorophyll.

8.3.3 Chlorophyll Detection Image Analysis

Autonomous chlorophyll detection, used only for periodic samples, relied on a single image of chlorophyll fluorescence intensity (excitation 450 nm, emission 740 nm) captured after the FI sprayed water on the sample. This was called the "trigger image" (Figure 8.8).

The detection algorithm reported the probability that chlorophyll was present anywhere in the image, triggering an autonomous dye followup if the probability was 50% or higher. The algorithm reported a high probability if there were any

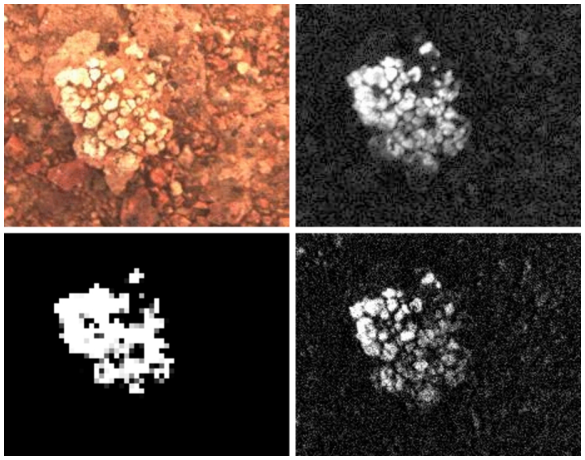


Figure 8.8: Autonomous chlorophyll detection: (top left) Portion of FI visual color image containing a lichen. (top right) Chlorophyll trigger image. (bottom left) An intermediate step of image processing; the brightness in each cell represents the estimated probability that it contains chlorophyll. (bottom right) After autonomous followup, the FI detected fluorescence from the DNA marker dye.

bright patches in the image. First it formed a sub-sampled image by splitting the original image into 4×4 cells and calculating the mean intensity $A(x, y)$ over each cell. This smoothing eliminated false detections from single-pixel shot noise. Second, it converted mean intensity for each cell to a probability $L(x, y)$ that the cell contained chlorophyll using a sigmoid or "fuzzy threshold" function $\sigma_{\alpha\beta}$. Finally, it calculated the overall probability L of chlorophyll being present in any cell of the image by combining the probabilities from individual cells using the heuristic assumption that the $L(x, y)$ measurements were uncorrelated.

Let $I(x, y)$ denote the pixel intensity at position (x, y) in the original image. We have

$$A(x,y) = \frac{1}{16} \sum_{\Delta x=0}^{3} \sum_{\Delta y=0}^{3} I(4x + \Delta x, 4y + \Delta y) \tag{8.1}$$

$$L(x,y) = \sigma_{\alpha\beta}(A(x,y)) = (1 + \exp(\alpha + \beta A(x,y)))^{-1} \tag{8.2}$$

$$L = 1 - \prod_{x,y} (1 - L(x,y)). \tag{8.3}$$

The fuzzy threshold function $\sigma_{\alpha\beta}$ used to convert cell intensity to probability of containing chlorophyll had two parameters α and β whose values were tuned using

training data. The training data were gathered by manually labeling the presence or absence of chlorophyll signal in individual 4×4 cells of two trigger images containing lichens, based on morphological cues from both the trigger image and an associated visual color image.

Let $n = 2$ denote the number of training images, let $A_i(x, y)$ denote the mean intensity in cell (x, y) of the subsampled version of the ith labeled image, and let $L_i^*(x, y)$ denote the corresponding manual label, with value 1 or 0 indicating the presence or absence of chlorophyll in the cell. The parameters α and β were set to maximize the likelihood of the manual labels using logistic regression.

$$(\alpha, \beta) \quad = \quad \arg \max_{(\alpha', \beta') \in R^2} \prod_{i=1}^{n} \prod_{x,y} \Big(1 - |L_i^*(x, y) - \sigma_{\alpha'\beta'}(A_i(x, y))| \Big) \quad (8.4)$$

While every labeled training image contained significant instances of chlorophyll fluorescence, many images from the Atacama Desert did not. To account for this discrepancy we included an additional parameter τ that represented a prior on the probability of finding chlorophyll in the image (equivalently, the proportion of the ground surface expected to be covered by photosynthetic organisms). This transformation, applied after training is complete, yields

$$L(x, y) \quad = \quad \frac{\tau \sigma_{\alpha\beta}(A(x, y))}{\tau \sigma_{\alpha\beta}(A(x, y)) + (1 - \sigma_{\alpha\beta}(A(x, y)))} \quad (8.5)$$

The value of L was then calculated from $L(x, y)$ using eq. (8.3) as before. We informally hand-tuned τ on a set of test images so that most images would fall on the correct side of the followup threshold (negative images below 50%, positive images above 50%). The resulting value of 0.005 was used for all of the reported results.

8.3.4 Chlorophyll Detection Experimental Results

The autonomous response system was evaluated onboard the rover during the 2005 Atacama field campaign. Our data set included 24 periodic samples collected during five transects. Periodic samples occurred at 30 m intervals and the standard transect length was 180 m, so nominally each transect should have included five periodic samples. In practice, the commanded lengths of the transects varied due to small positioning errors in the operator interface. As a result, transects could contain 4-6 periodic samples. Some transects were cut short for other reasons. Nightfall ended the last transect after only two periodic samples. In two of the

transects the rover filled the followup quota before the last periodic sample; we excluded later periodic samples from the analysis because the quota prevented any further followups.

Each sample image set was analyzed by a remote team that included field biologists and fluorescence experts. Using both the visual color image and the trigger image, they labeled the samples as positive (contains significant evidence of life) or negative (does not). We compared the scientist labels to the autonomous followup response.

The autonomous system and expert labels agreed for 19 of the 24 samples. 8 of the 24 samples were positive; 7 of the 11 samples chosen for autonomous followup were positive. Thus the yield, or proportion of dye samples that occurred after positive evidence of chlorophyll, was 90% higher when applying the science-aware system than would have been expected for randomly selected samples (significance level < 0.01 using one-tailed Fisher's exact test). In other words, relative to a baseline strategy of always applying dye, the science-aware system skipped 13 dye samples, saving more than three hours of robot time, while following up chlorophyll detection correctly in 7 out of 8 cases.

We were able to diagnose the cause of failure in some but not all cases. For example, a false positive could occur when the FI was dazzled by direct sunlight (normally it is shaded by the robot). A false negative could occur when the water spray failed to reach the sample due to high winds. These failures could be addressed by hardware modifications or by limiting FI operations to certain times of day. The image analysis could also be made more robust to corrupted images by diagnosing problems like dazzling and poor focus based on the image, or by adding more definitive cues such as squamous lichen morphology to the analysis.

8.4 Efficiently Mapping the Distribution of Life

One of the ultimate goals of rover exploration is to build maps that relate to models of the environment. For instance, in the Atacama, surface habitats for lichens and bacteria can be created by terrain features that locally modify air flow and insolation. The interaction of these variables with the presence of life is poorly understood. We propose a form of representative sampling called "sampling by regions" to efficiently answer this type of question.

Under the sampling by regions strategy, regions with homogeneous properties are identified in orbital imagery. The rover is used to characterize aspects of each region that cannot be measured from orbit. In the Atacama, orbital data can be used to study frequency of cloud cover, landforms that control airflow, and average slope, which affects insolation and wind exposure. One can identify local regions

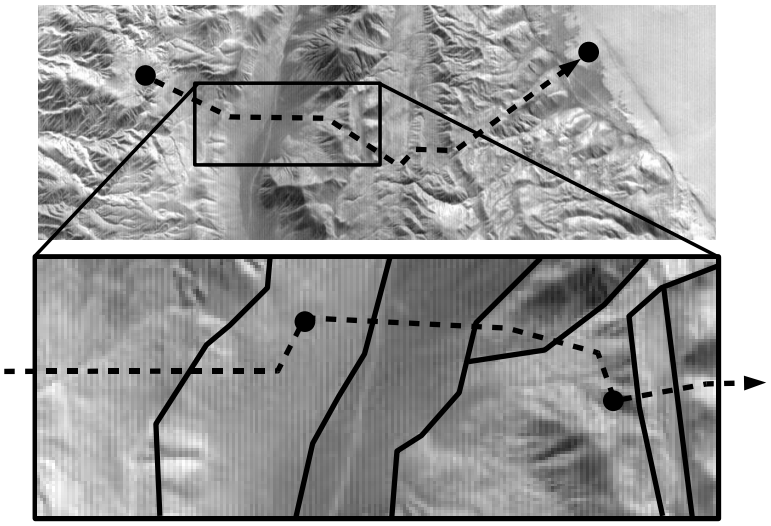


Figure 8.9: Representative sampling strategy at multiple scales.

that are relatively homogeneous with respect to these properties, and then address the properties' relationship to viable habitats by using the rover to characterize the presence or absence of life on a per-region basis.

8.4.1 Mapping Scenario

To operationalize sampling by regions, one can describe the environment using the different size scales shown in Figure 8.9. At large scale (kilometers, upper map), scientists designate a path based on long-term mission goals. At small scale (hundreds of meters, lower map), the rover's path should be optimized to travel through regions relevant to the modeling objectives. In the lower map, region boundaries are indicated with solid lines and the rover's path is marked with a dashed line.

Within the rover's field of view (tens of meters) in the small scale map, the rover should scan the environment and react appropriately if it detects cues associated with life, such as visible plants or signs of gully erosion. This reaction might include taking a detour toward the feature and gathering additional data to confirm the presence of life. In the lower map, detected features are marked with black circles; note that the rover has adjusted its path to visit them.

We developed an onboard system that intelligently controls rover motion and sensing at a small scale, while respecting the constraints of the large scale plan specified by scientists. The first step is off-board probabilistic planning based on

large-scale goals and a region map provided by scientists. This generates a robust policy that specifies how to react to different potential science instrument readings during execution. During traverse the rover refers to this policy to choose appropriate actions based on the actual instrument readings. In effect, it continuously replans the remainder of the traverse based on what it sees.

The probabilistic planning system attempts to generate the most efficient exploration policy. In order to define optimality, where there are multiple interacting goals, each goal must be assigned a relative priority that allows the system to perform appropriate trade-offs. Our scenario encodes these priorities by specifying a per-region reward for finding evidence of life and penalties for taking time-consuming actions.

