Planning and Acting in Partially Observable Stochastic Domains

Leslie Pack Kaelbling* Michael L. Littman† Anthony R. Cassandra

November 1, 1995

Abstract

In this paper, we bring techniques from operations research to bear on the problem of choosing optimal actions in partially observable stochastic domains. We begin by introducing the theory of Markov decision processes (MDPs) and partially observable MDPs (POMDPs). We then outline a novel algorithm for solving POMDPs off line and show how, in some cases, a finite-memory controller can be extracted from the solution to a POMDP. We conclude with a discussion of the complexity of finding exact solutions to POMDPs and of some possibilities for finding approximate solutions.

Consider the problem of a robot navigating in a large office building. The robot can move from hallway intersection to intersection and can make local observations of its world. Its actions are not completely reliable, however. Sometimes, when it intends to move, it stays where it is or goes too far; sometimes, when it intends to turn, it overshoots. It has similar problems with observation. Sometimes a corridor looks like a corner; sometimes a T-junction looks like an L-junction. How can such an error-plagued robot navigate, even given a map of the corridors?

In general, the robot will have to remember something about its history of actions and observations and use this information, together with its knowledge of the underlying dynamics of the world (the map and other information), to maintain an estimate of its location. Many engineering applications follow this approach, using methods like the Kalman filter [10] to maintain a running estimate of the robot's spatial uncertainty, expressed as an ellipsoid or normal distribution in Cartesian space. This approach will not do for our robot, though. Its uncertainty may be discrete: it might be almost certain that it is in the north-east corner of either the fourth or the seventh floors, though it admits a chance that it is on the fifth floor, as well.

Then, given an uncertain estimate of its location, the robot has to decide what actions to take. In some cases, it might be sufficient to ignore its uncertainty and take actions that would be appropriate for the most likely location. In other cases, it might be better for the robot to take actions for the purpose of gathering information, such as searching for a landmark or reading signs on the wall. In general, it will take actions that fulfill both purposes simultaneously.

*This work was supported in part by NSF grants IRI-9453383 and IRI-9312395.

†This work was supported in part by Bellcore.

1 Introduction

In this paper, we bring techniques from operations research to bear on the problem of choosing optimal actions in partially observable stochastic domains. Problems like the one described above can be modeled as *partially observable Markov decision processes* (POMDPs). Of course, we are not interested only in problems of robot navigation. Similar problems come up in factory process control, oil exploration, transportation logistics, and a variety of other complex real-world situations.

This is essentially a *planning* problem: given a complete and correct model of the world dynamics and a reward structure, find an optimal way to behave. In the artificial intelligence (AI) literature, a deterministic version of this problem has been addressed by adding knowledge preconditions to traditional planning systems [19]. Because we are interested in stochastic domains, however, we must depart from the traditional AI planning model. Rather than taking plans to be sequences of actions, which may only rarely execute as expected, we take them to be mappings from states to actions that specify the agent's behavior no matter what may happen. In many cases, we may not want a full policy; methods for developing partial policies and conditional plans for completely observable domains are the subject of much current interest [8, 7, 11]. A weakness of the methods described in this paper is that they require the states of the world to be represented enumeratively, rather than through compositional representations such as Bayes nets or probabilistic operator descriptions. However, this work has served as a substrate for development of more complex and efficient representations [3].

One important facet of the POMDP approach is that there is no distinction drawn between actions taken to change the state of the world and actions taken to gain information. This is important because, in general, every action has both types of effect. Stopping to ask questions may delay the robot's arrival at the goal or spend extra energy; moving forward may give the robot information that it is in a dead-end because of the resulting crash.

Much of the content of this paper is a recapitulation of work in the operations-research literature [15, 18, 25, 27, 31]. We have developed new ways of viewing the problem that are, perhaps, more consistent with the AI perspective. We begin by introducing the theory of Markov decision processes (MDPs) and POMDPs. We then outline a novel algorithm for solving POMDPs off line and show how, in some cases, a finite-memory controller can be extracted from the solution to a POMDP. We conclude with a brief discussion of approximation methods.

2 Markov Decision Processes

Markov decision processes serve as a basis for solving the more complex partially observable problems that we are ultimately interested in. An MDP is a model of an agent interacting synchronously with a world. As shown in Figure 1, the agent takes as input the state of the world and generates as output actions, which themselves affect the state of the world. In the MDP framework, it is assumed that, although there may be a great deal of uncertainty about the effects of an agent's actions, there is never any uncertainty about the agent's current state—it has complete and perfect perceptual abilities.

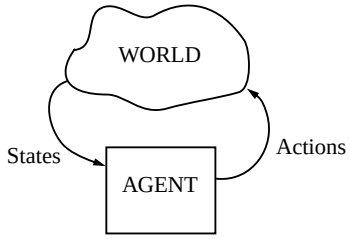

Figure 1: An MDP models the synchronous interaction between agent and world.

Markov decision processes are described in depth in a variety of texts [2, 20]; we will just briefly cover the necessary background.

2.1 Basic Framework

A Markov decision process can be described as a tuple $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$, where

- $\mathcal{S}$ is a finite set of states of the world;
- $\mathcal{A}$ is a finite set of actions;
- $T : \mathcal{S} \times \mathcal{A} \to \Pi(\mathcal{S})$ is the *state-transition function*, giving for each world state and agent action, a probability distribution over world states (we write $T(s, a, s')$ for the probability of ending in state s', given that the agent starts in state s and takes action a); and
- $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, giving the expected immediate reward gained by the agent for taking each action in each state (we write $R(s, a)$ for the expected reward for taking action a in state s).

In this model, the next state and the expected reward depend only on the previous state and the action taken; even if we were to condition on additional previous states, the transition probabilities and the expected rewards would remain the same. This is known as the *Markov* property.

In fact, MDPs can have infinite state and action spaces. The algorithms that we describe in this section apply only to the finite case; however, in the context of POMDPs, we will consider a class of MDPs with uncountably infinite state spaces.

2.2 Acting Optimally

We would like our agents to act in such a way as to maximize some measure of the long-run reward received. One such framework is *finite-horizon* optimality, in which the agent should act in order to maximize the expected sum of reward that it gets on the next k steps; it should maximize

$$E\left[\sum_{t=0}^{k} r_t\right] \quad ,$$

where r_t is the reward received on step t. This model is somewhat inconvenient, because it is rare that an appropriate k will be known exactly. We might prefer to consider an infinite lifetime for the agent. The most straightforward is the *infinite-horizon discounted* model, in which we sum the rewards over the infinite lifetime of the agent, but discount them geometrically using *discount factor* $0 < \gamma < 1$; the agent should act so as to optimize

$$E\left[\sum_{t=0}^{\infty} \gamma^t r_t\right] \quad .$$

In this model, rewards received earlier in its lifetime have more value to the agent; the infinite lifetime is considered, but the discount factor ensures that the sum is finite. This sum is also the expected amount of reward received if a decision to terminate the run is made on each step with probability $1 - \gamma$. The larger the discount factor, the more effect future rewards have on current decision making. In our future discussions of finite-horizon optimality, we will also use a discount factor; when it has value one, it is equivalent to the simple finite horizon case described above.

A policy is a description of the behavior of an agent. We consider two kinds of policies: stationary and non-stationary. A *stationary policy*, $\pi : \mathcal{S} \to \mathcal{A}$, specifies, for each state, an action to be taken. The choice of action depends only on the state and is independent of the time step. A *non-stationary policy* is a sequence of state-action mappings, indexed by time. The policy π_t is to be used to choose the action on the t^{th}-to-last step as a function of the current state, s_t. In the finite-horizon model, there is not necessarily an optimal stationary policy: the way an agent chooses its actions on the last step of its life is generally going to be very different from the way it chooses them when it has a long life ahead of it. In the infinite-horizon discounted model, the agent always has a constant expected amount of time remaining, so there is no reason to change action strategies: there is a stationary optimal policy.

