Planning and Acting in Partially Observable Stochastic Domains

Leslie Pack Kaelbling [1,2]

Computer Science Department
Brown University, Box 1910
Providence, RI 02912-1910, USA

Michael L. Littman [3]

Department of Computer Science
Duke University
Durham, NC 27708-0129, USA

Anthony R. Cassandra [1]

Microelectronics and Computer Technology Corporation (MCC)
3500 West Balcones Center Drive
Austin, TX 78759-5398, USA

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 how our approach relates to previous work, the complexity of finding exact solutions to POMDPs, and of some possibilities for finding approximate solutions.

Key words: planning, uncertainty, partially observable Markov decision processes

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

[1] Supported in part by NSF grants IRI-9453383 and IRI-9312395.
[2] Supported in part by DARPA/Rome Labs Planning Initiative grant F30602-95-1-0020
[3] Supported in part by Bellcore and NSF CAREER grant IRI-9702576.

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 [26] 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.

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 [43]. 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 situations 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 [15,61,13]. 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 algorithms for more complex and efficient representations [6]. Section 6 describes the relation between the present approach and prior research in more detail.

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.

Thus, from the POMDP perspective, optimal performance involves something akin to a "value of information" calculation, only more complex; the agent chooses between actions based on the amount of information they provide, the amount of reward they produce, and how they change the state of the world.

This paper is intended to make two contributions. The first is to recapitulate work from the operations-research literature [36,42,56,59,64] and to describe its connection to closely related work in AI. The second is to describe a novel algorithmic approach for solving POMDPs exactly. 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 related work and 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

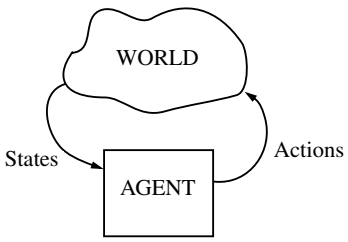

Fig. 1. An MDP models the synchronous interaction between agent and world.

current state—it has complete and perfect perceptual abilities.

Markov decision processes are described in depth in a variety of texts [3,49]; 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—the state and reward at time $t + 1$ is dependent only on the state at time t and the action at time t.

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

$$E\left[\sum_{t=0}^{k-1} r_t\right] \quad ,$$

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 (closer to 1), the more effect future rewards have on current decision making. In our forthcoming 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*, $\pi : \mathcal{S} \to \mathcal{A}$, is a situation-action mapping that 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 situation-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, the optimal policy is not typically stationary: 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 equation 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 situation-action mapping 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 [24] 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 is only guaranteed to occur 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 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 [62] 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 [66] 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,

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

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

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 [49].

3 Partial Observability

For MDPs we can compute the optimal policy π and use it to act by simply executing $\pi(s)$ for current state s. 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 [55]. Randomness effectively allows the agent to sometimes choose different actions in different locations with the same appearance, increasing the probability that it might choose a good action; in practice deterministic observation-action mappings are prone to getting trapped in deterministic loops [32].

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.

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.[4] 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 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

[4] 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; such a model could be converted into a POMDP model as described above, at the cost of expanding the state space.

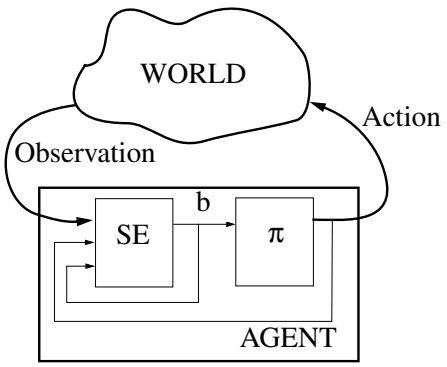

Fig. 2. A POMDP agent can be decomposed into a state estimator (SE) and a policy (π).

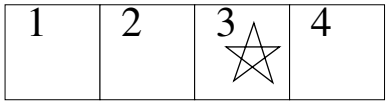

Fig. 3. In this simple POMDP environment, the empty squares are all indistinguishable on the basis of their immediate appearance, but the evolution of the belief state can be used to model the agent's location.

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 [1,56]. This means that the process over belief states is Markov, and that no additional data about the past would help to increase the agent's expected reward.