In addition, one must specify how to deal with uncertainty. If a plan includes searching a region for life, and it is unknown whether the region contains life, one cannot evaluate in advance how successful the plan will be. We choose to model the problem as a POMDP, which means we take a risk-neutral decision-theoretic perspective toward uncertainty. Uncertain aspects of the environment model and uncertain outcomes of rover actions are modeled probabilistically, and the optimal plan is the one that has the highest expected or "average" value.

8.4.2 *LifeSurvey* Problem Definition

We evaluated our tactical replanning system onboard the Zoë rover in a controlled outdoor test site in Pittsburgh (Figure 8.10, right). This allowed us to simplify several aspects of the problem. First, since there were no appropriately distributed natural features in our test area, we used simple artificial targets (10 cm squares of white posterboard) as a stand-in for evidence of life.

Second, rather than generating a region map based on satellite images of the test area, we generated an arbitrary region map by hand and then modified the test area to match the map. To be more precise, since the region map only specifies the likelihood of life in each cell, we randomly generated multiple target layouts for each region map, using the specified likelihoods.

Figure 8.11 shows two region maps and two corresponding randomly drawn target layouts for each map. Regions are indicated as contiguous groups of cells with the same shading (some shading tones are reused). Each region is marked with the probability that cells in that region will contain evidence of life. White squares in the target layouts indicate the locations of artificial targets.

In keeping with the overall scenario, the region map represented a segment of a long traverse. The rover started at the left side of the map and had to eventually exit from the right side in order to conform to the large scale plan.

Figure 8.10 (left) helps to explain the actions available to the rover at each

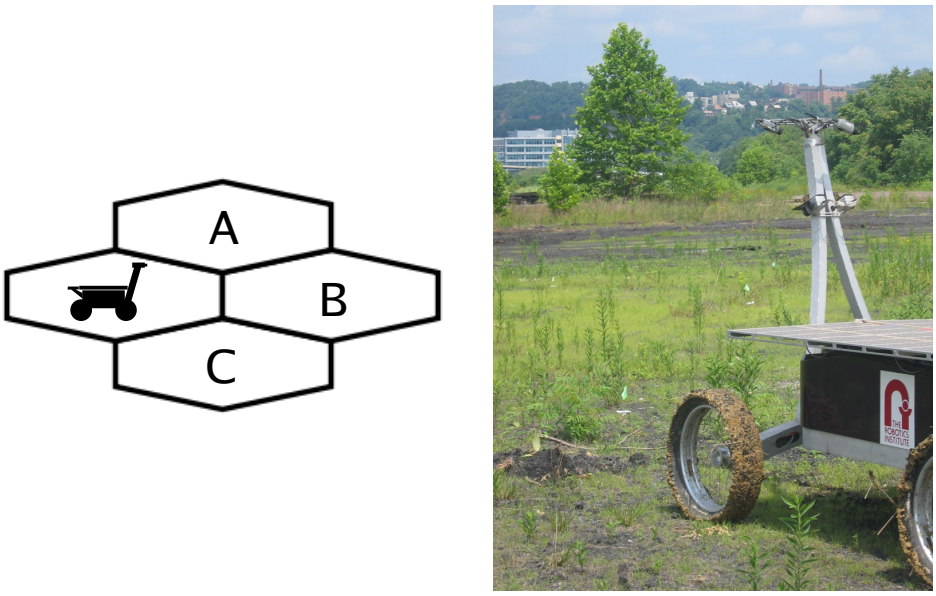


Figure 8.10: *LifeSurvey*: (left) Actions available to the rover. (right) The Zoë rover at the test site in Pittsburgh.

step of execution. In a single action, the rover could either (1) scan all three of the marked cells, returning a noisy signal as to whether they contain life, or (2) perform a simple move or sampling move to any one of the marked cells. Sampling moves differed from simple moves in that they caused the rover to take additional detailed measurements as it entered the new cell. They were intended to confirm the presence of life.

The planning objective was to maximize expected reward. The rover received a single reward for each region: +50 points if it performed a sampling move into a life-containing cell in the region; otherwise, +20 points if it passed through a life-containing cell in the region; otherwise, +5 points if the rover entered the region; otherwise, no reward. Each action incurred a cost: -1 point for each move, and -5 points for each scan or sampling move. Thus the rover needed to find confirmed evidence of life in as many regions as possible, while minimizing the number of detours, scans, and sampling moves.

The observation returned by the scan action was a tuple of three independent readings for the cells A, B, and C. The possible values for each reading could be interpreted roughly as “negative”, “maybe”, or “positive”, corresponding to different confidence levels from the onboard detection routine searching for artificial

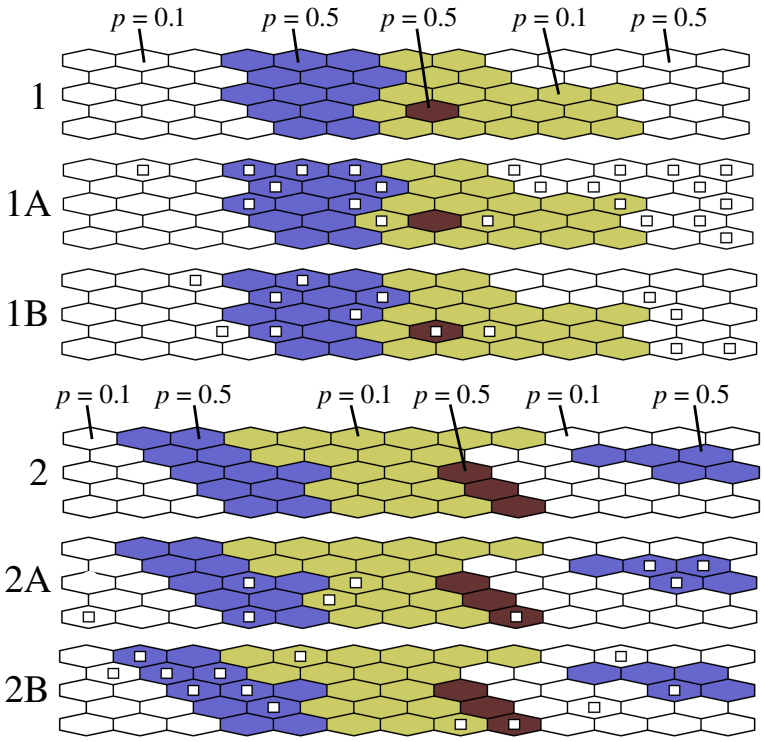


Figure 8.11: *LifeSurvey* region maps and target layouts.

markers.

The sensor noise parameters used in the planning model were learned from a training set that included detection routine outputs and ground truth labels gathered over several runs in the testing environment. Cells without markers returned negative/maybe/positive readings roughly 72%/12%/16% of the time, respectively; cells with markers had the distribution 9%/5%/86%.

Cells of the map were hexagons, laid out in rows that ran west to east. Compared to a rectangular grid, the hexagonal grid layout had the advantage that there was a clearer correspondence between the cells that the rover moved through and the cells covered by its navcam imagery. In contrast, when moving diagonally in a rectangular grid, the rover would image small portions of several cells as it passed over a cell corner, making it more problematic to say which cells it sensed.

Due to the relatively small size of the map cells in our experiments, the rover could not turn sharply enough to move in arbitrary directions. We handled this problem in two ways. First, the centers of adjacent cells in a row were 10 m apart

and the rows were 3.5 m wide, leading the individual hexagons to have a somewhat "squashed" aspect ratio. These proportions meant that the motion direction for northeast and southeast moves deviated from east moves by an angle of only 35°.

Second, the planner was prohibited from commanding sharp turns. The rover could not travel northeast and then southeast (or vice versa) on consecutive moves; an east move needed to be present in between. This was handled in the planning model by adding an extra state variable recording the direction of the most recent move and penalizing any motion action inconsistent with the turning constraint.[1]

Readers interested in using the `LifeSurvey` problem can download CIVA-compressed flat models in Cassandra's POMDP model format, along with source code for generating new problem instances, as part of our freely available ZMDP software package (Smith, 2006).

We close the *LifeSurvey* discussion with a more formal specification of the variables involved. We have:

- **Actions.** There are seven total actions: three non-sampling move actions (`NE`, `E`, and `SE`), three sampling move actions (`NE+sample`, `E+sample`, and `SE+sample`), and one `scan` action.

- **Observations.** There are 27 distinct observations. A single `scan` action returns a tuple of three independent readings, one for each of the three cells immediately ahead of the rover. Each cell reading is either `negative` ($-$), `maybe` ($?$), or `positive` ($+$). Thus there are $3^3 = 27$ observations in all, ranging from $\langle -, -, - \rangle$ to $\langle +, +, + \rangle$. Actions other than `scan` always return a null observation; in order to avoid increasing the number of observations, our convention is to represent the null observation as $\langle -, -, - \rangle$.

- **State variables.** Let there be k total cells in the region map, forming n regions. The state of the system can be factored into the following state variables:

  - `position`. Specifies the cell the rover currently occupies. With k cells in the region map, `position` can take on k possible values.
  - `lastMoveDirection`. Specifies the direction of the rover's last move. This variable is needed only to enforce the rover's non-holonomic motion constraint. Can take on three possible values (`NE`, `E`, or `SE`).

[1] In POMDP planning it is generally problematic to prohibit use of an action in particular states. What should happen when, for example, there is a 10% chance that the system is in a state where the action is prohibited? In part because of this potential for semantic confusion, some model specification languages (including the one we use, developed by Anthony Cassandra) do not provide a way to specify states in which an action is prohibited. We usually work around this limitation by specifying that illegal actions have no effect (return to the same state) and incur a large penalty.