Given a policy, we can evaluate it based on the long-run value that the agent expects to gain from executing it. In the finite-horizon case, let $V_{\pi,t}(s)$ be the expected sum of reward gained from starting in state s and executing non-stationary policy π for t steps. Clearly, $V_{\pi,1}(s) = R(s, \pi_1(s))$; that is, on the last step, the value is just the expected reward for taking the action specified by the final element of the policy. Now, we can define $V_{\pi,t}(s)$ inductively as

$$V_{\pi,t}(s) = R(s, \pi_t(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi_t(s), s') V_{\pi,t-1}(s') \quad .$$

The t-step value of being in state s and executing non-stationary policy π is the immediate reward, $R(s, \pi_t(s))$, plus the discounted expected value of the remaining $t - 1$ steps. To evaluate the future, we must consider all possible resulting states s', the likelihood of their occurrence $T(s, \pi_t(s), s')$, and their $(t-1)$-step value under policy π, $V_{\pi,t-1}(s')$. In the infinite-horizon discounted case, we write $V_\pi(s)$ for the expected discounted sum of future reward for starting in state s and executing policy π. It is recursively defined by

$$V_\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi(s), s') V_\pi(s') \quad .$$

The value function, V_π, for policy π is the unique simultaneous solution of this set of linear equations, one for each state s.

Now we know how to compute a value function, given a policy. Sometimes, we will need to go the opposite way, and compute a *greedy policy* given a value function. It really only makes sense to do this for the infinite-horizon discounted case; to derive a policy for the finite horizon, we would need a whole sequence of value functions. Given any value function V, a greedy policy with respect to that value function, π_V, is defined as

$$\pi_V(s) = \operatorname*{argmax}_a \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V(s') \right] \quad .$$

This is the policy obtained by, at every step, taking the action that maximizes expected immediate reward plus the expected discounted value of the next state, as measured by V.

What is the optimal finite-horizon policy, π^*? The agent's last step is easy: it should maximize its final reward. So

$$\pi_1^*(s) = \operatorname*{argmax}_a R(s,a) \quad .$$

The optimal policy for the t^{th} step, π_t^*, can be defined in terms of the optimal $(t-1)$-step value function $V_{\pi_{t-1}^*, t-1}$ (written for simplicity as V_{t-1}^*):

$$\pi_t^*(s) = \operatorname*{argmax}_a \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V_{t-1}^*(s') \right] \quad ;$$

V_{t-1}^* is derived from π_{t-1}^* and V_{t-2}^*.

In the infinite-horizon discounted case, for any initial state s, we want to execute the policy π that maximizes $V_\pi(s)$. Howard [9] showed that there exists a stationary policy, π^*, that is optimal for every starting state. The value function for this policy, V_{π^*}, also written V^*, is defined by the set of equations

$$V^*(s) = \max_a \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V^*(s') \right] \quad ,$$

which has a unique solution. An optimal policy, π^*, is just a greedy policy with respect to V^*.

Another way to understand the infinite-horizon value function, V^*, is to approach it by using an ever-increasing discounted finite horizon. As the horizon, t, approaches infinity, V_t^* approaches V^*. This only occurs when the discount factor, γ, is less than 1, which tends to wash out the details of exactly what happens at the end of the agent's life.

2.3 Computing an Optimal Policy

There are many methods for finding optimal policies for MDPs. In this section, we explore *value iteration* because it will also serve as the basis for finding policies in the partially observable case.

Value iteration proceeds by computing the sequence V_t of discounted finite-horizon optimal value functions, as shown in Table 1 (the superscript $*$ is omitted, because we shall

$V_1(s) := 0$ for all s
$t := 1$
loop
  $t := t + 1$
  loop for all $s \in \mathcal{S}$ and for all $a \in \mathcal{A}$
    $Q_t^a(s) := R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V_{t-1}(s')$
    $V_t(s) := \max_a Q_t^a(s)$
  end loop
until $|V_t(s) - V_{t-1}(s)| < \epsilon$ for all $s \in \mathcal{S}$

Table 1: The value iteration algorithm for finite state space MDPs.

henceforth only be considering optimal value functions). It makes use of an auxiliary function, $Q_t^a(s)$, which is the t-step value of starting in state s, taking action a, then continuing with the optimal $(t-1)$-step non-stationary policy. The algorithm terminates when the maximum difference between two successive value functions (known as the *Bellman error magnitude*) is less than some ϵ.

It can be shown [29] that there exists a t^*, polynomial in $|\mathcal{S}|$, $|\mathcal{A}|$, the magnitude of the largest value of $R(s,a)$, and $1/(1-\gamma)$, such that the greedy policy with respect to V_{t^*} is equal to the optimal infinite-horizon policy, π^*. Rather than calculating a bound on t^* in advance and running value iteration for that long, we instead use the following result regarding the Bellman error magnitude, due to Williams and Baird [32] in order to terminate with a near-optimal policy.

If $|V_t(s) - V_{t-1}(s)| < \epsilon$ for all s, then the value of the greedy policy with respect to V_t does not differ from V^* by more than $2\epsilon\gamma/(1-\gamma)$ at any state. That is,

$$\max_{s \in \mathcal{S}} |V_{\pi_{V_t}}(s) - V^*(s)| < 2\epsilon \frac{\gamma}{1-\gamma} \quad .$$

It is often the case that $\pi_{V_t} = \pi^*$ long before V_t is near V^*; tighter bounds may be obtained using the *span semi-norm* on the value function [20].

3 Partial Observability

What happens if the agent is no longer able to determine the state it is currently in with complete reliability? A naive approach would be for the agent to map the most recent observation directly into an action without remembering anything from the past. In our hallway navigation example, this amounts to performing the same action in every location that looks the same—hardly a promising approach. Somewhat better results can be obtained by adding randomness to the agent's behavior: a policy can be a mapping from observations to probability distributions over actions [24].

In order to behave truly effectively in a partially observable world, it is necessary to use memory of previous actions and observations to aid in the disambiguation of the states of the world. The POMDP framework provides a systematic method of doing just that.

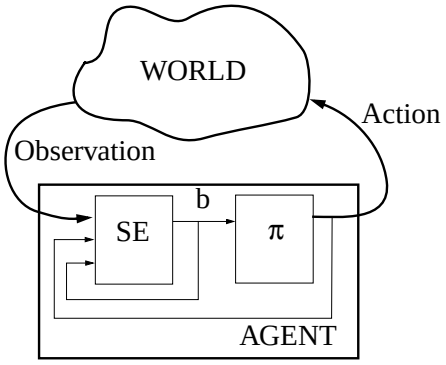

Figure 2: A POMDP agent can be decomposed into a state estimator (SE) and a policy (π).

3.1 POMDP Framework

A partially observable Markov decision process can be described as a tuple $\langle \mathcal{S}, \mathcal{A}, T, R, \Omega, O \rangle$, where

- $\mathcal{S}$, $\mathcal{A}$, T, and R describe a Markov decision process;
- Ω is a finite set of observations the agent can experience of its world; and
- $O : \mathcal{S} \times \mathcal{A} \to \Pi(\Omega)$ is the *observation function*, which gives, for each action and resulting state, a probability distribution over possible observations (we write $O(s', a, o)$ for the probability of making observation o given that the agent took action a and landed in state s').

A POMDP is an MDP in which the agent is unable to observe the current state. Instead, it makes an observation based on the action and resulting state.[1] The agent's goal remains to maximize expected discounted future reward.

3.2 Problem Structure

We decompose the problem of controlling a POMDP into two parts, as shown in Figure 2. The agent makes observations and generates actions. It keeps an internal *belief state*, b, that summarizes its previous experience. The component labeled SE is the *state estimator*: it is responsible for updating the belief state based on the last action, the current observation, and the previous belief state. The component labeled π is the policy: as before, it is responsible for generating actions, but this time as a function of the agent's belief state rather than the state of the world.

What, exactly, is a belief state? One choice might be the most probable state of the world, given the past experience. Although this might be a plausible basis for action in some cases, it is not sufficient in general. In order to act effectively, an agent must take into account its own degree of uncertainty. If it is lost or confused, it might be appropriate

[1] It is possible to formulate an equivalent model in which the observation depends on the previous state instead of, or in addition to, the resulting state, but it complicates the exposition and adds no more expressive power.

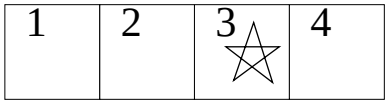

Figure 3: Simple POMDP to illustrate belief state evolution.

for it to take sensing actions such as asking for directions, reading a map, or searching for a landmark. In the POMDP framework, such actions are not explicitly distinguished: their informational properties are described via the observation function.

Our choice for belief states will be probability distributions over states of the world. These distributions encode the agent's subjective probability about the state of the world and provide a basis for acting under uncertainty. Furthermore, they comprise a *sufficient statistic* for the past history and initial belief state of the agent: given the agent's current belief state (properly computed), no additional data about its past actions or observations would supply any further information about the current state of the world.