To illustrate the evolution of a belief state, we will use the simple example depicted in Figure 3; the algorithm for computing belief states is provided in the next section. 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 $0 \leq b(s) \leq 1$ 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 $\mathrm{SE}(b,a,o)$ 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

$$\begin{aligned}\tau(b,a,b') &= \Pr(b'|a,b) \\ &= \sum_{o\in\Omega} \Pr(b'|a,b,o)\Pr(o|a,b) \quad ,\end{aligned}$$

where

$$\Pr(b'|b,a,o) = \begin{cases} 1 \text{ if } \mathrm{SE}(b,a,o) = b' \\ 0 \text{ otherwise} \quad ; \end{cases}$$

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 appears to be 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, the belief state represents the true occupation probabilities for all states $s \in \mathcal{S}$, and therefore the reward function ρ represents the true expected reward to the agent.

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 original POMDP [59,1]. 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

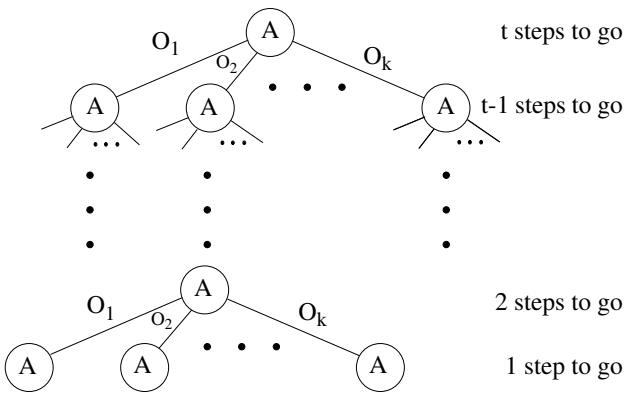

Fig. 4. A t-step policy tree captures a sequence of t steps, each of which can be conditioned on the outcome of previous actions. Each node is labelled with the action that should be taken if it is reached.

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 non-stationary 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. [5]

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

[5] Policy trees are essentially equivalent to "decision trees" as used in decision theory to represent a sequential decision policy; but not to "decision trees" as used in machine learning to compactly represent a single-stage decision rule.

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 non-stationary 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 V_p that is linear in b, and V_t is the upper surface of this collection of 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 a bowl shape that is composed of planar facets; a typical example is shown in Figure 6, but it is possible for the "bowl" to be tipped on its side or to degenerate to a single plane. This general pattern repeats itself in higher dimensions, but becomes difficult to contemplate and even harder to draw!

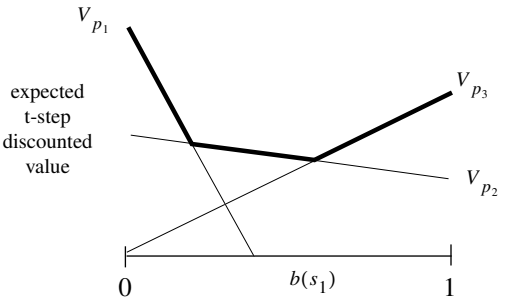

Fig. 5. The optimal t-step value function is the upper surface of the value functions associated with all t-step policy trees.

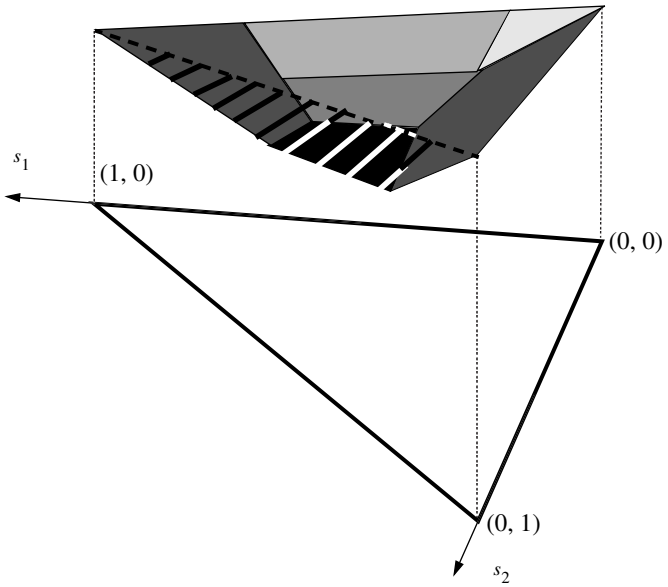

Fig. 6. A value function in three dimensions is made up of the upper surface of a set of planes.