- `usedScanInThisCell`. Specifies whether the `scan` action has been used yet in the current cell. Can be `true` or `false`. Onboard the rover multiple scans from the same cell typically give identical results, so that after the first scan subsequent scans are useless. We added the `usedScanInThisCell` variable to prevent the planner from trying to use redundant scans. In the planning model, if `usedScanInThisCell` is true and the `scan` action is applied, it returns a null observation and incurs a large "illegal action" penalty.
- $\texttt{lifeInCell}_c$ ($c = 1, \ldots, k$). Specifies whether cell c contains life. Can take the values `life` (`L`) or `no-life` (`N`). With k cells in the region map, there are 2^k possible joint values for the `lifeInCell` variables.
- $\texttt{rewardInRegion}_r$ ($r = 1, \ldots, n$). Specifies the largest reward level achieved by the rover so far in region r. Recall that the rover can get different rewards for characterizing a region in different ways (+0, +5, +20, or +50), but in the end receives only the most valuable single reward from each region. The `rewardInRegion` variable is needed to keep track so that rewards are not double-counted. With n regions, there are 4^n possible joint values for the `rewardInRegion` variables.

Thus the number of possible joint assignments to all state variables is $|\mathcal{S}| = k \times 3 \times 2 \times 2^k \times 4^n$. For the two region maps used in our experiments:

- The *LifeSurvey1* problem instance uses region map 1 from Figure 8.11. It has $k = 63$ and $n = 5$, which gives $|\mathcal{S}| = 3.57 \times 10^{24}$ in the naive flat model representation. After CIVA compression (see Chapter 7), the compressed flat model has $|\mathcal{S}| = 7,001$.
- The *LifeSurvey2* problem instance uses region map 2 from Figure 8.11. It has $k = 63$ and $n = 6$, giving $|\mathcal{S}| = 1.43 \times 10^{25}$. After CIVA compression, $|\mathcal{S}| = 7,841$.

8.5 Experimental Evaluation

We compared the performance of three planners on several versions of the *LifeSurvey* problem, both onboard the robot and in simulation.

8.5.1 *LifeSurvey* Planners: Blind, Reactive, and POMDP

We evaluated three planners. Under the blind planner, the rover simply moves to the right in a straight line, always using sampling moves. The blind planner's policy confirms the presence of life only if it is found on the straight-line path.

Under the reactive planner, the rover follows a set of hand-generated rules designed to confirm the presence of life through combined use of scanning and sampling. At the beginning of the run, the reactive planner selects a *reference path* to use throughout the run (the selection process is explained later).

Once the reference path is selected, the policy is extremely simple. The rover moves along the reference path, scanning after every move action. If a cell returns a positive scan result, the rover performs a sampling move to the relevant cell, detouring off the reference path if necessary. (However, cells with a positive scan reading are ignored if they are in a region where the rover believes it has already confirmed the presence of life.) After taking a detour the rover drives back towards the reference path, as long as that does not conflict with sampling cells that return a positive scan reading. This kind of simple reactive policy is often appealing to domain experts because it is so easy to understand.

The most complicated part of the reactive planner is the initial selection of the reference path. The planner calculates a heuristic value for each path and selects the path with the highest value. The heuristic value of a path is an estimate of the expected number of regions where the presence of life will be confirmed if the rover follows that path, under some strong simplifying assumptions.

The heuristic (1) calculates the number $n_r(x)$ of distinct cells that will be scanned in each region r if the rover follows path x (ignoring the possibility of detours), and (2) assumes that if any of the scanned cells in region r contains life, the rover will confirm the presence of life in region r (ignoring the effect of sensor noise in the scan action). Let p_r be the prior probability that any individual cell in region r contains life. Under these assumptions, the expected number of regions where the presence of life will be confirmed is

$$V(x) := \sum_r [1 - (1 - p_r)^{n_r(x)}]. \tag{8.6}$$

Given a map on the scale of region maps 1 and 2 (see Figure 8.11), the number of paths through the map is large enough that brute force evaluation of $V(x)$ for all paths is intractable. However, by structuring the search appropriately and using an admissible pruning technique, we were able to find the reference path maximizing $V(x)$ after evaluating only a few thousand paths, taking less than two seconds.

The third planner generated an approximately optimal POMDP policy. The *LifeSurvey* POMDP problems we worked with were extremely challenging. All of

Planner	Search actions	Regions confirmed	Reward
Blind	12.0 ± 0.0	2.5 ± 0.0	68 ± 0
Reactive	20.0 ± 0.7	$\mathbf{3.4} \pm 0.6$	61 ± 19
POMDP	$\mathbf{7.5} \pm 1.0$	3.0 ± 0.5	$\mathbf{113} \pm 16$

Table 8.1: *LifeSurvey* onboard testing: planner performance.

our computation was performed on a 3.2 GHz Pentium-4 processor with 2 GB of main memory.

We refer to region map 1 from Figure 8.11 as the *LifeSurvey1* problem. With a naive flat model representation, *LifeSurvey1* would have had 3.5×10^{24} states. We used the CIVA state abstraction method presented in Chapter 7 to generate a compressed flat model with only 7,001 states. Less than two seconds were required to generate and write out the compressed model.

We approximately solved the compressed POMDP using focused value iteration. The `mask/prune` representation was used for the lower bound, the `mask` representation was used for the upper bound, and FRTDP was used for heuristic search. After 10^3 seconds (about 20 minutes) of wallclock time, the solver was able to produce a policy whose expected long-term reward was guaranteed to be within 20% of the optimal policy.

Chapters 4 and 6 also report comparative performance for other variants of focused value iteration as applied to the CIVA-compressed *LifeSurvey1* problem instance.

8.5.2 Onboard Testing

The reactive and POMDP planners were each evaluated on 20 runs of the robot through the test course. Two region maps were used, designated "1" and "2"; thus each planner generated two policies, one for each map. Four target layouts were randomly drawn, two for each region map, designated "1A", "1B", "2A", and "2B". For each planner, the robot ran the test course five times through each of the four target layouts, giving the total of 20 runs per policy. Figure 8.11 shows the region maps and target layouts.

The blind planner was evaluated on the same four target layouts in simulation, and the results are reported alongside the onboard results for the other planners; this comparison is valid because the only source of randomness from the environment is sensor noise, and the actions selected by the blind policy do not depend on observations.

Results are shown in Table 8.1. All reported values are mean values across the

20 onboard runs. We also report a 95% confidence interval for the mean value, estimated using the bootstrap method. Note that, since the reward received by the blind policy is a deterministic function of the target layout, the blind policy results could be estimated exactly by averaging over the four target layouts.

"Search actions" gives the number of scan and sampling move actions used per run (smaller values are better). "Regions confirmed" gives the number of regions in which the presence of life was confirmed with a sampling move action (higher values are better). "Reward" reports the mean reward received (higher values are better).

The POMDP policy performed best in terms of search actions and reward, by statistically significant margins. The reactive policy confirmed the presence of life in more regions, but at the cost of many more search actions than the other policies.

8.5.3 Simulation Testing: Adapting to Changes in the Problem

The POMDP planner has the desirable property that it is *model-based*, meaning that its policy is completely derived from the system model, with no "hidden parameters" that must be manually adjusted in order to achieve good policy quality if the problem changes. We hypothesize that this model-based planning approach should provide better robustness across changes in the problem than the blind and reactive planners.

To test this hypothesis, we compared the performance of the three planners under different variations of the *LifeSurvey1* problem (that is, the *LifeSurvey* problem with region map 1). Specifically, we adjusted the relative magnitude of costs and rewards; all action costs were multiplied by a constant factor c, called the *cost multiplier*. We tested five settings of c ranging from 0 to 2. With $c = 0$ all actions are free, with $c = 1$ we recover the original *LifeSurvey* problem, and with $c = 2$ actions are significantly more costly than usual. We expect that changes to c in turn change the optimal policy by making more or fewer search actions worthwhile in the sense that their expected reward outweighs their cost.

We used the POMDP planner to generate five policies, one for each setting of c. As before, the POMDP planner was run on a 3.2 GHz Pentium-4 processor with 2 GB of main memory. We ran the planner for 10^3 seconds, about 20 minutes, to generate each policy. (Note that, in between the onboard testing and simulation testing, we made minor improvements to the POMDP planner implementation, allowing it to converge slightly faster.) The blind and reactive planners are unaffected by c, so their policies were the same across all versions of the problem.

Each policy was tested on each setting of c for 10^5 runs in simulation. All simulation runs used the same region map, with a new target layout drawn randomly for each run. The system model used in simulation was identical to the model

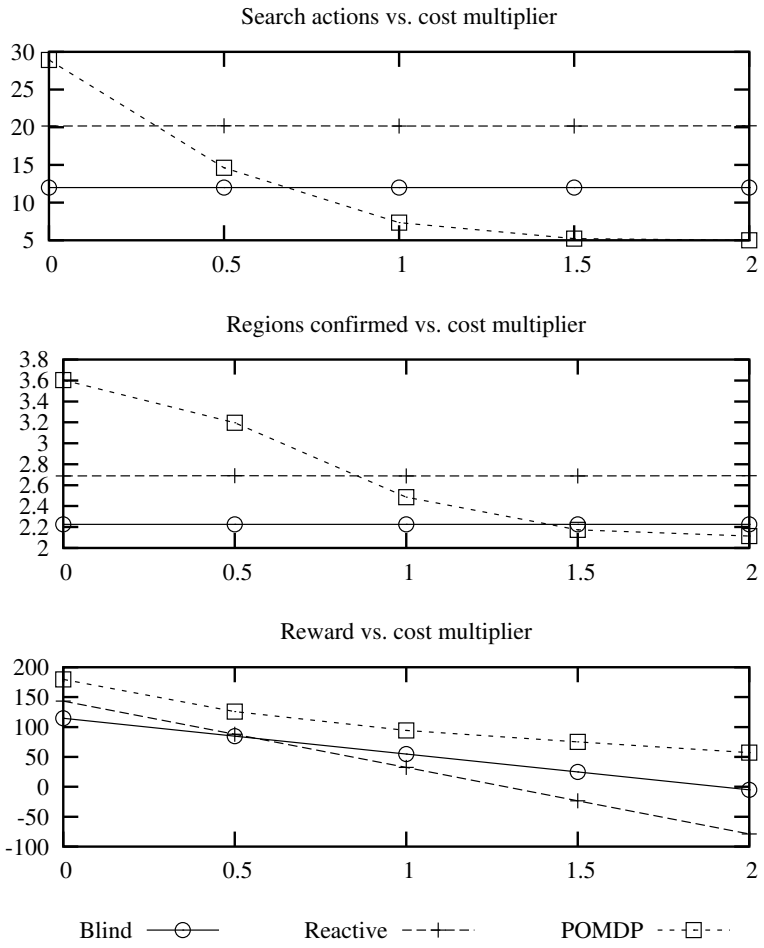


Figure 8.12: *LifeSurvey* simulation testing: planner performance vs. cost multiplier.

provided to the POMDP planner. Reported performance measurements are mean values over all runs. Due to the large number of runs, the radius of the 95% confidence interval for the mean was small, at most 0.08 for any measurement; thus we do not report individual confidence intervals.