To illustrate the evolution of a belief state, we will use the simple example depicted in Figure 3. There are four states in this example, one of which is a goal state, indicated by the star. There are two possible observations: one is always made when the agent is in state 1, 2, or 4; the other, when it is in the goal state. There are two possible actions: EAST and WEST. These actions succeed with probability 0.9, and when they fail, the movement is in the opposite direction. If no movement is possible in a particular direction, then the agent remains in the same location.

Assume that the agent is initially equally likely to be in any of the three non-goal states. Thus, its initial belief state is $\begin{bmatrix}0.333 & 0.333 & 0.000 & 0.333\end{bmatrix}$, where the position in the belief vector corresponds to the state number.

If the agent takes action EAST and does not observe the goal, then the new belief state becomes $\begin{bmatrix}0.100 & 0.450 & 0.000 & 0.450\end{bmatrix}$. If it takes action EAST again, and still does not observe the goal, then the probability mass becomes concentrated in the right-most state: $\begin{bmatrix}0.100 & 0.164 & 0.000 & 0.736\end{bmatrix}$. Notice that as long as the agent does not observe the goal state, it will always have some non-zero belief that it is in any of the non-goal states, since the actions have non-zero probability of failing.

3.3 Computing Belief States

A belief state b is a probability distribution over $\mathcal{S}$. We let $b(s)$ denote the probability assigned to world state s by belief state b. The axioms of probability require that $O \leq b(s) \leq 1$ for all $s \in \mathcal{S}$ and that $\sum_{s \in \mathcal{S}} b(s) = 1$. The state estimator must compute a new belief state, b', given an old belief state b, an action a, and an observation o. The new degree

of belief in some state s', $b'(s')$, can be obtained from basic probability theory as follows:

$$\begin{aligned} b'(s') &= \Pr(s'|o,a,b) \\ &= \frac{\Pr(o|s',a,b)\Pr(s'|a,b)}{\Pr(o|a,b)} \\ &= \frac{\Pr(o|s',a)\sum_{s\in\mathcal{S}}\Pr(s'|a,b,s)\Pr(s|a,b)}{\Pr(o|a,b)} \\ &= \frac{O(s',a,o)\sum_{s\in\mathcal{S}}T(s,a,s')b(s)}{\Pr(o|a,b)} \end{aligned}$$

The denominator, $\Pr(o|a,b)$, can be treated as a normalizing factor, independent of s', that causes b' to sum to 1. The state estimation function $SE(b,a,o)$ has as its output the new belief state b'.

Thus, the state-estimation component of a POMDP controller can be constructed quite simply from a given model.

3.4 Finding an Optimal Policy

The policy component of a POMDP agent must map the current belief state into action. Because the belief state is a sufficient statistic, the optimal policy is the solution of a continuous-space "belief MDP." It is defined as follows:

- $\mathcal{B}$, the set of belief states, comprise the state space;
- $\mathcal{A}$, the set of actions, remains the same;
- $\tau(b,a,b')$ is the state-transition function, which is defined as

$$\begin{aligned} \tau(b,a,b') &= \Pr(b'|b,a) \\ &= \sum_{o\in\Omega}\Pr(b'|b,a,o)\Pr(o|b,a) \\ &= \begin{cases} \Pr(o|b,a) & \text{if } SE(b,a,o)=b' \\ 0 & \text{otherwise} \end{cases}; \end{aligned}$$

  and

- $\rho(b,a)$ is the reward function on belief states, constructed from the original reward function on world states:

$$\rho(b,a)=\sum_{s\in\mathcal{S}}b(s)R(s,a) \quad .$$

The reward function may seem strange; the agent is rewarded for merely believing that it is in good states. However, because the state estimator is constructed from a correct observation and transition model of the world, it is not possible for the agent to purposely delude itself into believing that it is in a good state when, in fact, it is not.

This belief MDP is such that an optimal policy for it, coupled with the correct state estimator, will give rise to optimal behavior (in the discounted infinite-horizon sense) for the

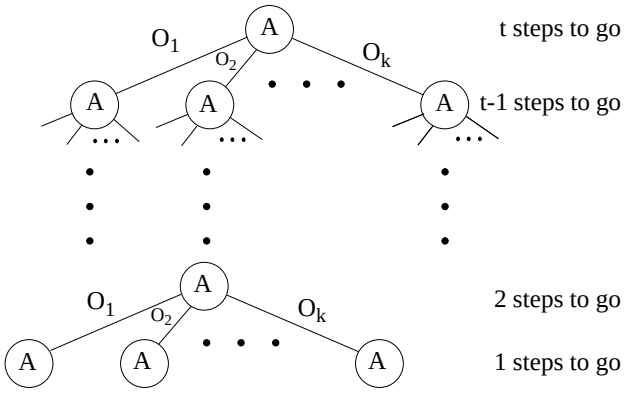

Figure 4: A t-step policy tree.

original POMDP [27, 1]. The remaining problem, then, is to solve this MDP. It is very difficult to solve continuous-space MDPs in the general case, but, as we shall see in the next section, the optimal value function for the belief MDP has special properties that can be exploited to simplify the problem.

4 Value functions for POMDPs

As in the case of discrete MDPs, if we can compute the optimal value function, then we can use it to directly determine the optimal policy. This section concentrates on finding an approximation to the optimal value function. We approach the problem using value iteration to construct, at each iteration, the optimal t-step discounted value function over belief space.

4.1 Policy Trees

When an agent has one step remaining, all it can do is take a single action. With two steps to go, it can take an action, make an observation, then take another action, perhaps depending on the previous observation. In general, an agent's non-stationary t-step policy can be represented by a *policy tree* as shown in Figure 4. It is a tree of depth t that specifies a complete t-step policy. The top node determines the first action to be taken. Then, depending on the resulting observation, an arc is followed to a node on the next level, which determines the next action. This is a complete recipe for t steps of conditional behavior.

Now, what is the expected discounted value to be gained from executing a policy tree p? It depends on the true state of the world when the agent starts. In the simplest case, p is a 1-step policy tree (a single action). The value of executing that action in state s is

$$V_p(s) = R(s, a(p)) \quad ,$$

where $a(p)$ is the action specified in the top node of policy tree p. More generally, if p is a

t-step policy tree, then

$$\begin{aligned} V_p(s) &= R(s,a(p)) + \gamma \quad \text{Expected value of the future} \\ &= R(s,a(p)) + \gamma \sum_{s' \in \mathcal{S}} \Pr(s'|s,a(p)) \sum_{o_i \in \Omega} \Pr(o_i|s',a(p)) V_{o_i(p)}(s') \\ &= R(s,a(p)) + \gamma \sum_{s' \in \mathcal{S}} T(s,a(p),s') \sum_{o_i \in \Omega} O(s',a(p),o_i) V_{o_i(p)}(s') \quad , \end{aligned}$$

where $o_i(p)$ is the $(t-1)$-step policy subtree associated with observation o_i at the top level of a t-step policy tree p. The expected value of the future is computed by first taking an expectation over possible next states, s', then considering the value of each of those states. The value depends on which policy subtree will be executed which, itself, depends on which observation is made. So, we take another expectation, with respect to the possible observations, of the value of executing the associated subtree, $o_i(p)$, starting in state s'.

Because the agent will never know the exact state of the world, it must be able to determine the value of executing a policy tree, p, from some belief state b. This is just an expectation over world states of executing p in each state:

$$V_p(b) = \sum_{s \in \mathcal{S}} b(s) V_p(s) \quad .$$

It will be useful, in the following exposition, to express this more compactly. If we let $\alpha_p = \langle V_p(s_1), \ldots, V_p(s_n) \rangle$, then $V_p(b) = b \cdot \alpha_p$.

Now we have the value of executing the policy tree p in every possible belief state. To construct an optimal t-step policy, however, it will generally be necessary to execute different policy trees from different initial belief states. Let $\mathcal{P}$ be the finite set of all t-step policy trees. Then

$$V_t(b) = \max_{p \in \mathcal{P}} \; b \cdot \alpha_p \quad .$$

That is, the optimal t-step value of starting in belief state b is the value of executing the best policy tree in that belief state.