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. This has some connection to the notion of "value of information," [25] where an agent can incur a cost to move it from a high-entropy to a low-entropy state; this is only worthwhile when the value of the information (the difference in value between the two states) exceeds the cost of gaining the information.

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 situation-action mapping for execution on the t^{th} step from the end. The optimal value

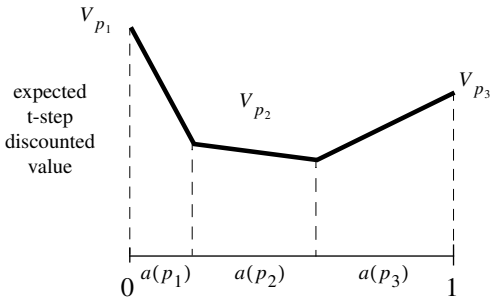

Fig. 7. The optimal t-step situation-action mapping is determined by projecting the optimal value function back down onto the belief space.

function can be projected back down onto the belief space, yielding a partition into polyhedral regions. Within each region, there is some single policy tree p such that $b \cdot \alpha_p$ is maximal over the entire region. The optimal action for each belief state in this region is $a(p)$, the action in the root node of policy tree p; furthermore, the entire policy tree p can be executed from this point by conditioning the choice of further actions directly on observations, without updating the belief state (though this is not necessarily an efficient way to represent a complex policy). Figure 7 shows the projection of the optimal value function down into a policy partition in the two-dimensional example introduced in Figure 5; over each of the intervals illustrated, a single policy tree can be executed to maximize expected reward.

4.2 Value Functions as Sets of Vectors

It is possible, in principle, that every possible policy tree 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 tree p_d is completely dominated by (everywhere less than or equal to) the value function for policy tree p_b. The situation with the value function for policy tree 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, $\tilde{\mathcal{V}}$, it is possible to define a unique[6] minimal subset $\mathcal{V}$ that represents the same value function. We will call this a *parsimonious* representation of the value function, and say that a policy tree is *useful* if it is a component of the parsimonious representation of the value function.

[6] We assume here that two policy trees with the same value function are identical.

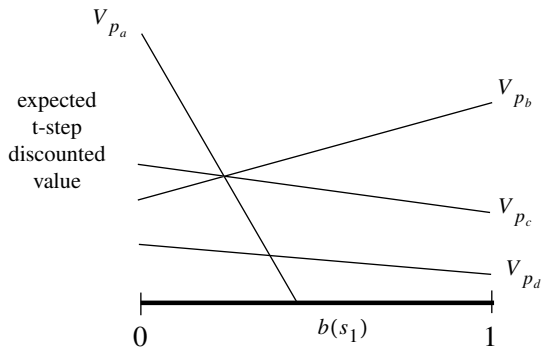

Fig. 8. Some policy trees may be totally dominated by others and can be ignored.

Given a vector, α, and a set of vectors $\mathcal{V}$, we define $\mathcal{R}(\alpha, \mathcal{V})$ to be the region of belief space over which α dominates; that is,

$$\mathcal{R}(\alpha, \mathcal{V}) = \{b \mid b \cdot \alpha > b \cdot \tilde{\alpha}, \text{for all } \tilde{\alpha} \in \mathcal{V} - \alpha \text{ and } b \in \mathcal{B}\} \quad .$$

It is relatively easy, using a linear program, to find a point in $\mathcal{R}(\alpha, \mathcal{V})$ if one exists, or to determine that the region is empty [9].

The simplest pruning strategy, proposed by Sondik [58,42], is to test $\mathcal{R}(\alpha, \tilde{\mathcal{V}})$ for every α in $\tilde{\mathcal{V}}$ and remove those α that are nowhere dominant. A much more efficient pruning method was proposed by Lark and White [64] and is described in detail by Littman [35] and by Cassandra [9]. Because it has many subtle technical details, it is not described here.