Figure 8.12 presents performance variations as a function of the cost multiplier. The performance measures "Search actions", "Regions confirmed", and "Reward" are defined as in Table 8.1.

The blind and reactive policies are the same regardless of c, so their perfor-

mance in the "Search actions" and "Regions confirmed" plots is constant, and their performance in the "Reward" plot is a linear function of c whose slope is determined by the mean cost the policy incurred in the original model. The reactive policy incurs more cost, so its reward decreases more steeply as c increases. The two lines are seen to cross near $c = 0.5$.

In contrast, the POMDP planner automatically adjusts its policy as c increases, using fewer search actions and confirming life in fewer regions as a result. It outperforms both the blind and reactive planners by a wide margin over all values of c. We also tried to understand the detailed behavior of the POMDP planner as c varied:

- $c = 0$: When the cost multiplier is zero, no costs are incurred for any action. In this context, it is easy to see that: (1) on entering a new cell, it is always at least as good to scan as to make a move, since the scan action has no cost and often provides useful information; (2) sampling moves are always at least as good as non-sampling moves, since there is no cost difference and sampling often has some chance of providing reward. In viewing simulation traces from the POMDP planner, we observe that it visits all the regions, scans after about 90% of all moves, and never uses a non-sampling move. The focused value iteration bounds indicate that the value of the optimal policy lies in the interval $[179.5, 184.5]$.

- $c = 0.5$: At this cost level, the POMDP planner uses the scan action about 50% of the time, and about 50% of its moves are sampling moves. It is not always clear why the planner chooses to scan in one location, blindly sample without scanning in another, and simply pass by a third; presumably a number of factors influence the decision, and a full explanation would involve looking at how they balance quantitatively.

  One somewhat surprising behavior is that after receiving a positive scan reading and sampling that cell, the planner sometimes samples a second cell in the same region, apparently just to be sure that at least one of the samples contains detectable life. This is not something that would necessarily occur to a human policy designer. The value of the optimal policy lies in the interval $[124.9, 139.2]$.

- $c = 1$: The POMDP planner uses the scan action about 30% of the time, and uses sampling moves about 20% of the time. Scan actions are now mostly applied to high-value regions with $p = 0.5$. The value of the optimal policy lies in the interval $[94.4, 103.4]$.

- $c = 1.5$: At this high cost level, the POMDP planner's policy is particularly

easy to understand. It uses a low-effort strategy to confirm the presence of life in the three high-value regions with $p = 0.5$, completely ignoring the two low-value regions with $p = 0.1$.

Two of the high-value regions are relatively large. The planner deals with those regions by applying a single scan action from a location where all three forward cells are in the high-value region; it then uses a sampling move if one of the cells gives a positive reading. One of the high-value regions contains only a single cell. The planner deals with this region by blindly sampling the single cell without a preceding scan action. The value of the optimal policy is 75.0.

For the cases $c = 1.5$ and $c = 2$, the POMDP planner was able to exactly solve the problem up to our desired regret bound of $\epsilon = 10^{-3}$ within the allotted time. These versions of the problem appear to be easier because the heuristic search can use its upper bound to avoid evaluating the majority of policies, in particular the longer plans that are too costly because they include many scan actions and detours.

- $c = 2$: The policy is broadly similar to that with $c = 1.5$. The small differences in search actions and regions confirmed are due to (1) following a slightly different path, and (2) different responses to certain observations. (For example, if a scan returns multiple positive readings, the policies may differ in terms of which forward cell they choose to sample, for reasons that are not clear.) The value of the optimal policy is 57.4.

We can also compare the simulation results with $c = 1$ to the onboard results in Table 8.1. We are pleased to see that the performance ordering across all three performance measures is the same, although the actual values differ. There are a number of possible explanations for the differences, most obviously that the onboard testing reports an average result over region maps 1 and 2, whereas the simulation testing focuses on region map 1 only.

These results show that the POMDP planner can successfully adapt its policy to changes in the problem, significantly outperforming the blind and reactive planners across all values of c. We also analyzed some of the interesting strategies that were automatically generated by the POMDP planner.

8.5.4 Simulation Testing: Robustness to Model Error

Future planetary exploration robots are unlikely to have access to accurate probabilistic planning models for novel environments. Especially early on, the probabilities specified by their planning models are likely to be rough *a priori* estimates

from domain experts or learned values from terrestrial environments that may differ significantly from the target environment. As a result, it is important to understand how robust the POMDP planner is to model errors. If it only shows strong performance when it has access to an extremely accurate planning model, it may not be as useful for real exploration applications as our earlier experiments suggest.

In order to study this issue, we ran a series of simulation experiments in which the simulation model was held constant, but the planning model was corrupted. Specifically, in the planning model only, we systematically changed the prior probabilities that cells contained life, multiplying all priors by a constant factor q, called the *prior multiplier*. We tested five settings of q ranging from 0 to 2. With $q = 0$ the planner believes with certainty that there is no life present, with $q = 1$ the planning model has uncorrupted priors, and with $q = 2$ the planning model significantly overestimates the priors.

We chose this type of systematic error because it is easy to inject into the planning model, and because its structure is simple enough that the resulting policy changes should be easy to understand. Of course, a real implementation of an exploration problem like *LifeSurvey* would likely be subject to many other types of error, including random noise in the region priors, unmodeled correlations in the presence of life between neighboring cells, and inaccuracies in the sensor noise model. This experiment only scratches the surface.

We used the POMDP planner to generate five policies, one for each setting of q. As before, the POMDP planner was run on a 3.2 GHz Pentium-4 processor with 2 GB of main memory. We ran the planner for 10^3 seconds (about 20 minutes) to generate each policy. We also generated five distinct policies with the reactive planner, since the path it selects depends on the region priors. The blind planner is unaffected by q, so its policies were the same across all versions of the planning model.

As before, each policy was tested on each setting of q for 10^5 runs in simulation. All simulation runs used the same region map, with a new target layout drawn randomly for each run. The system model used by the simulator was held constant as the planning model varied. Reported performance measurements are mean values over all runs. As before, the errors in the estimated means were small enough (< 0.1) that we do not report individual confidence intervals.

Figure 8.13 presents performance variations as a function of the prior multiplier. The performance measures "Search actions", "Regions confirmed", and "Reward" are defined as in Table 8.1.

The blind policy does not depend on q, and the simulator model did not vary in this experiment, so the blind planner performance is constant across all values of q. One can verify that the values are equal to those in the earlier simulation experiment with $c = 1$.

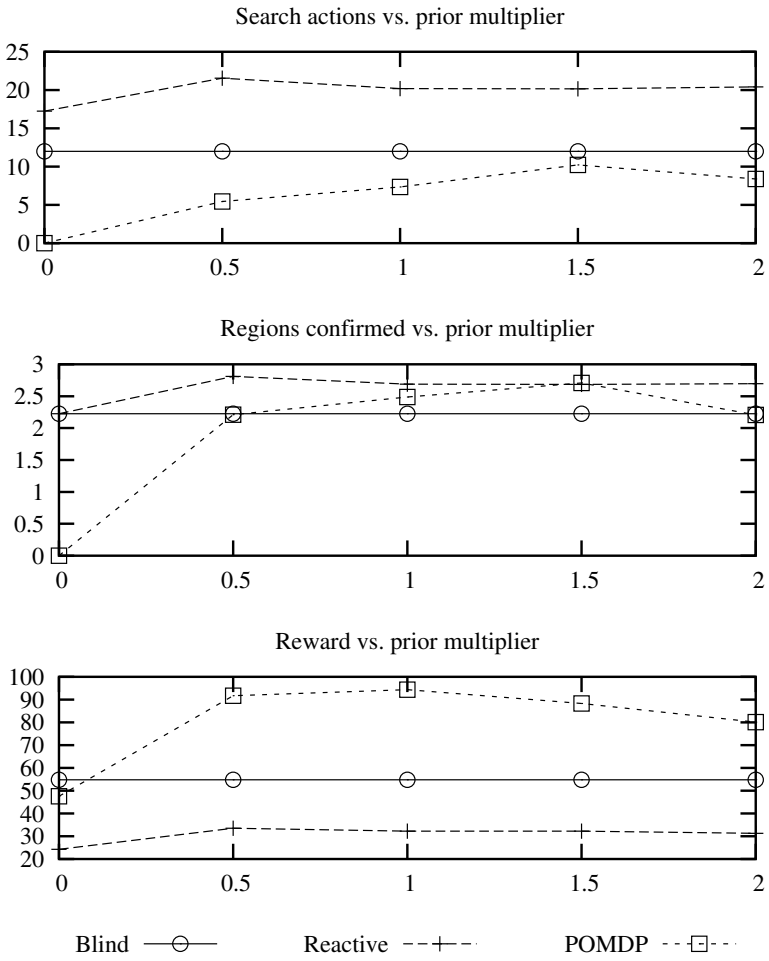


Figure 8.13: *LifeSurvey* simulation testing: planner performance vs. prior multiplier (holding simulation model constant).

Under the reactive planner, the prior multiplier q affects only the initial selection of the reference path, by changing the p_r values used in the path evaluation heuristic (8.6). When $q = 0$, all paths have the same heuristic value, and the reactive planner breaks the tie by arbitrarily selecting a path that follows the northern border of the map, where part of the sensor footprint of the scan action is off the edge of the map. This means the rover effectively scans fewer cells, reducing the number of search actions and hurting overall performance slightly. For other values of q, the path is routed through high-value regions and is fairly consistent, varying

slightly only because the heuristic is not linear in the p_r values.