This definition of the value function leads us to some important geometric insights into its form. Each policy tree, p, induces a value function that is linear in b, and V_t is the upper surface of those functions. So, V_t is piecewise-linear and convex. Figure 5 illustrates this property. Consider a world with only two states. In such a world, a belief state consists of a vector of two non-negative numbers, $\langle b(s_1), b(s_2) \rangle$, that sum to 1. Because of this constraint, a single number is sufficient to describe the belief state. The value function associated with a policy tree p_1, V_{p_1}, is a linear function of $b(s_1)$ and is shown in the figure as a line. The value functions of other policy trees are similarly represented. Finally, V_t is the maximum of all the V_{p_i} at each point in the belief space, giving us the upper surface, which is drawn in the figure with a bold line.

When there are three world states, a belief state is determined by two values (again because of the *simplex constraint*, which requires the individual values to be non-negative and sum to 1). The belief space can be seen as the triangle in two-space with vertices $(0,0)$, $(1,0)$, and $(0,1)$. The value function associated with a single policy tree is a plane in three-space, and the optimal value function is typically a bowl shape that is composed of

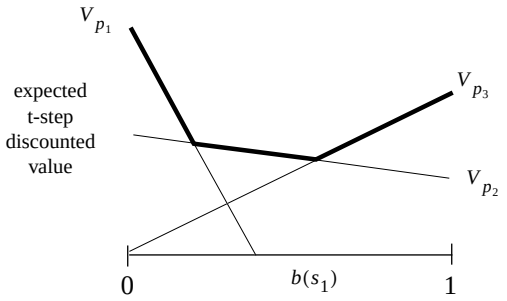

Figure 5: The optimal t-step value function is the upper surface of the value functions associated with all t-step policy trees.

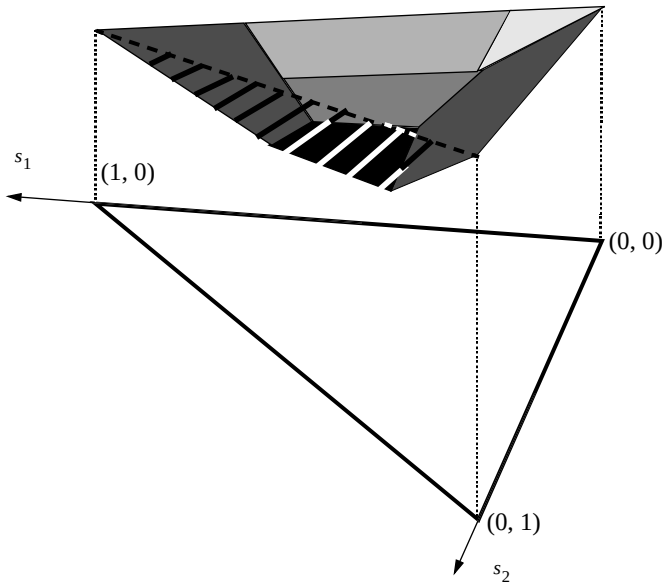

Figure 6: A value function in three dimensions.

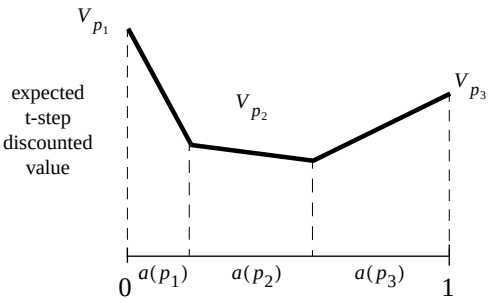

Figure 7: The optimal t-step policy is determined by projecting the optimal value function back down onto the belief space.

planar facets; an example is shown in Figure 6. This general pattern repeats itself in higher dimensions, but becomes difficult to contemplate and even harder to draw!

The convexity of the optimal value function makes intuitive sense when we think about the value of belief states. States that are in the "middle" of the belief space have high entropy—the agent is very uncertain about the real underlying state of the world. In such belief states, the agent cannot select actions very appropriately and so tends to gain less long-term reward. In low-entropy belief states, which are near the corners of the simplex, the agent can take actions more likely to be appropriate for the current state of the world and, so, gain more reward.

Given a piecewise-linear convex value function and the t-step policy trees from which it was derived, it is straightforward to determine the optimal policy for execution on the t^{th} step from the end. The optimal value function can be projected back down onto the belief space, yielding a partition into polyhedral regions. Within each region, there is a single policy tree p such that $b \cdot \alpha_p$ is maximal. The optimal action for each belief state in this region is $a(p)$, the action in the root node of policy tree p. Figure 7 shows the projection of the optimal value function down into a policy partition in the two-dimensional example introduced in Figure 5.

4.2 The Infinite Horizon

In the previous section, we showed that the optimal t-step value function is always piecewise-linear and convex. This is not necessarily true for the infinite-horizon discounted value function; it remains convex [30], but may have infinitely many facets. Still, the optimal infinite-horizon discounted value function can be approximated arbitrarily closely by a finite horizon value function for a sufficiently long horizon [27, 22].

The optimal infinite-horizon discounted value function can be approximated via value iteration, in which the series of t-step discounted value functions is computed; the iteration is stopped when the difference between two successive results is small, yielding an arbitrarily good piecewise-linear and convex approximation to the desired value function. From the approximate value function we can extract a stationary policy that is approximately optimal.

Recall that the value-iteration algorithm has the following form:

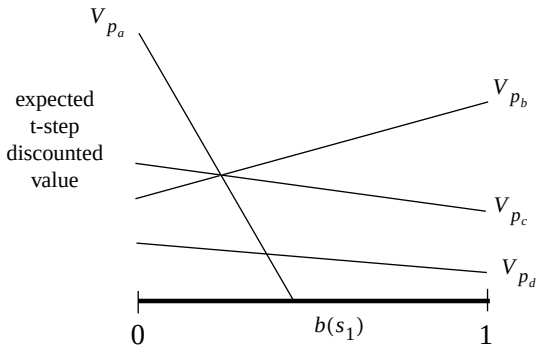

Figure 8: Some policy trees may be totally dominated by others and can be ignored.

loop
    t := t + 1
    compute the optimal t-step discounted value function, V_t
until $\sup_b |V_t(b) - V_{t-1}(b)| < \epsilon$

We have already developed the components of an extremely naive version of this algorithm. On each iteration, we can enumerate all of the t-step policy trees, then compute the maximum of their value functions to get V_t. If the value functions are represented by sets of policy trees, the test for termination can be implemented exactly using linear programming [12]. This is, of course, hopelessly computationally intractable. Each t-step policy tree contains $(|\Omega|^t - 1)/(|\Omega| - 1)$ nodes (the branching factor is $|\Omega|$, the number of possible observations). Each node can be labeled with one of $|\mathcal{A}|$ possible actions, so the total number of t-step policy trees is

$$|\mathcal{A}|^{\frac{|\Omega|^t - 1}{|\Omega| - 1}} \quad ,$$

which grows astronomically in t.

It is possible, in principle, that each of these policy trees might represent the optimal strategy at some point in the belief space and, hence, that each would contribute to the computation of the optimal value function. Luckily, however, this seems rarely to be the case. There are generally many policy trees whose value functions are totally dominated by or tied with value functions associated with other policy trees. Figure 8 shows a situation in which the value function associated with policy p_d is completely dominated by (everywhere less than or equal to) the value function for policy p_b. The situation with the value function for policy p_c is somewhat more complicated; although it is not completely dominated by any single value function, it *is* completely dominated by p_a and p_b taken together.

Given a set of policy trees, $\mathcal{V}$, it is possible to define a unique minimal subset that represents the same value function that $\mathcal{V}$ represents. We will call the elements of this set the *useful* policy trees.

The ability to find the set of useful policy trees serves as a basis for a more efficient version of the value-iteration algorithm [18]. We let $\mathcal{V}$ stand for a set of policy trees, though for each tree we need only actually store the top-level action and the vector of values, α.

$t := 1$
$\mathcal{V}_1 :=$ the set of 1-step policy trees (one for each action)
loop
    $t := t + 1$
    compute $\mathcal{V}_t^+$, the set of possibly useful t-step policy trees, from $\mathcal{V}_{t-1}$
    prune $\mathcal{V}_t^+$ to get $\mathcal{V}_t$, the useful set of t-step policy trees
until $\sup_b |V_t(b) - V_{t-1}(b)| < \epsilon$

The idea behind this algorithm is the following: $\mathcal{V}_{t-1}$, the set of useful $(t-1)$-step policy trees, can be used to construct a superset of the useful t-step policy trees. A t-step policy tree is composed of a root node with an associated action, a, with $|\Omega|$ subtrees, which are $(t-1)$-step policy trees. We propose to restrict our choice of subtrees to those $(t-1)$-step policy trees that were useful. For any belief state and any choice of policy subtree, there is always a useful subtree that is at least as good at that state; so there is never any reason to include a non-useful policy subtree.