4.3 One Step of Value Iteration

The value function for a POMDP can be computed using value iteration, with the same basic structure as for the discrete MDP case. The new problem, then, is how to compute a parsimonious representation of V_t from a parsimonious representation of V_{t-1}.

One of the simplest algorithms for solving this problem [58,42], which we call exhaustive enumeration, works by constructing a large representation of V_t, then pruning it. 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. 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 $\mathcal{V}_t^+$ of the useful t-step policy trees. A t-step policy tree is composed of a root node with an associated action a and $|\Omega|$ subtrees, each a $(t-1)$-step policy tree. 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; 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. Pruning requires one linear program for each element of the starting set of policy trees and does not add to the asymptotic complexity of the algorithm.

Although it keeps parsimonious representations of the value functions at each step, this algorithm still does more much work than may be necessary. Even if $\mathcal{V}_t$ is very small, it goes through the step of generating $\mathcal{V}_t^+$, which always has size exponential in $|\Omega|$. In the next sections, we present the Witness algorithm and some complexity analysis, and then briefly outline some other algorithms for this problem that attempt to be more efficient than the approach of exaustively generating $\mathcal{V}_t^+$.

4.4 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 [10] and Smallwood and Sondik [56] 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 [34]. In fact, the existence of an algorithm that runs in time polynomial in $|\mathcal{S}|$, $|\mathcal{A}|$, $|\Omega|$, $|\mathcal{V}_{t-1}|$, and $|\mathcal{V}_t|$ would settle the long-standing complexity-theoretic question "Does NP=RP?" in the affirmative [34], 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 $\mathcal{Q}_t^a$ sets 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 $|\mathcal{Q}_t^a|$ (specifically, run time is polynomial in the size of the inputs, the outputs, and an important intermediate result). 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 [34]. 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, 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 running time of Witness is polynomial. It is worth noting, however, that it is possible to create families of POMDPs that Cheng's linear support algorithm (sketched in Section 4.5) can solve in polynomial time that take the witness exponential time to solve; they are problems in which $\mathcal{S}$ and $\mathcal{V}_t$ are very small and $\mathcal{Q}_t^a$ is exponentially larger for some action a.

From the definition of the state estimator SE and the t-step value function $V_t(b)$, we can express $Q_t^a(b)$ (recall that this is the value of taking action a in belief state b and continuing optimally for $t-1$ steps) 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, $b' = \mathrm{SE}(b,a,o)$. 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 Section 4.1, 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 for each action, represented by parsimonious sets of policy trees, are returned by calls to `witness` using the value function from the previous iteration. 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, $\mathcal{V}_t$.

4.4.1 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_a of policy trees is initialized with a single policy tree, with action a at the root, that is the best for some arbitrary belief state (this is easy to do, as described in the following paragraph). At each

$\mathcal{V}_1 := \{\langle 0, 0, \ldots, 0\rangle\}$
$t := 1$
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.

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_a? We call such a belief state a *witness* because it can, in a sense, testify to the fact that the set U_a 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_a 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.

4.4.2 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* [34]: The true Q-function, Q_t^a, differs from the approximate Q-function, $\hat{Q}_t^a$, if and only if there is some $p \in U_a$, $o \in \Omega$, 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 $\tilde{p} \in U_a$. 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,

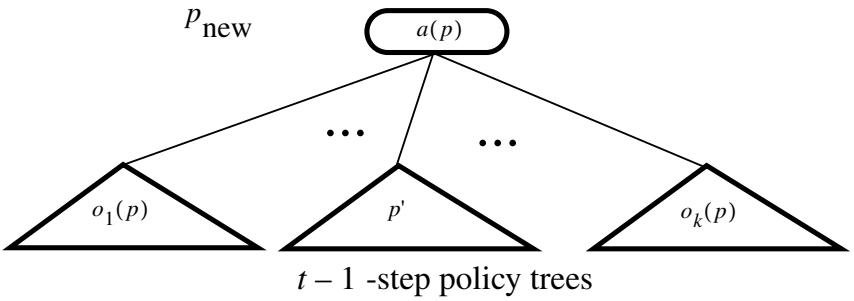

Fig. 9. A new policy tree can be constructed by replacing one of its subtrees.

there are no witness points. A proof of this theorem is included in the appendix.