The policy of the POMDP planner varies most as q changes. With $q = 0$, the planner believes that there is no life in the map and dashes straight to the exit with non-sampling moves. It never performs search actions and never confirms the presence of life in any region, but still gets small rewards for visiting regions and moving through cells containing life.

With $q = 2$, high-value regions that normally have $p = 0.5$ now have $p = 1$ in the planner model. Thus the planner believes that it can blindly sample any cell in such a region and be assured of confirming the presence of life. Low-value regions that normally have $p = 0.1$ now have $p = 0.2$ in the planner model. The POMDP policy aggressively scans these regions, and falls short of its predicted performance because the scans come up negative more frequently than they should according to the planning model.

For the intermediate values $q = 0.5, 1, 1.5$, the POMDP policy varies, but luckily the policies for all three values make liberal use of scanning actions in high-value regions, so the robot receives sensor information that to some extent corrects the corrupted priors. The POMDP planner performance is (unsurprisingly) highest with uncorrupted priors at $q = 1$, but nearly the same performance is achieved at $q = 0.5$ and $q = 1.5$.

These results show that the POMDP planner is fairly robust to one type of systematic model error in a broad range ($q = 0.5$ or 1.5), although performance eventually degrades as the error becomes too large ($q = 0$ or 2). The POMDP planner again significantly outperformed the blind and reactive planners at all values except $q = 0$.

8.6 Conclusions

In the future, highly capable Mars rovers will drive longer distances and will face the communications bottlenecks inherent in over-the-horizon science. As we have shown, autonomous science can help to overcome these difficulties by adaptively planning exploration decisions that best match scientists' goals. Our specific technical contributions included:

1. The first demonstration of a science rover autonomously responding to the detection of life in the field (§8.3).

2. The first demonstration of autonomous mapping that takes both prior maps and rover science observations into account, and the first demonstration of a rover executing a plan built by automated reasoning about potential future science opportunities (§8.4).

The automatic life detection experiment provided us with valuable experience integrating science autonomy technology with onboard software systems and with the command cycle of remote science operations. The efficient mapping experiment showed that POMDP planning can be useful in science autonomy applications. Together, these experiments address different challenges that will be relevant to future field deployment of POMDP planning for science autonomy.

In the context of this thesis, the *LifeSurvey* domain was a capstone challenge in POMDP planning. On the one hand, *LifeSurvey* was motivated by the demands of real future applications, and on the other it was a POMDP problem with interesting structure that was intractable using existing state-of-the-art solution algorithms. In order to generate an approximately optimal policy we used nearly all of the techniques developed in the course of this thesis work, from value function representation to heuristic search to state abstraction.

Chapter 9

Conclusions

The thesis of this work is that probabilistic planning that reasons about information-gathering actions can significantly improve the efficiency of robotic exploration. To support that claim, we developed techniques to speed up probabilistic planning, used these techniques to generate exploration policies, and showed that the resulting policies outperformed manually generated policies both in simulation and onboard a robot.

We developed the unifying framework of focused value iteration for understanding a class of planning algorithms for MDPs and POMDPs. Focused value iteration cleanly separates the key issues of value function representation and heuristic search for selecting points to update. Because techniques in these two areas are analyzed independently, we can adapt search strategies from fully observable MDPs to POMDPs and develop theoretical results that depend only on the value function representation, so that they apply across multiple search strategies.

We presented a number of efficiency improvements for POMDP value function representations, including α vector masking, passive pruning, and hybrid tabular representations. We identified a trade-off one must make in designing an updatable representation—it can provide either strong update generalization or fast updates, but not both. For the problems we studied, performance was best when we compromised these two objectives.

With respect to the commonly used max-planes representation for POMDP value functions, we developed two new bounds on the number of α vectors required to represent the optimal value function V^* within ϵ. For the first bound, we focused on approximating V^* well in the neighborhood of beliefs that are quickly reachable from the initial belief b_0, requiring fewer α vectors in other parts of the belief simplex but still guaranteeing that policies based on the approximation have small regret. For the second bound, we found a way to smooth out the non-linearities of

V^* so that each α vector could effectively cover more volume. Roughly speaking, this approach reduced the number of α vectors needed from n to $\sqrt{n}$.

We developed two heuristic search algorithms, Heuristic Search Value Iteration (HSVI) and Focused Real-Time Dynamic Programming (FRTDP). Both algorithms use lower and upper bounds on V^* to direct forward exploration during search. The guiding principle of HSVI is to choose the successor state that contributes most to the uncertainty in the value of the initial state s_0; this uncertainty is related to the regret of the output policy. FRTDP is based on the same idea, but its choices are less myopic. Compared to related previous algorithms, both HSVI and FRTDP showed significant performance improvements on benchmark problems.

In order to handle large factored POMDPs, we developed a state space compression algorithm called conditionally irrelevant variable abstraction (CIVA). CIVA temporarily abstracts away state variables in contexts where they are provably irrelevant. With appropriate problem structure, CIVA can exponentially reduce the size of the unfactored state space. In one instance, we used CIVA to reduce the unfactored state space from more than 10^{24} states to less than 10^4, putting it within reach of our focused value iteration solution algorithms.

We ran two main experiments in robotic exploration. The first experiment, conducted in the Atacama Desert of Chile, demonstrated that enabling a science rover to autonomously react to signs of life can significantly improve its exploration efficiency. The second experiment, conducted in a controlled outdoor environment, showed that by using advanced POMDP planning techniques we can tractably generate exploration policies that outperform simple manually generated policies.

9.1 Software Contributions

Most of the algorithms and POMDP and MDP models used in this work are freely available as part of the ZMDP software package, which can be downloaded from my web page at `http://www.cs.cmu.edu/~trey/zmdp/`.

ZMDP is written in C++ and runs on the Linux and Mac OS X platforms. It reads POMDP and MDP models specified in Tony Cassandra's model format, providing a number of options for solving problems and plotting anytime algorithm performance. It has been used to teach about POMDP solution algorithms in one class (CMU 16-830: Planning, Execution, and Learning), and has been downloaded more than 200 times. Please do not hesitate to try it out, and contact me if you have any problems.

Algorithm 9.1 Continuous focused value iteration (variant of Algorithm 3.3).

```
1:  function continuousFocusedValueIteration(δ) :
2:     [executes a policy π such that regret(π) ≤ δ]
3:     x ← <SEARCH>.initialSearchState(s_0)
4:     V^L ← <LB>.initialValueFunction()
5:     V^U ← <UB>.initialValueFunction()
6:     loop:
7:        if (V^U(s_0) − V^L(s_0)) ≤ δ :
8:           apply action chooseAction(V^L, s_0)
9:           s_0 ← {updated state based on action and observed outcome}
10:          x ← <SEARCH>.initialSearchState(s_0)
11:       s, x ← <SEARCH>.chooseUpdatePoint(V^L, V^U, x)
12:       V^L ← <LB>.focusedUpdate(V^L, s)
13:       V^U ← <UB>.focusedUpdate(V^U, s)
14:
15: function chooseAction(V^L, s) :
16:    return arg max_a [R(s, a) + γ Σ_s' T(s, a, s')V^L(s')]
```

9.2 Future Work

There are many opportunities to extend the work presented in this thesis document.

9.2.1 Continuous Planning

In many real-world planning domains, human planners deal with complexity by forming an incomplete or sub-optimal initial policy and elaborating it during execution as new information becomes available. In the automated planning community, this is sometimes called continuous planning (Myers, 1999) or interleaving planning and execution (Nourbakhsh, 1997).

It is particularly easy to adapt focused value iteration to a continuous planning context. With the heuristics we developed, focused value iteration tends to focus planning effort on likely outcomes, so if the planner is stopped early it generates policies that perform well in the nominal case but poorly when there are surprises at execution time. This problem can be partially remedied by allowing the agent to stop and perform further planning when an off-nominal outcome occurs. Furthermore, if the agent is in state s and must decide whether to act immediately or plan further, it can base its decision on the focused value iteration regret bound $V^U(s) - V^L(s)$.

Continuous focused value iteration, Algorithm 9.1, is just one proposed adaptation for continuous planning. In this approach, a single set of value function bounds is maintained throughout execution. At each opportunity to take an action, the algorithm updates the "initial state" of the planner to be the current execution state and performs focused updates until the regret of its output policy (starting from the current state) is bounded to less than δ. Since the algorithm starts executing a policy whose regret is less than δ and incrementally improves the policy during execution, it is likely to actually achieve significantly smaller regret through improved responses to off-nominal outcomes. Of course, many other approaches are possible depending on the relative importance of policy quality and planning time constraints.

9.2.2 Better Understanding of MDP Heuristic Search

In deterministic planning problems there is a well-developed theory of optimal heuristic search. For example, under appropriate conditions, the A^* algorithm is known to be optimal in the sense that any algorithm that is guaranteed to generate an optimal policy must examine all the states that A^* examines (Hart et al., 1968; Pearl, 1984).

MDP heuristic search is complicated both by uncertain action outcomes and by the fact that we are often interested in generating a policy that is approximately rather than exactly optimal (Hansen and Zilberstein, 2001; Bonet and Geffner, 2003a; McMahan et al., 2005). Under these constraints, we probably cannot obtain any heuristic search optimality result as strong as that of A^* for deterministic problems. Indeed, little is known about how far MDP heuristic search can be improved.

One triumph of the A^* theoretical analysis was the clean separation between the generic A^* search algorithm and the problem-specific heuristic information. In contrast, our sense is that state-of-the-art generic MDP search algorithms still hard-code decisions that would more properly be based on problem-specific information. In order to move intelligence from the algorithm to the heuristic, we will likely need richer ways to specify "admissible" heuristics, going beyond (for example) simple upper or lower bound values on states. This is a potentially rich area for future work.

9.2.3 Integrating POMDP Planning With Rover Operations

We have demonstrated that our POMDP planning techniques can generate good policies for simple exploration problems, but a number of issues need to be addressed before our technology can be applied to a full-up rover mission. Here are a few:

- Planetary rovers typically face complex operational constraints (Tompkins et al., 2004; Bresina et al., 2005). For example, science instruments may require long warm-up periods before use, or the rover may need to end its daily operations on a south-facing slope. There are a number of approaches for enforcing such constraints while using a POMDP planner. It may be feasible to compile simple constraints into the planning model itself, although more complex constraints are likely to make planning intractable.