The time complexity of a single iteration of this algorithm can be divided into two parts: generation and pruning. There are $|\mathcal{A}||\mathcal{V}_{t-1}|^{|\Omega|}$ elements in $\mathcal{V}_t^+$: there are $|\mathcal{A}|$ different ways to choose the action and all possible lists of length $|\Omega|$ may be chosen from the set $\mathcal{V}_{t-1}$ to form the subtrees. The value functions for the policy trees in $\mathcal{V}_t^+$ can be computed efficiently from those of the subtrees.

Although this algorithm may represent a large computational savings over the naive enumeration strategy, it still does more work than may be necessary. Even if $\mathcal{V}_t$ is very small, we must go through the step of generating $\mathcal{V}_t^+$, which always has size exponential in $|\Omega|$. In the next section, we sketch a more efficient algorithm.

4.3 The Witness Algorithm

To improve the complexity of the value-iteration algorithm, we must avoid generating $\mathcal{V}_t^+$; instead, we would like to generate the elements of $\mathcal{V}_t$ directly. If we could do this, we might be able to reach a computation time per iteration that is polynomial in $|\mathcal{S}|$, $|\mathcal{A}|$, $|\Omega|$, $|\mathcal{V}_{t-1}|$, and $|\mathcal{V}_t|$. Cheng [5] and Smallwood and Sondik [25] also try to avoid generating all of $\mathcal{V}_t^+$ by constructing $\mathcal{V}_t$ directly. However, their algorithms still have worst-case running times exponential in at least one of the problem parameters [13]. In fact, if we could solve this problem, then RP=NP [13], so we will pursue a slightly different approach.

Instead of computing $\mathcal{V}_t$ directly, we will compute, for each action a, a set $\mathcal{Q}_t^a$ of t-step policy trees that have action a at their root. We can compute $\mathcal{V}_t$ by taking the union of the $\mathcal{Q}_t^a$ for all actions and pruning as described in the previous section. The *witness* algorithm is a method for computing $\mathcal{Q}_t^a$ in time polynomial in $|\mathcal{S}|$, $|\mathcal{A}|$, $|\Omega|$, $|\mathcal{V}_{t-1}|$, and $|\mathcal{Q}_t^a|$. It is possible that the $\mathcal{Q}_t^a$ are exponentially larger than $\mathcal{V}_t$, but this seems to be rarely the case in practice.

In what sense is the witness algorithm superior to previous algorithms for solving POMDPs, then? Experiments indicate that the witness algorithm is faster in practice over a wide range of problem sizes [13]. The primary complexity-theoretic difference is that the witness algorithm runs in polynomial time in the number of policy trees in $\mathcal{Q}_t^a$. There are example problems that cause the other algorithms, although they never construct the $\mathcal{Q}_t^a$'s directly,

$\mathcal{V}_1 := \{\langle 0, 0, \ldots, 0\rangle\}$
$t := 1$
loop
  $t := t + 1$
  foreach a **in** $\mathcal{A}$
    $\mathcal{Q}_t^a := \text{witness}(\mathcal{V}_{t-1}, a)$
  prune $\bigcup_a \mathcal{Q}_t^a$ to get $\mathcal{V}_t$
until $\sup_b |V_t(b) - V_{t-1}(b)| < \epsilon$

Table 2: Outer loop of the witness algorithm.

to run in time exponential in the number of policy trees in $\mathcal{Q}_t^a$. That means, if we restrict ourselves to problems in which $|\mathcal{Q}_t^a|$ is polynomial, that the resulting running time is polynomial.

From the definition of the state estimator, SE, and the t-step value function, $V_t(b)$, we can express $Q_t^a(b)$ formally as

$$Q_t^a(b) = \sum_{s \in \mathcal{S}} b(s) R(s, a) + \gamma \sum_{o \in \Omega} \Pr(o|a, b) V_{t-1}(b'_o) \quad ,$$

where b'_o is the belief state resulting from taking action a and observing o from belief state b; that is, $b' = \text{SE}(b, a, o)$. Since V is the value of the best action, we have $V_t(b) = \max_a Q_t^a(b)$.

Using arguments similar to those in the previous section, we can show that these Q-functions are piecewise-linear and convex and can be represented by collections of policy trees. Let $\mathcal{Q}_t^a$ be the collection of policy trees that specify Q_t^a. Once again, we can define a unique minimal useful set of policy trees for each Q function. Note that the policy trees needed to represent the function V_t are a subset of the policy trees needed to represent all of the Q_t^a functions: $\mathcal{V}_t \subseteq \bigcup_a \mathcal{Q}_t^a$. This is because maximizing over actions and then policy trees is the same as maximizing over the pooled sets of policy trees.

The code in Table 2 outlines our approach to solving POMDPs. The basic structure remains that of value iteration. At iteration t, the algorithm has a representation of the optimal t-step value function. Within the value-iteration loop, separate Q-functions are found for each action, represented by sets of policy trees. The union of these sets forms a representation of the optimal value function. Since there may be extraneous policy trees in the combined set, it is pruned to yield the useful set of t-step policy trees, $\mathcal{V}_t$.

4.3.1 Witness inner loop

The basic structure of the witness algorithm is as follows. We would like to find a minimal set of policy trees for representing Q_t^a for each a. We consider the Q-functions one at a time. The set U of policy trees is initialized with a single policy tree that is the best for some arbitrary belief state. At each iteration we ask, Is there some belief state, b, for which the true value, $Q_t^a(b)$, computed by one-step lookahead using $\mathcal{V}_{t-1}$, is different from the estimated value, $\hat{Q}_t^a(b)$, computed using the set U? We call such a belief state a *witness* because it

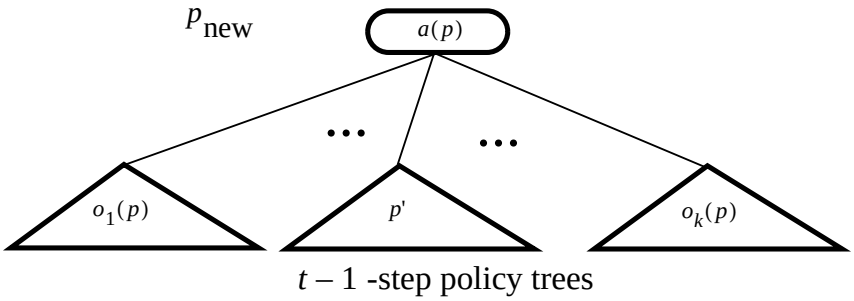

Figure 9: Constructing a new policy.

can, in a sense, testify to the fact that the set U is not yet a perfect representation of $Q_t^a(b)$. Note that for all b, $\hat{Q}_t^a(b) \leq Q_t^a(b)$; the approximation is always an underestimate of the true value function.

Once a witness is identified, we find the policy tree with action a at the root that will yield the best value at that belief state. To construct this tree, we must find, for each observation o, the $(t-1)$-step policy tree that should be executed if observation o is made after executing action a. If this happens, the agent will be in belief state $b' = \text{SE}(b, a, o)$, from which it should execute the $(t-1)$-step policy tree $p_o \in \mathcal{V}_{t-1}$ that maximizes $V_{p_o}(b')$. The tree p is built with subtrees p_o for each observation o. We add the new policy tree to U to improve the approximation. This process continues until we can prove that no more witness points exist and therefore that the current Q-function is perfect.

4.3.2 Identifying a witness

To find witness points, we must be able to construct and evaluate alternative policy trees. If p is a t-step policy tree, o_i an observation, and p' a $(t-1)$-step policy tree, then we define p_{new} as a t-step policy tree that agrees with p in its action and all its subtrees except for observation o_i, for which $o_i(p_{\text{new}}) = p'$. Figure 9 illustrates the relationship between p and p_{new}.