4.4.3 Checking the witness condition

The witness theorem requires us to search for a $p \in U_a$, an $o \in \Omega$, a $p' \in \mathcal{V}_{t-1}$ and a $b \in \mathcal{B}$ such that Condition 1 holds, or to guarantee that no such quadruple exists. Since U_a, 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.

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 $\tilde{p} \in U_a$, $V_{p_{\text{new}}}(b) - V_{\tilde{p}}(b)$ 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_a 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 $\tilde{p} \in U_a$. 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. [7]

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 $\tilde{p}$ trees. Otherwise, it is and b is a witness point.

[7] In many linear-programming packages, all variables have implicit non-negativity constraints, so the $b(s) \geq 0$ constraints are not needed.

Inputs:
U_a, p_{new}

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

Maximize: δ

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

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.

4.4.4 A single step of value iteration

The complete value-iteration step starts with an agenda containing any single useful policy tree and with U_a empty. It takes a policy tree off the top of the agenda and uses it as p_{new} in the linear program of Table 3 to determine whether it is an improvement over the policy trees in U_a. If a witness point is discovered, the best policy tree for that point is calculated and added to U_a 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.

Since we know that no more than $\mathcal{Q}_t^a$ witness points are discovered (each adds a tree to the set of useful policy trees), only $|\mathcal{V}_{t-1}||\Omega||\mathcal{Q}_t^a|$ trees can ever be added to the agenda (in addition to the one tree in the initial agenda). Each linear program solved has $|\mathcal{S}|$ variables and no more than $1 + |\mathcal{S}| + |\mathcal{Q}_t^a|$ constraints. Each of these linear programs either removes a policy tree from the agenda (this happens at most $1 + (|\mathcal{V}_{t-1}| - 1)|\Omega||\mathcal{Q}_t^a|$ times) or a witness point is discovered (this happens at most $|\mathcal{Q}_t^a|$ times).

These facts imply that 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 [54].

4.5 Alternative Approaches

The witness algorithm is by no means the only exact algorithm for solving finite-horizon POMDPs. The first such algorithm was described by Sondik [58,56]. The one-pass works by identifying linear regions of the value function one at a time. For each one, it creates a set of constraints that form the border of the true region, then searches those borders to determine whether another region exists beyond the border. Although the algorithm is sophisticated and, in principle, avoids exhaustively enumerating the set of possibly useful policy trees at each iteration, it appears to run more slowly than the simpler enumeration methods in practice, at least for problems with small state spaces [10].

In the process of motivating the one-pass algorithm, Sondik [58] applies the same ideas to finding Q-functions instead of the complete value function. The resulting algorithm might be called the two-pass algorithm [9], and, its form is much like the witness algorithm because it first constructs each separate Q-function, then combines the Q-functions together to create the optimal value function. Although it appears that the algorithm attracted no attention and was never implemented in over 25 years after the completion of Sondik's dissertation, it was recently implemented and found to be faster than any of the algorithms that predated the witness algorithm [9].

As pointed out in Section 4, value functions in belief space have a natural geometric interpretation. For small state spaces, algorithms that exploit this geometry are quite efficient [16]. An excellent example of this is Cheng's linear support algorithm [10]. This algorithm can be viewed as a variation of the witness algorithm in which witness points are sought at the corners of regions of the approximate value function defined by the algorithm's equivalent of the set U. In two dimensions, these corners can be found easily and efficiently; the linear support algorithm can be made to run in low-order polynomial time for problems with two states. In higher dimensions, more complex algorithms are needed and the number of corners is often exponential in the dimensionality. Thus, the geometric approaches are useful only in POMDPs with extremely small state spaces.

Zhang and Liu [67] describe the incremental-pruning algorithm, later generalized by Cassandra, Littman, and Zhang [7]. This algorithm is simple to implement and empirically faster than the witness algorithm, while sharing its good worst-case complexity in terms of $\sum_a |\mathcal{Q}_t^a|$. The basic algorithm works like the exhaustive enumeration algorithm described in Section 4.3, but differs in that it repeatedly prunes out non-useful policy trees during the generation

procedure. As a result, compared to exhaustive enumeration, very few non-useful policy trees are considered and the algorithm runs extremely quickly.