  A potentially more promising approach is to rely on different types of planning at different levels of abstraction. The overall plan for a command cycle could consist of multiple segments, with some segments using POMDP planning and some segments using other approaches. At finer levels of detail, actions selected by the POMDP planner could be macros that invoke a lower-level planner.

- There are a number of usability issues that have not been addressed by the POMDP community. For example, in real applications, it is often easier to elicit user preferences with a mixed-initiative system that allows users to interact with the planner (Bresina et al., 2005). To our knowledge, so far there has been no research on human-computer interaction to control POMDP planning.[1]

- In order to generate good policies, we need good probabilistic models. Unfortunately, even in well-understood domains, domain experts often do a poor job of estimating the probabilities of events. This problem is only compounded for planetary rovers exploring novel environments that are poorly understood. Better techniques are needed for model learning and robust planning in the presence of model errors (Singh et al., 2003; Jaulmes et al., 2005).

9.3 Summary

We have presented several techniques for improving the scalability of MDP and POMDP planning, in areas ranging from value function representation to heuristic search to state abstraction. We used these techniques to solve extremely challenging POMDPs related to robotic exploration, generating approximately optimal policies with strong regret bounds. While much research remains to be done, our

[1] Here we are drawing a distinction between human-computer interaction to control a POMDP planner and use of a POMDP planner to manage human interaction, which has been studied for example in dialog management and preference elicitation systems (Roy et al., 2000; Boutilier, 2002).

experiments with a rover in the Atacama Desert of Chile and in controlled outdoor environments provide early validation both for the overall concept of science autonomy and for POMDP exploration planning in particular.

Bibliography

Aberdeen, D. (2003). *Policy-Gradient Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, Australia National University.

Abrams, M. (2000). ASTER: Data products for the high spatial resolution imager on NASA's EOS-AM1 platform. *Int. J. of Remote Sensing*, 21(5):847–859.

Andre, D. and Russell, S. (2002). State abstraction for programmable reinforcement learning agents. In *Proc. Nat. Conf. on Artificial Intelligence (AAAI)*, pages 119–125.

Astrom, K. J. (1965). Optimal control of Markov decision processes with incomplete state estimation. *J. Math. Analysis and Applications*, 10:174–205.

Baird, L. C. and Moore, A. W. (1999). Gradient descent for general reinforcement learning. In *Proc. Advances in Neural Information Processing Systems (NIPS)*.

Barto, A., Bradtke, S., and Singh, S. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1-2):81–138.

Baum, J. and Nicholson, A. E. (1998). Dynamic non-uniform abstractions for approximate planning in large structured domains. In *Proc. Pacific Rim Int. Conf. on Artificial Intelligence*.

Baxter, J. and Bartlett, P. L. (2000). Reinforcement learning on POMDPs via direct gradient ascent. In *Proc. Int. Conf. on Machine Learning (ICML)*.

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

Bernstein, D. S., Givan, R., Immerman, N., and Zilberstein, S. (2002). The complexity of decentralized control of Markov decision processes. *Math. of Operations Research*, 27(4):819–840.

Bernstein, D. S., Hansen, E. A., and Zilberstein, S. (2005). Bounded policy iteration for decentralized POMDPs. In *Proc. Int. Joint Conf. on Artificial Intelligence*.

Bertsekas, D. P. and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA.

Bonet, B. and Geffner, H. (2003a). Faster heuristic search algorithms for planning with uncertainty and full feedback. In *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI)*.

Bonet, B. and Geffner, H. (2003b). Labeled RTDP: Improving the convergence of real time dynamic programming. In *Proc. Int. Conf. on Applied Planning and Scheduling (ICAPS)*.

Bornstein, B. and Castaño, R. (2005). Creation and testing of an artificial neural network based carbonate detector for Mars rovers. In *Proc. IEEE Aerospace Conf.*

Boutilier, C. (2002). A POMDP formulation of preference elicitation problems. In *Proc. Nat. Conf. on Artificial Intelligence (AAAI)*, pages 239–246, Edmonton.

Boutilier, C. and Dearden, R. (1994). Using abstractions for decision-theoretic planning with time constraints. In *Proc. Nat. Conf. on Artificial Intelligence (AAAI)*.

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

Boutilier, C. and Poole, D. (1996). Computing optimal policies for partially observable decision processes using compact representations. In *Proc. Nat. Conf. on Artificial Intelligence*, pages 1168–1175.

Brafman, R. I. (1997). A heuristic variable grid solution method for POMDPs. In *Proc. Nat. Conf. on Artificial Intelligence (AAAI)*.

Brafman, R. I. and Shani, G. (2004). Resolving perceptual aliasing in the presence of noisy senors. In *Proc. Advances in Neural Information Processing Systems (NIPS)*.

Bresina, J. L., Jónsson, A. K., Morris, P. H., and Rajan, K. (2005). Mixed-initiative planning in MAPGEN: Capabilities and shortcomings. In *Proc. Int. Conf. on Planning and Scheduling (ICAPS) Workshop on Mixed-Initiative Planning and Scheduling*.

Cabrol, N. A., Wettergreen, D. S., Warren-Rhodes, K., Grin, E. A., Moersch, J. E., Chong Diaz, G., Cockell, C. S., Coppin, P., Demergasso, C., Dohm, J. M., Ernst, L. A., Fisher, G., Hardgrove, C., Hock, A. N., Marinangeli, L., Minkley, E. G., Ori, G. G., Piatek, J. L., Waggoner, A. S., Weinstein, S. J., Wyatt, M., Smith, T., Thompson, D. R., Wagner, M. D., Jonak, D., Stubbs, K., Thomas, G., Pudenz, E., and Glasgow, J. (2007). Life in the Atacama: Searching for life with rovers (science overview). *J. Geophys. Res. Biogeosciences*. (In press).

Casper, J. and Murphy, R. R. (2003). Human-robot interactions during the robot-assisted urban search and rescue response at the World Trade Center. *IEEE Transactions on Systems, Man and Cybernetics Part B*, 33(3):367–385.

Cassandra, A. R. (1998a). *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, Brown Univ., Dept. of Computer Science.

Cassandra, A. R. (1998b). A survey of POMDP applications. In *Proc. Amer. Ass. for Artificial Intelligence (AAAI) Fall Symp.: Planning with Partially Observable Markov Processes*.

Cassandra, A. R., Kaelbling, L. P., and Kurien, J. A. (1996). Acting under uncertainty: Discrete bayesian models for mobile-robot navigation. In *Proc. IEEE Int. Conf. on Intelligent Robots and Systems (IROS)*.

Cassandra, A. R., Kaelbling, L. P., and Littman, M. L. (1994). Acting optimally in partially observable stochastic domains. In *Proc. Nat. Conf. on Artificial Intelligence (AAAI)*, volume 2, pages 1023–1028, Seattle, WA.

Cassandra, A. R., Littman, M. L., and Zhang, N. L. (1997). Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In *Proc. Int. Conf. on Uncertainty in Artificial Intelligence (UAI)*.

Castaño, A., Anderson, R. C., Castaño, R., Estlin, T., and Judd, M. (2004). Intensity-based rock detection for acquiring onboard rover science. In *Proc. Lunar and Planetary Science Conf. (LPSC)*.

Castaño, R., Anderson, R. C., Estlin, T., DeCoste, D., Fisher, F., Gaines, D., Mazzoni, D., and Judd, M. (2003a). Rover traverse science for increased mission science return. In *Proc. IEEE Aerospace*, Big Sky, Montana.

Castaño, R., Anderson, R. C., Estlin, T., DeCoste, D., Fisher, F., Gaines, D., Mazzoni, D., and Judd, M. (2003b). Rover traverse science for increased mission science return. In *Proc. IEEE Aerospace Conf.*

Charlin, L., Poupart, P., and Shioda, R. (2006). Automated hierarchy discovery for planning in partially observable environments. In *Proc. Advances in Neural Information Processing Systems (NIPS)*.

Cheng, H.-T. (1988). *Algorithms for partially observable Markov decision processes*. PhD thesis, University of British Columbia, Vancouver, BC, Canada.

Chien, S., Sherwood, R., Tran, D., Castaño, R., Cichy, B., Davies, A., Rabideau, G., Tang, N., Burl, M., Mandl, D., Frye, S., Hingemihle, J., D'Augustino, J., Bote, R., Trout, B., Schulman, S., Ungar, S., Gaasback, J. V., Boyer, D., Griffin, M., Burke, H.-H., Greeley, R., Doggett, T., Williams, K., Baker, V., and Dohm, J. M. (2003). Autonomous science on the EO-1 mission. In *Proc. Int. Symp. on Artificial Intelligence, Robotics, and Automation in Space (iSAIRAS)*, Nara, Japan.

Chien, S., Sherwood, R., Tran, D., Cichy, B., Rabideau, G., Castaño, R., Davies, A., Mandl, D., Frye, S., Trout, B., Shulman, S., and Boyer, D. (2005). Using autonomy flight software to improve science return on Earth Observing One. *J. Aerospace Computing, Communication and Information*, 2:196.

Chrisman, L. (1992). Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proc. Nat. Conf. on Artificial Intelligence (AAAI)*.

Conway, J. H. and Sloane, N. J. A. (1999). *Sphere Packings, Lattices, and Groups*. Springer, third edition.

Dearden, R., Meuleau, N., Ramakrishnan, S., Smith, D. E., and Washington, R. (2003). Incremental contingency planning. In *Proc. Int. Conf. on Applied Planning and Scheduling (ICAPS) Workshop on Planning under Uncertainty*.

Dial, G., Bowen, H., Gerlach, F., Grodecki, J., and Oleszczuk, R. (2003). IKONOS satellite, imagery, and products. *Remote Sensing of Environment*, 88(1):23–36.

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

Dongarra, J. J., Croz, J. D., Hammarling, S., and Hanson, R. J. (1988). An extended set of FORTRAN basic linear algebra subprograms. *ACM Trans. Math. Soft.*, 14:1–17.

Drake, A. W. (1962). *Observations of a Markov process through a noisy channel*. PhD thesis, Massachusetts Inst. of Tech.