Now we can state the witness theorem [13]: The true Q-function, Q_t^a, differs from the approximate Q-function, $\hat{Q}_t^a$, if and only if there is some $p \in U$, $o \in \mathcal{O}$, and $p' \in \mathcal{V}_{t-1}$ for which there is some b such that

$$V_{p_{\text{new}}}(b) > V_{\tilde{p}}(b) \quad , \tag{1}$$

for all $\tilde{p} \in U$. That is, if there is a belief state, b, for which p_{new} is an improvement over all the policy trees we have found so far, then b is a witness. Conversely, if none of the trees can be improved by replacing a single subtree, there are no witness points.

4.3.3 Checking the witness condition

The witness theorem requires us to search for a $p \in U$, an $o \in \Omega$, a $p' \in \mathcal{V}_{t-1}$ and a b such that Condition 1 holds, or to guarantee that no such quadruple exists. Since U, Ω, and $\mathcal{V}_{t-1}$ are finite and (we hope) small, checking all combinations will not be too time consuming. However, for each combination, we need to search all the belief states to test Condition 1. This we can do using linear programming.

Inputs:

U, p_{new}

Variables:

δ, $b(s)$ for each $s \in \mathcal{S}$

Maximize: δ

Improvement constraints:

For each $\tilde{p}$ in U: $V_{p_{\text{new}}}(b) - V_{\tilde{p}}(b) \geq \delta$

Simplex constraints:

For each $s \in \mathcal{S}$: $b(s) \geq 0$

$\sum_{s \in \mathcal{S}} b(s) = 1$

Table 3: The linear program used to find witness points.

For each combination of p, o and p' we compute the policy tree p_{new}, as described above. For any belief state b and policy tree $\tilde{p} \in U$, $V_{p_{\text{new}}}(b) - V_{\tilde{p}}(b)$ gives the advantage of following policy tree p_{new} instead of $\tilde{p}$ starting from b. We would like to find a b that maximizes the advantage over all policy trees $\tilde{p}$ the algorithm has found so far.

The linear program in Table 3 solves exactly this problem. The variable δ is the minimum amount of improvement of p_{new} over any policy tree in U at b. It has a set of constraints that restrict δ to be a bound on the difference and a set of simplex constraints that force b to be a well-formed belief state. It then seeks to maximize the advantage of p_{new} over all $\tilde{p} \in U$. Since the constraints are all linear, this can be accomplished by linear programming. The total size of the linear program is one variable for each component of the belief state and one representing the advantage, plus one constraint for each policy tree in U, one constraint for each state, and one constraint to ensure that the belief state sums to one.

If the linear program finds that the biggest advantage is not positive, that is, that $\delta \leq 0$, then p_{new} is not an improvement over all $\tilde{p}$s. Otherwise, it is and b is a witness point.

4.3.4 A single step of value iteration

The complete value iteration step starts with an agenda containing a single policy tree. It takes a policy tree off the top of the agenda and uses the linear program in Table 3 to determine whether it is an improvement over the policy trees in U. If a witness point is discovered, its associated policy tree is added to U and all policy trees that differ from the current policy tree in a single subtree are added to the agenda. If no witness points are discovered, then that policy tree is removed from the agenda. When the agenda is empty, the algorithm terminates.

The running time of a single pass of value iteration using the witness algorithm is bounded

by a polynomial in the size of the state space ($|\mathcal{S}|$), the size of the action space ($|\mathcal{A}|$), the number of policy trees in the representation of the previous iteration's value function ($|\mathcal{V}_{t-1}|$), the number of observations ($|\Omega|$), and the number of policy trees in the representation of the current iteration's Q-functions ($\sum_a |\mathcal{Q}_t^a|$). Note that we must assume that the number of bits of precision used in specifying the model is polynomial in these quantities since the polynomial running time of linear programming is expressed as a function of the input precision [23]. A more detailed derivation of the running time is available [13].

5 Understanding Policies

In this section we introduce a very simple example and use it to illustrate some properties of POMDP policies. Other examples are explore in an earlier paper [4].

5.1 The Tiger Problem

Imagine an agent standing in front of two closed doors. Behind one of the doors is a tiger and behind the other is a large reward. If the agent opens the door with the tiger, then a large penalty is received (presumably in the form of some amount of bodily injury). Instead of opening one of the two doors, the agent can listen, in order to gain some information about the location of the tiger. Unfortunately, listening is not free; in addition, it is also not entirely accurate. There is a chance that the agent will hear a tiger behind the left-hand door when the tiger is really behind the right-hand door, and vice versa.

We refer to the state of the world when the tiger is on the left as s_l and when it is on the right as s_r. The actions are LEFT, RIGHT, and LISTEN. The reward for opening the correct door is $+10$ and the penalty for choosing the door with the tiger behind it is -100. The cost of listening is -1. There are only two possible observations: to hear the tiger on the left (TL) or to hear the tiger on the right (TR). Immediately after the agent opens a door and receives a reward or penalty, the problem resets, randomly relocating the tiger behind one of the two doors.

The transition and observation models can be described in detail as follows. The LISTEN action does not change the state of the world. The LEFT and RIGHT actions cause a transition to world state s_l with probability .5 and to state s_r with probability .5. When the world is in state s_l, the LISTEN action results in observation TL with probability 0.85 and the observation TR with probability 0.15; conversely for world state s_r. No matter what state the world is in, the LEFT and RIGHT actions result in either observation with probability 0.5.

5.2 Finite-Horizon Policies

The optimal undiscounted finite-horizon policies for the tiger problem are rather surprising. Let us begin with the policy for the time step $t = 1$, when the agent only gets to make a single decision. If the agent believes with high probability that the tiger is on the left, then the best action is to open the right door; if it believes that the tiger is on the right, the best action is to open the left door. But what if the agent is highly uncertain about the tiger's location? The best thing to do is listen. Guessing incorrectly will incur a penalty of -100,

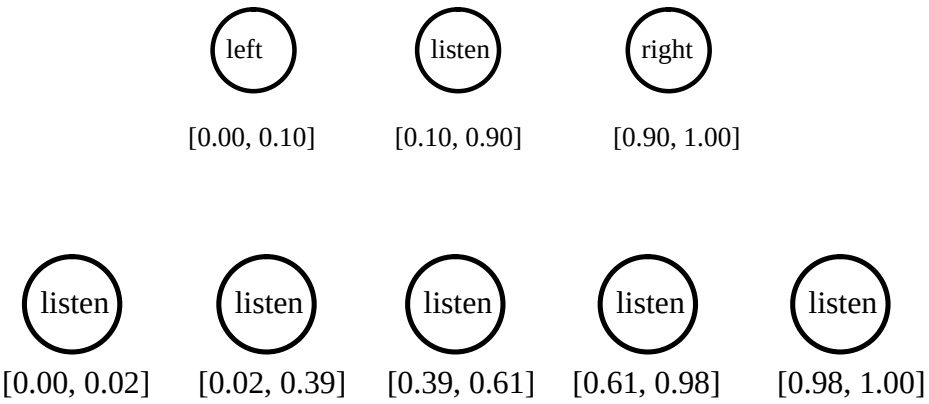

Figure 11: Tiger example policy for $t = 2$

whereas guessing correctly will yield a reward of $+10$. When the agent's belief has no bias either way, it will guess wrong as often as it guesses right, so its expected reward for opening a door will be $(-100 + 10)/2 = -45$. Listening always has value -1, which is greater than the value of opening a door at random. Figure 10 shows the optimal 1-step policy. Each of the policy trees is shown as a node; below each node is the belief interval[2] over which the policy tree dominates; inside each node is the action at the root of the policy tree.

We now move to the case in which the agent can act for two time steps. The policy for $t = 2$ is shown in Figure 11 and has a surprising property: it always chooses to listen. Why? Because if the agent were to open one of the doors at $t = 2$, then, on the next step, the tiger would be randomly placed behind one of the doors and the agent's belief state would be reset to $(0.5, 0.5)$. So after opening a door, the agent would be left with no information about the tiger's location and with one action remaining. We just saw that with one step to go and $b = (0.5, 0.5)$ the best thing to do is listen. Therefore, if the agent opens a door when $t = 2$, it will listen on the last step. It is a better strategy to listen when $t = 2$ in order to make a more informed decision on the last step.

Another interesting property of the 2-step policy is that there are multiple policy trees with the same action. This implies that the value function is not linear, but is made up of five linear regions. The belief states within a single region are similar in that when they are transformed, via $SE(b, a, o)$, the resulting belief states will all lie in the same belief region defined by the policy for $t = 1$. In other words, every single belief state in a particular region, r, for $t = 2$ will, for the same action and observation, be transformed to a belief state that lies in some region, r' of the policy for $t = 1$. This relationship is shown in Figure 12.