White and Scherer [65] propose an alternative approach in which the reward function is changed so that all of the algorithms discussed in this chapter will tend to run more efficiently. This technique has not yet been combined with the witness algorithm, and may provide some improvement.

4.6 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 [63], 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 [59,51].

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.

Sondik [59] and Hansen [23] have shown how to use algorithms like witness algorithm that perform exact dynamic-programming backups in POMDPs in a policy-iteration algorithm to find exact solutions to many infinite-horizon problems.

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 explored in an earlier paper [8].

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 (essentially resetting the problem). 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 striking in the richness of their structure. Let us begin with the situation-action mapping 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, 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 non-stationary policy. Each of the policy trees is shown as a node; below each node is the belief interval[8] over which the policy tree dominates; inside each node is the action at the root of the policy tree.

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

Fig. 10. The optimal situation-action mapping for $t = 1$ for the tiger problem shows that each of the three actions is optimal for *some* belief state.

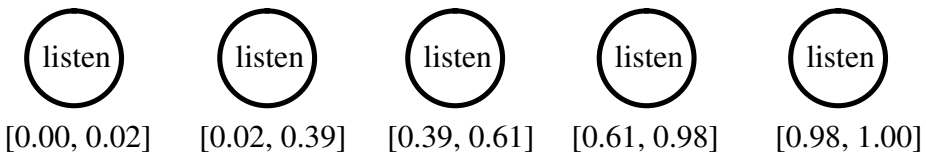

Fig. 11. The optimal situation-action mapping for $t = 2$ in the tiger problem consists only of the LISTEN action.

We now move to the case in which the agent can act for two time steps. The optimal 2-step non-stationary policy begins with the situation-action mapping for $t = 2$ shown in Figure 11. This situation-action mapping has a surprising property: it never chooses to act, only 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 non-stationary policy is that there are multiple policy trees with the same action at the root. 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 $\mathrm{SE}(b, a, o)$, the resulting belief states will all lie in the same belief region defined by the situation-action mapping for $t = 1$. In other words, every single belief state in a particular region r of the situation-action mapping for $t = 2$, will, for the same action and observation, be transformed to a belief state that lies in some region r' of the situation-action mapping for $t = 1$. This relationship is shown in Figure 12.

The optimal non-stationary policy for $t = 3$ also consists solely of policy trees with the listen action at their roots. 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 $t = 2$ case.

This argument for listening in the first steps no longer applies after $t = 3$;

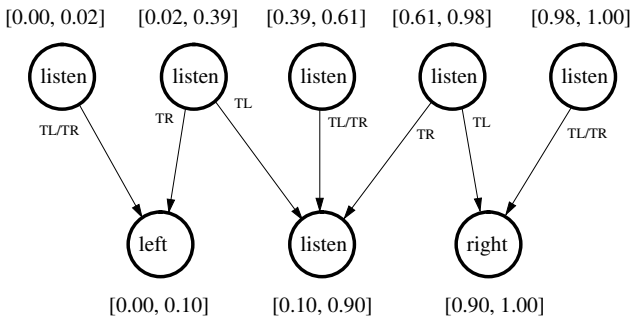

Fig. 12. The optimal non-stationary policy for $t = 2$ illustrates belief state transformations from $t = 2$ to $t = 1$. It consists of five separate poicy trees.

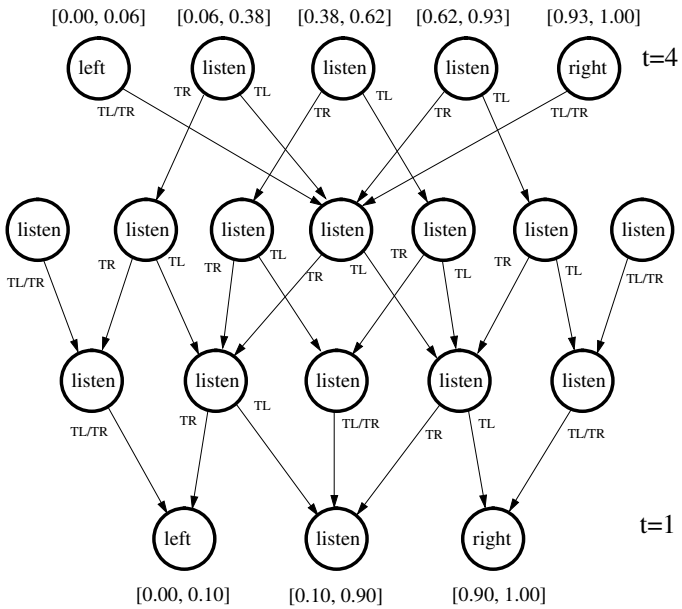

Fig. 13. The optimal non-stationary policy for $t = 4$ has a rich structure.