Draper, D., Hanks, S., and Weld, D. (1994). Probabilistic planning with information gathering and contingent execution. In *Proc. Int. Conf. on Artificial Intelligence Planning Systems*, pages 31–36.

Dunlop, H. (2006). Automatic rock detection and classification in natural scenes. Technical Report CMU-RI-TR-06-40, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Estlin, T., Castaño, R., Anderson, R. C., Gaines, D., Fisher, F., and Judd, M. (2003). Learning and planning for mars rover science. In *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI) Workshop on Issues in Designing Physical Agents for Dynamic Real-Time Environments: World Modeling, Planning, Learning, and Communicating*, Acapulco, Mexico.

Estlin, T., Fisher, F., Gaines, D., Chouinard, C., Schaffer, S., and Nesnas, I. (2002). Continuous planning and execution for an autonomous rover. In *Proc. Int. NASA Workshop on Planning and Scheduling for Space*, Houston, TX.

Feng, Z. and Hansen, E. A. (2001). Approximate planning for factored POMDPs. In *European Conf. on Planning*.

Feng, Z. and Hansen, E. A. (2004). An approach to state aggregation for POMDPs. In *Proc. Nat. Conf. on Artificial Intelligence (AAAI) Workshop on Learning and Planing in Markov Processes*, San Jose, CA.

Ferguson, D., Stentz, A., and Thrun, S. (2004). PAO* for planning with hidden state. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*.

Fine, S., Singer, Y., and Tishby, N. (1998). The hierarchical hidden Markov model: Analysis and applications. *Machine Learning*, 32:189–208.

Fox, J., Castaño, R., and Anderson, R. C. (2002). Onboard autonomous rock shape analysis for Mars rovers. In *Proc. IEEE Aerospace Conf.*

Geffner, H. and Bonet, B. (1998). Solving large POMDPs by real time dynamic programming. In *Proc. Amer. Ass. for Artificial Intelligency (AAAI) Fall Symposium on POMDPs*.

Givan, R., Dean, T., and Greig, M. (2003). Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*, 147:163–223.

Gor, V., Castaño, R., Manduchi, R., Anderson, R. C., and Mjolsness, E. (2000). Autonomous rock detection for Mars terrain. In *Proc. Amer. Inst. of Aeronautics and Astronautics (AIAA) Space Conf.*

Granas, A. and Dugundji, J. (2003). *Fixed Point Theory*. Springer-Verlag, New York.

Gruber, P. (1993). Asymptotic estimates for smooth and stepwise approximations of convex bodies I. *Forum Mathematicum*, 5:281–297.

Guestrin, C., Koller, D., and Parr, R. (2001). Multiagent planning with factored MDPs. In *Proc. Advances in Neural Information Processing Systems (NIPS)*.

Gulick, V. C., Morris, R. L., Ruzon, M. A., and Roush, T. L. (2001). Autonomous image analyses during the 1999 Marsokhod rover field test. *J. Geophys. Res.*, 106(E4):7745.

Hansen, E. A. (1998). Solving POMDPs by searching in policy space. In *Proc. Int. Conf. on Uncertainty in Artificial Intelligence (UAI)*, Madison, Wisconsin.

Hansen, E. A., Bernstein, D. S., and Zilberstein, S. (2004). Dynamic programming for partially observable stochastic games. In *Proc. Nat. Conf. on Artificial Intelligence (AAAI)*, San Jose, CA.

Hansen, E. A. and Feng, Z. (2000). Dynamic programming for POMDPs using a factored state representation. In *Proc. Int. Conf. on Applied Planning and Scheduling (ICAPS)*.

Hansen, E. A. and Zhou, R. (2003). Synthesis of hierarchical finite-state controllers for POMDPs. In *Proc. Int. Conf. on Automated Planning and Scheduling (ICAPS)*, Trento, Italy.

Hansen, E. A. and Zilberstein, S. (2001). LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129:35–62.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths in graphs. *IEEE Trans. on Systems Science and Cybernetics*, SSC-4(2):100–107.

Hauskrecht, M. (1997). Incremental methods for computing bounds in partially observable Markov decision processes. In *Proc. Nat. Conf. on Artificial Intelligence (AAAI)*, pages 734–739, Providence, RI.

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

Hauskrecht, M. and Fraser, H. (2000). Planning treatment of ischemic heart disease with partially observable Markov decision processes. *Artificial Intelligence in Medicine*, 18:221–244.

Hoey, J., von Bertoldi, A., Poupart, P., and Mihailidis, A. (2007). Assisting persons with dementia during handwashing using a partially observable Markov decision process. In *Proc. Int. Conf. on Vision Systems (ICVS)*.

Hsiao, K., Kaelbling, L. P., and Lozano-Perez, T. (2007). Grasping POMDPs. In *Proc. Int. Conf. on Robotics and Automation (ICRA)*.

Izadi, M. and Precup, D. (2006). Exploration in POMDP belief space and its impact on value function approximation. In *Proc. European Conf. on Artificial Intelligence (ECAI)*.

Jaulmes, R., Pineau, J., and Precup, D. (2005). Active learning in partially observable Markov decision processes. In *Proc. European Conf. on Machine Learning*.

Kaelbling, L. P. (1993). *Learning in Embedded Systems*. The MIT Press.

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

Kakade, S. (2001). A natural policy gradient. In *Proc. Advances in Neural Information Processing Systems (NIPS)*.

Kearns, M., Mansour, Y., and Ng, A. Y. (2000). Approximate planning in large POMDPs via reusable trajectories. In *Proc. Advances in Neural Information Processing Systems (NIPS)*.

Korf, R. E. (1990). Real-time heuristic search. *Artificial Intelligence*, 42(189–211).

Leonard, N., Paley, D., Lekien, F., Sepulchre, R., Fratantoni, D. M., and Davis, R. (2007). Collective motion, sensor networks, and ocean sampling. *Proc. IEEE*, 95(1):48–74.

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

Littman, M. L., Cassandra, A. R., and Kaelbling, L. P. (1995). Learning policies for partially observable environments: scaling up. In *Proc. Int. Conf. on Machine Learning (ICML)*.

Littman, M. L., Cassandra, A. R., and Kaelbling, L. P. (1996). Efficient dynamic-programming updates in partially observable Markov decision processes. Technical Report CS-95-19, Brown Univ., Providence, RI.

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

Loch, J. and Singh, S. (1998). Using eligibility traces to find the best memoryless policy in partially observable Markov decision processes. In *Proc. Int. Conf. on Machine Learning (ICML)*, pages 323–331.

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

Maimone, M. W., Biesadecki, J., Tunstel, E., Cheng, Y., and Leger, P. C. (2006). Surface navigation and mobility intelligence on the Mars Exploration Rovers. In *Intelligence for Space Robotics*, pages 45–69. TSI Press, San Antonio, TX.

McAllester, D. and Singh, S. (1999). Approximate planning for factored POMDPs using belief state simplification. In *Proc. Int. Conf. on Uncertainty in Artificial Intelligence (UAI)*.

McCallum, A. R. (1995a). Instance-based utile distinctions for reinforcement learning. In *Proc. Int. Conf. on Machine Learning (ICML)*.

McCallum, A. R. (1995b). *Reinforcement learning with selective perception and hidden state*. PhD thesis, Univ. of Rochester.

McMahan, H. B., Likhachev, M., and Gordon, G. J. (2005). Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *Proc. Int. Conf. on Machine Learning (ICML)*.

Meuleau, N., Kim, K.-E., and Kaelbling, L. P. (1999). Solving POMDPs by searching the space of finite policies. In *Proc. Int. Conf. on Uncertainty in Artificial Intelligence (UAI)*.

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

Moorehead, S. (2001). *Autonomous Surface Exploration for Mobile Robots*. PhD thesis, Robotics Institute, Carnegie Mellon University. CMU-RI-TR-01-30.

Munos, R. (2004). Error bounds for approximate value iteration. Technical Report CMAP 527, École Polytechnique.

Myers, K. L. (1999). CPEF: A continuous planning and execution framework. *AI Magazine*, 20(4):63–69.

Ng, A. Y. and Jordan, M. (2000). PEGASUS: A policy search method for large MDPs and POMDPs. In *Proc. Int. Conf. on Uncertainty in Artificial Intelligence (UAI)*.

Nourbakhsh, I. (1997). *Interleaving Planning and Execution for Autonomous Robots*. Kluwer Academic Publishers.

Nourbakhsh, I., Powers, R., and Birchfield, S. (1995). DERVISH: An office-navigating robot. *AI Magazine*, 16(2):53–60.

Papadimitriou, C. H. and Tsitsiklis, J. N. (1987). The complexity of Markov decision processes. *Math. of Operations Research*, 12(3):441–450.

Parr, R. and Russell, S. (1995). Approximating optimal policies for partially obsevable stochastic domains. In *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI)*.

Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

Pearlman, J., Carman, S., Segal, C., Jarecke, P., Clancy, P., and Browne, W. (2001). Overview of the Hyperion Imaging Spectrometer for the NASA EO-1mission. *Proc. Int. Geoscience and Remote Sensing Symp. (IGARSS)*, 7.

Pedersen, L. (2000). *Robotic Rock Classification and Autonomous Exploration*. PhD thesis, Robotics Institute, Carnegie Mellon University. CMU-RI-TR-01-14.

Pedersen, L. (2001). Autonomous characterization of unknown environments. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 277–284.

Pedersen, L. (2002). Science target assessment for Mars rover instrument deployment. In *Proc. IEEE Int. Conf. on Intelligent Robotics and Systems (IROS)*, pages 817–822.

Pedersen, L., Clancey, W. J., Sierhuis, M., Muscettola, N., Smith, D. E., Lees, D., Rajan, K., Ramakrishnan, S., Tompkins, P., Vera, A., and Dayton, T. (2006). Field demonstration of surface human-robotic exploration activity. In *Proc. Amer. Ass. for Artificial Intelligence (AAAI) Spring Symp.: To Boldly Go Where No Human-Robot Team Has Gone Before*.

Pedersen, L., Wagner, M. D., Apostolopoulos, D., and Whittaker, W. L. (2001). Autonomous robotic meteorite identification in Antarctica. In *Proc. Int. Conf. on Robotics and Automation (ICRA)*, pages 4158–4165.