The optimal policy for $t = 3$ also consists solely of policy trees with the listen action. If the agent starts from the uniform belief state, $b = (0.5, 0.5)$, listening once does not change the belief state enough to make the expected value of opening a door greater than that of listening. The argument for this parallels that for the 2-step policy.

This argument for listening in the first steps no longer applies after $t = 3$; the optimal policies for $t > 3$ all choose to open a door for some initial belief states. Figure 13 shows the structure that emerges for the optimal policies from $t = 1$ to $t = 4$. Notice that for $t = 3$ there are two nodes that do not have any incoming arcs from $t = 4$. This happens because

[2]The belief interval is specified in terms of $b(s_l)$ only since $b(s_r) = 1 - b(s_l)$.

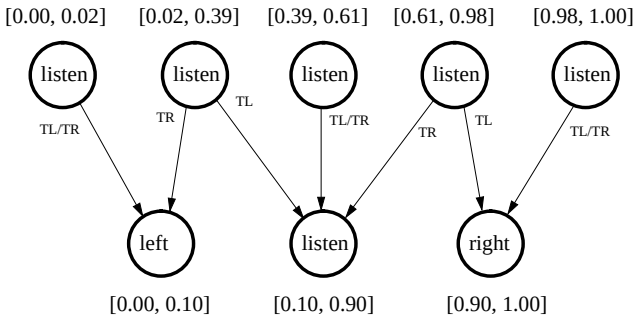

Figure 12: Belief state mapping from $t = 2$ to $t = 1$

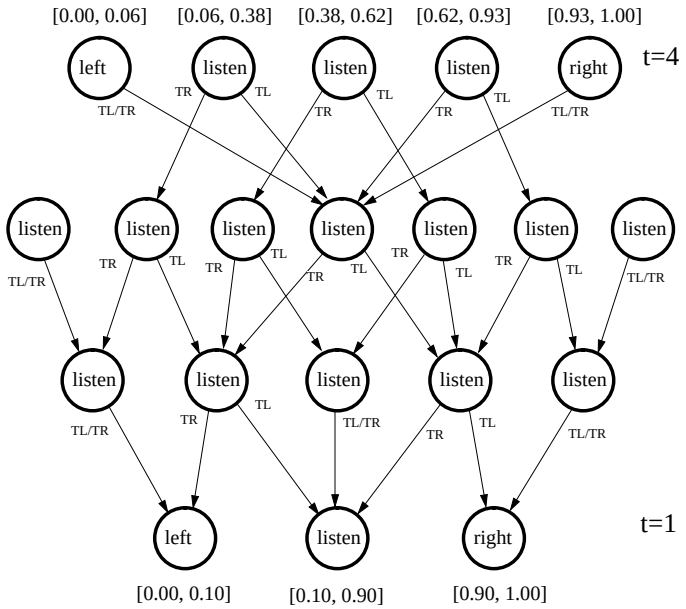

Figure 13: Policies and relationship for $t = 1$ through $t = 4$

there is no belief state at $t = 4$ for which the optimal action and any resulting observation generates a new belief state that lies in either of the regions defined by the unused nodes at $t = 3$.

This graph can also be interpreted as a compact representation of all of the useful policy trees at every level. The forest of policy trees is transformed into a directed acyclic graph by collapsing all of the nodes that stand for the same policy tree into one.

5.3 Infinite-Horizon Policies

When we include a discount factor to decrease the value of future rewards, the structure of the finite-horizon POMDP value function changes slightly. As the horizon, t, increases, the rewards received for the final few steps have decreasing influence on the policy for earlier time steps and the value function begins to converge. In many cases, for large t, and $\gamma = 0.75$, the policy looks much the same as the policy for $t - 1$. Figure 14 shows the solution to the *discounted* finite-horizon version of the tiger problem for large values of t. Notice that the

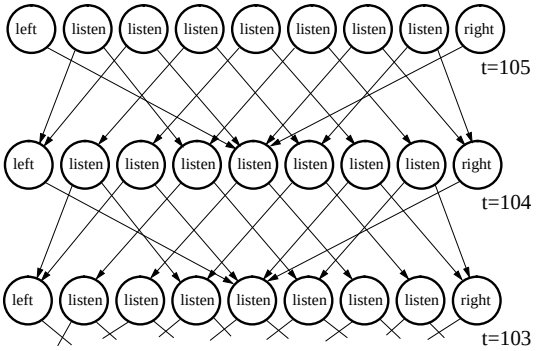

Figure 14: Structure of solution for large t

structure of the graph is exactly the same from one time to the next. The α vectors for each of the nodes, which together define the value function, differ only after the fifteenth decimal place. This structure first appears at time step $t = 56$ and remains constant up to $t = 105$. When $t = 105$, the precision of the algorithm used to calculate the policy can no longer discern any difference between the vectors' values for succeeding intervals. At this point, we have an approximately optimal value function for the infinite-horizon discounted problem.

This POMDP has the property that the optimal infinite-horizon value function has a finite number of linear segments. An associated optimal policy has a finite description and is called finitely transient [26, 22]. POMDPs with optimal finitely transient policies can sometimes be solved in finite time using value iteration. In other POMDPs, the infinite-horizon value function has an infinite number of segments; on these problems the sets $\mathcal{V}_t$ grow with each iteration. The best we can hope for is to solve these POMDPs approximately.

5.4 Policy Graphs

One drawback of the POMDP approach is that the agent must maintain a belief state and use it to select an optimal action on every step; if the underlying state space or $\mathcal{V}$ is large, then this computation can be expensive. In many cases, it is possible to encode the policy in a graph that can be used to select actions without any explicit representation of the belief state [27]; we refer to such graphs as *policy graphs*. Recall Figure 14, in which the algorithm has nearly converged upon an infinite-horizon policy for the tiger problem. Because the policy has the same structure at every level, we can make it stationary by redrawing the edges from one level to itself as if it were the succeeding level. This rearrangement of edges is shown in Figure 15, and the result is redrawn in Figure 16 as a policy graph.

Some of the nodes of the graph will never be visited once either door is opened and the belief state is reset to $(0.5, 0.5)$. If the agent always starts in a state of complete uncertainty, then it will never be in a belief state that lies in the region of these non-reachable nodes. This results in a simpler version of the policy graph, shown in Figure 17. The policy graph has a simple interpretation: keep listening until you have heard the tiger twice more on one side than the other.

Because the nodes represent a partition of the belief space and because all belief states within a particular region will map to a single node on the next level, the policy graph representation does not require the agent to maintain an on-line representation of the belief