Pineau, J. (2004). *Tractable Planning Under Uncertainty: Exploiting Structure*. PhD thesis, Robotics Institute, Carnegie Mellon University.

Pineau, J. and Gordon, G. J. (2005). POMDP planning for robust robot control. In *Proc. Int. Symp. on Robotics Res. (ISRR)*.

Pineau, J., Gordon, G. J., and Thrun, S. (2003a). Applying metric-trees to belief-point POMDPs. In *Proc. Advances in Neural Information Processing Systems (NIPS)*.

Pineau, J., Gordon, G. J., and Thrun, S. (2003b). Point-based value iteration: An anytime algorithm for POMDPs. In *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI)*, Acapulco, Mexico.

Pineau, J., Gordon, G. J., and Thrun, S. (2003c). Policy-contingent abstraction for robust robot control. In *Proc. Int. Conf. on Uncertainty in Artificial Intelligence (UAI)*.

Pineau, J., Gordon, G. J., and Thrun, S. (2006). Anytime point-based approximations for large POMDPs. *J. Artificial Intelligence Res.*, 27(335–380).

Porta, J. M., Vlassis, N., Spaan, M. T. J., and Poupart, P. (2006). Point-based value iteration for continuous POMDPs. *J. Machine Learning Res.*, 7:2329–2367.

Poupart, P. and Boutilier, C. (2003a). Bounded finite state controllers. In *Proc. Advances in Neural Information Processing Systems (NIPS)*.

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

Poupart, P. and Boutilier, C. (2004). VDCBPI: an approximate scalable algorithm for large scale POMDPs. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, Vancouver.

Rote, G. (1992). The convergence rate of the Sandwich algorithm for approximating convex functions. *Computing*, 48:337–361.

Roth, M., Simmons, R., and Veloso, M. (2006). What to communicate? execution-time decision in multi-agent POMDPs. In *Proc. Int. Symp. on Distributed Autonomous Robotic Systems (DARS)*.

Roy, N. and Gordon, G. J. (2003). Exponential family PCA for belief compression in POMDPs. In *Proc. Advances in Neural Information Processing Systems (NIPS)*.

Roy, N., Gordon, G. J., and Thrun, S. (2003). Planning under uncertainty for reliable health care robotics. In *Proc. Int. Conf. on Field and Service Robotics (FSR)*.

Roy, N., Gordon, G. J., and Thrun, S. (2005). Finding approximate POMDP solutions through belief compression. *J. Artificial Intelligence Res.*, 23:1–40.

Roy, N., Pineau, J., and Thrun, S. (2000). Spoken dialog management for robots. In *Proc. Ass. for Computational Linguistics*, Hong Kong.

Roy, N. and Thrun, S. (1999). Coastal navigation with mobile robots. In *Proc. Advances in Neural Information Processing Systems (NIPS)*.

Shani, G., Brafman, R. I., and Shimony, S. E. (2005). Model-based online learning of POMDPs. In *Proc. European Conf. on Machine Learning (ECML)*.

Shani, G., Brafman, R. I., and Shimony, S. E. (2006). Prioritizing point-based POMDP solvers. In *Proc. European Conf. on Machine Learning (ECML)*.

Shani, G., Brafman, R. I., and Shimony, S. E. (2007). Forward search value iteration for POMDPs. In *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI)*.

Simmons, R. and Koenig, S. (1995). Probabilistic robot navigation in partially observable environments. In *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 1080–1087.

Singh, S., Littman, M. L., Jong, N. K., Pardoe, D., and Stone, P. (2003). Learning predictive state representations. In *Proc. Int. Conf. on Machine Learning (ICML)*.

Smith, D. E. (2004). Choosing objectives in oversubscription planning. In *Proc. Int. Conf. on Automated Planning and Scheduling (ICAPS)*.

Smith, T. (2006). ZMDP software for POMDP and MDP planning. http://www.cs.cmu.edu/˜trey/zmdp/.

Smith, T., Niekum, S., Thompson, D. R., and Wettergreen, D. S. (2005). Concepts for science autonomy during robotic traverse and survey. In *Proc. IEEE Aerospace Conf.*

Smith, T. and Simmons, R. (2004). Heuristic search value iteration for POMDPs. In *Proc. Int. Conf. on Uncertainty in Artificial Intelligence (UAI)*.

Smith, T. and Simmons, R. (2005). Point-based POMDP algorithms: Improved analysis and implementation. In *Proc. Int. Conf. on Uncertainty in Artificial Intelligence (UAI)*.

Smith, T. and Simmons, R. (2006). Focused real-time dynamic programming for MDPs: Squeezing more out of a heuristic. In *Proc. Nat. Conf. on Artificial Intelligence (AAAI)*.

Sondik, E. J. (1971). *The optimal control of partially observable Markov processes*. PhD thesis, Stanford University.

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

Spaan, M. T. J. and Vlassis, N. (2005). Perseus: Randomized point-based value iteration for POMDPs. *J. Artificial Intelligence Res.*, 24:195–220.

Spaan, M. T. J., Vlassis, N., and Gordon, G. J. (2006). Decentralized planning under uncertainty for teams of communicating agents. In *Proc. Int. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS)*.

Squyres, S. W., Grotzinger, J. P., Arvidson, R. E., J. F. Bell, I., Calvin, W., Christensen, P. R., Clark, B. C., Crisp, J. A., Farrand, W. H., Herkenhoff, K. E., Johnson, J. R., Klingelhöfer, G., Knoll, A. H., McLennan, S. M., McSween, H. Y., Morris, R. V., Rice, J. W., Rieder, R., and Soderblom, L. A. (2004). In situ evidence for an ancient aqueous environment at Meridiani Planum, Mars. *Science*, 306(5702):1709–1714.

St. Aubin, R., Hoey, J., and Boutilier, C. (2000). APRICODD: Approximate policy construction using decision diagrams. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, pages 1089–1095.

Stentz, A. (1994). Optimal and efficient path planning for partially-known environments. In *Proc. Int. Conf. on Robotics and Automation (ICRA)*.

Tarski, A. (1955). A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math*, 5:285–309.

Theocharous, G. (2002). *Hierarchical Learning and Planning in Partially observable Markov Decision Processes*. PhD thesis, Michigan State University.

Thompson, D. R., Niekum, S., Smith, T., and Wettergreen, D. (2005a). Automatic detection and classification of geological features of interest. In *Proc. IEEE Aerospace Conf.*

Thompson, D. R., Smith, T., and Wettergreen, D. (2005b). Data mining during rover traverse: From images to geologic signatures. In *Proc. Int. Symp. on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS)*.

Thompson, D. R., Smith, T., and Wettergreen, D. (2006). Autonomous detection of novel biologic and geologic features in atacama desert rover imagery. In *Proc. Lunar and Planetary Science Conf. (LPSC)*.

Thrun, S. (2000). Monte carlo POMDPs. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, pages 1064–1070. MIT Press.

Tompkins, P., Stentz, A., and Whittaker, W. L. (2004). Field experiments in mission-level path execution and re-planning. In *Proceedings of the 8th Conference on Intelligent Autonomous Systems (IAS-8)*.

Tompkins, P. D. (2005). *Mission-Directed Path Planning for Planetary Rover Exploration*. PhD thesis, Robotics Institute, Carnegie Mellon University.

Urmson, C. P., Simmons, R. G., and Nesnas, I. (2003). A generic framework for robotic navigation. In *Proc. IEEE Aerospace Conf.*

Virin, Y., Shani, G., Shimony, S. E., and Brafman, R. I. (2007). Scaling up: Solving POMDPs through value based clustering. In *Proc. Nat. Conf. on Artificial Intelligence (AAAI)*.

Waggoner, A. S., Weinstein, S. J., Pane, D., Ernst, L. A., Warren-Rhodes, K., Dohm, J. M., Hock, A. N., Piatek, J. L., Emani, S., Wagner, M. D., Fisher, G., Minkley, E. G., Dansey, L. E., Smith, T., Grin, E. A., Stubbs, K., Thomas, G., Cockell, C. S., Marinangeli, L., Ori, G. G., Heys, S., Teza, J. P., Moersch, J. E., Coppin, P., Chong Diaz, G., Wettergreen, D. S., Cabrol, N. A., and Lanni, F. (2007). Application of pulsed-excitation fluorescence imager for daylight detection of sparse life in tests in the Atacama Desert. *J. Geophys. Res. Biogeosciences*. (In press).

Wagner, M. D., Apostolopoulos, D., Shillcutt, K., Shamah, B., Simmons, R., and Whittaker, W. L. (2001). The science autonomy system of the Nomad robot. In *Proc. Int. Conf. on Robotics and Automation (ICRA)*, pages 1742–1749.

Wang, T., Poupart, P., Bowling, M., and Schuurmans, D. (2006). Compact, convex upper bound iteration for approximate POMDP planning. In *Proc. Nat. Conf. on Artificial Intelligence (AAAI)*, Boston, MA.

Weisstein, E. (1999). Hypersphere. *Mathworld–A Wolfram Web Resource*.

Wettergreen, D. S., Cabrol, N. A., Baskaran, V., Calderón, F., Heys, S., Jonak, D., Lüders, R. A., Pane, D., Smith, T., Teza, J. P., Tompkins, P. D., Villa, D., Williams, C., and Wagner, M. D. (2005). Second experiments in the robotic investigation of life in the Atacama Desert of Chile. In *Proc. Int. Symp. on Artificial Intelligence, Robotics, and Automation in Space (iSAIRAS)*.

White, C. C. (1991). Partially observed Markov decision processes: A survey. *Annals of Operations Research*, 32.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256.

Williams, R. J. and Baird, L. C. (1993). Tight performance bounds on policies based on imperfect value functions. Technical report, Northeastern University, College of Computer Science, Boston, MA.

Zhang, N. L. and Zhang, W. (2001). Speeding up the convergence of value iteration in partially observable Markov decision processes. *J. Artificial Intelligence Res.*, 14:29–51.

Zhou, R. and Hansen, E. A. (2001). An improved grid-based approximation algorithm for POMDPs. In *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI)*.

Author Index