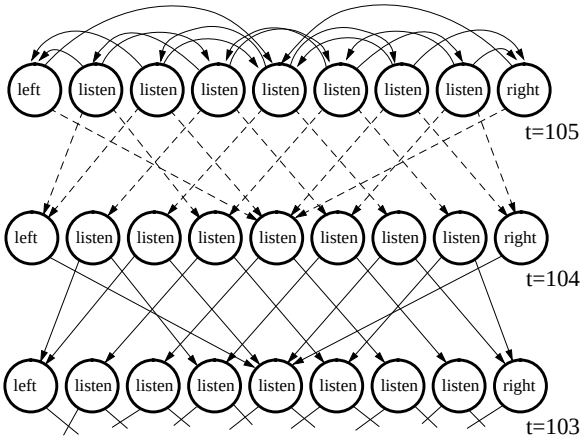

Figure 15: Rearranging edges to form a stationary policy

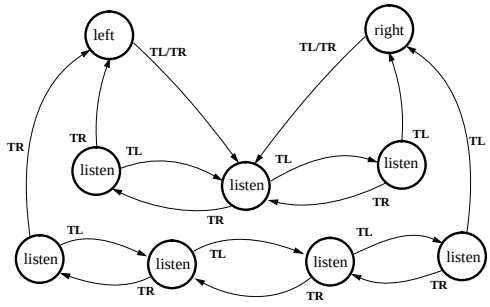

Figure 16: Policy graph for tiger example

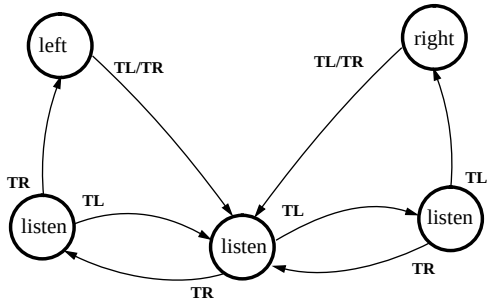

Figure 17: Trimmed policy graph for tiger example

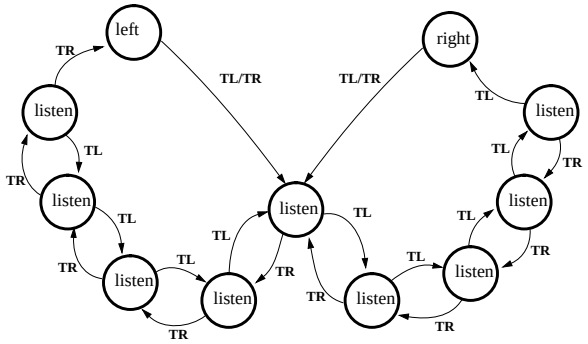

Figure 18: Policy graph for tiger example with listening reliability of 0.65

state; the current node is a sufficient representation of the current belief. In order to execute a policy graph, the initial belief state is used to choose a starting node. After that, the agent need only maintain a pointer to a current node in the graph. On every step, it takes the action specified by the current node, receives an observation, then follows the arc associated with that observation to a new node. This process continues indefinitely.

A policy graph is essentially a finite-state controller. It uses the minimal possible amount of memory to act optimally in a partially observable environment. It is a surprising and pleasing result that it is possible to start with a discrete problem, reformulate it in terms of a continuous belief space, then map the continuous solution back into a discrete controller.

It is also important to note that there is no *a priori* bound on the size of the optimal policy graph in terms of the size of the problem. In the tiger problem, for instance, if the probability of getting correct information from the LISTEN action is reduced from 0.85 to 0.65, then the optimal policy graph, shown in Figure 18, is much larger, because the agent must hear the tiger in the same place 5 times more than in the other before being sufficiently confident to act. As the observation reliability decreases, an increasing amount of memory will be required.

6 Extensions and Conclusions

The POMDP model provides a firm foundation for work on planning under uncertainty in action and observation. It gives a uniform treatment of action to gain information and action to change the world. Although they are derived through the domain of continuous belief spaces, elegant finite-state controllers may sometimes be constructed using algorithms such as the witness algorithm.

However, experimental results [13] suggest, even the witness algorithm becomes impractical for problems of modest size ($|\mathcal{S}| > 15$ and $|\Omega| > 15$). Our current work explores the use of function-approximation methods for representing value functions and the use of simulation in order to concentrate the approximations on the frequently visited parts of the belief space [14]. The results of this work are encouraging and have allowed us to get a very good solution to an 89 state, 16 observation instance of a hallway navigation problem similar to the one described in the introduction. We are optimistic and hope to extend these techniques (and others) to get good solutions to large problems.

Another area that is not addressed in this paper is the acquisition of a world model. One approach is to extend techniques for learning hidden Markov models [21, 28] to learn POMDP models. Then, we could apply algorithms of the type described in this paper to the learned models. Another approach is to combine the learning of the model with the computation of the policy. This approach has the potential significant advantage of being able to learn a model that is complex enough to support optimal (or good) behavior without making irrelevant distinctions; this idea has been pursued by Chrisman [6] and McCallum [16, 17].

References

[1] K. J. Astrom. Optimal control of Markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications*, 10:174–205, 1965.

[2] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control.* Athena Scientific, Belmont, Massachusetts, 1995. Volumes 1 and 2.

[3] Craig Boutilier and David Poole. Computing optimal policies for partially observable decision processes using compact representations. (manuscript), 1995.

[4] Anthony R. Cassandra, Leslie Pack Kaelbling, and Michael L. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, WA, 1994.

[5] Hsien-Te Cheng. *Algorithms for Partially Observable Markov Decision Processes.* PhD thesis, University of British Columbia, British Columbia, Canada, 1988.

[6] Lonnie Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 183–188, San Jose, California, 1992. AAAI Press.

[7] Thomas Dean, Leslie Pack Kaelbling, Jak Kirman, and Ann Nicholson. Planning under time constraints in stochastic domains. *Artificial Intelligence*, To Appear.

[8] Mark Drummond and John Bresina. Anytime synthetic projection. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 138–144. Morgan Kaufmann, 1990.

[9] Ronald A. Howard. *Dynamic Programming and Markov Processes.* The MIT Press, Cambridge, Massachusetts, 1960.

[10] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the American Society of Mechanical Engineers Journal of Basic Engineering*, 82:35–45, 1960.

[11] N. Kushmerick, S. Hanks, and D. Weld. An Algorithm for Probabilistic Planning. Technical Report 93-06-03, University of Washington Department of Computer Science and Engineering, June 1993. To appear in Artificial Intelligence.

[12] Michael L. Littman. The witness algorithm: Solving partially observable Markov decision processes. Technical Report CS-94-40, Brown University, Providence, Rhode Island, 1994.

[13] Michael L. Littman, Anthony R. Cassandra, and Leslie Pack Kaelbling. An efficient algorithm for dynamic programming in partially observable Markov decision processes. Technical Report CS-95-19, Brown University, Providence, Rhode Island, 1995.

[14] Michael L. Littman, Anthony R. Cassandra, and Leslie Pack Kaelbling. Learning policies for partially observable environments: Scaling up. In *Proceedings of the Twelfth International Conference on Machine Learning*. Morgan Kaufmann, 1995.

[15] William S. Lovejoy. A survey of algorithmic methods for partially observed Markov decision processes. *Annals of Operations Research*, 28(1):47–65, 1991.

[16] R. Andrew McCallum. Overcoming incomplete perception with utile distinction memory. In *Proceedings of the Tenth International Conference on Machine Learning*, Amherst, Massachusetts, 1993. Morgan Kaufmann.

[17] R. Andrew McCallum. Instance-based utile distinctions for reinforcement learning with hidden state. In *Proceedings of the Twelfth International Conference Machine Learning*, pages 387–395, San Francisco, CA, 1995. Morgan Kaufmann.

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

[19] Robert C. Moore. A formal theory of knowledge and action. In Jerry R. Hobbs and Robert C. Moore, editors, *Formal Theories of the Commonsense World*. Ablex Publishing Company, Norwood, New Jersey, 1985.

[20] Martin L. Puterman. *Markov Decision Processes*. John Wiley & Sons, New York, 1994.

[21] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[22] Katsushige Sawaki and Akira Ichikawa. Optimal control for partially observable Markov decision processes over an infinite horizon. *Journal of the Operations Research Society of Japan*, 21(1):1–14, March 1978.

[23] Alexander Schrijver. *Theory of Linear and Integer Programming*. Wiley-Interscience, 1986.

[24] Satinder Pal Singh, Tommi Jaakkola, and Michael I. Jordan. Model-free reinforcement learning for non-Markovian decision problems. In *Machine Learning: Proceedings of the Eleventh International Conference*, pages 284–292, San Francisco, California, 1994. Morgan Kaufmann.

[25] Richard D. Smallwood and Edward J. Sondik. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, 21:1071–1088, 1973.

[26] Edward J. Sondik. *The Optimal Control of Partially Observable Markov Processes*. PhD thesis, Stanford University, Stanford, California, 1971.

[27] Edward J. Sondik. The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. *Operations Research*, 26(2):282–304, 1978.

[28] Andreas Stolcke and Stephen Omohundro. Hidden Markov model induction by Bayesian model merging. In Stephen José Hanson, Jack D. Cowan, and C. Lee Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 11–18. Morgan Kaufmann, San Mateo, California, 1993.

[29] Paul Tseng. Solving H-horizon, stationary Markov decision problems in time proportional to $\log(H)$. *Operations Research Letters*, 9(5):287–297, 1990.

[30] C. C. White and D. Harrington. Application of Jensen's inequality for adaptive suboptimal design. *J. Optim. Theory Appl.*, 32:89–100, 1980.

[31] Chelsea C. White, III. Partially observed Markov decision processes: A survey. *Annals of Operations Research*, 32, 1991.

[32] Ronald J. Williams and Leemon C. Baird, III. Tight performance bounds on greedy policies based on imperfect value functions. Technical Report NU-CCS-93-13, Northeastern University, College of Computer Science, Boston, MA, November 1993.