the optimal situation-action mappings for $t > 3$ all choose to open a door for some belief states. Figure 13 shows the structure that emerges in the optimal non-stationary policy for $t = 4$. Notice that for $t = 3$ there are two nodes that do not have any incoming arcs from $t = 4$. This happens because 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.

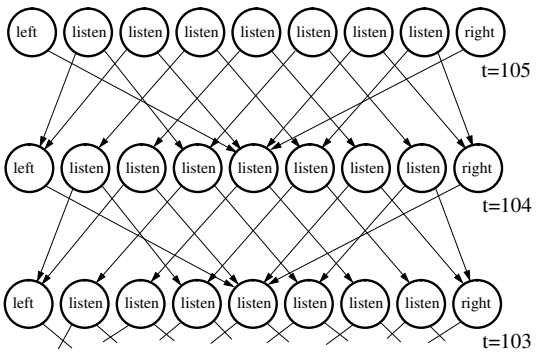

Fig. 14. The optimal non-stationary policy for large t converges.

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 situation-action mappings for earlier time steps and the value function begins to converge. In many discounted POMDP problems, the optimal situation-action mapping for large t looks much the same as the optimal situation-action mapping for $t-1$. Figure 14 shows a portion of the optimal non-stationary policy for the *discounted* finite-horizon version of the tiger problem for large values of t. Notice that the structure of the graph is exactly the same from one time to the next. The 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 through $t = 105$. When $t = 105$, the precision of the algorithm used to calculate the situation-action mappings 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 [58,51,9]. POMDPs with finitely transient optimal policies can sometimes be solved in finite time using value iteration. In POMDPs with optimal policies that are not finitely transient, 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. It is not known whether there is a way of using the value-iteration approach described in this paper for solving *all* POMDPs with finitely transient optimal policies in finite time; we conjecture that there is. The only finite-time algorithm that has been described for solving POMDPs with finitely transient optimal policies over the infinite horizon is a version of policy iteration described by Sondik [58]. The simpler policy-

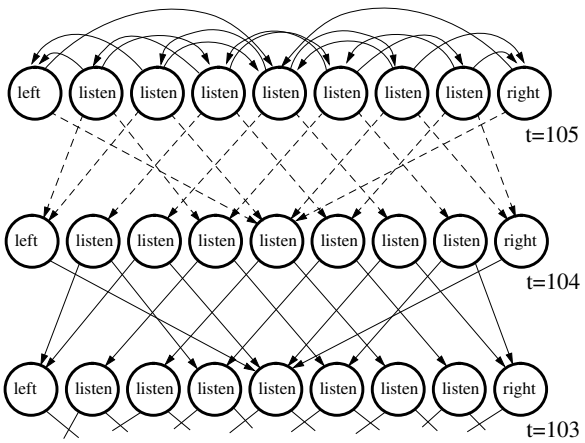

Fig. 15. Edges can be rearranged to form a stationary policy.

iteration algorithm due to Hansen [23] has not been proven to converge for all such POMDPs. [9]

5.4 Plan 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 that $\mathcal{V}$ 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 [59]; we refer to such graphs as *plan graphs*. Recall Figure 14, in which the algorithm has nearly converged upon an infinite-horizon policy for the tiger problem. Because the situation-action mappings at every level have the same structure, we can make the non-stationary policy into a stationary one 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 plan 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

[9] As a technical aside, if there are POMDPs that have finitely transient optimal policies for which neither value iteration nor Hansen's policy-iteration algorithm converges, the tiger problem is a good candidate. This is because the behavior of these algorithms on this problem appears to be extremely sensitive to the numerical precision used in comparisons—the better the precision, the longer the algorithms take to converge. In fact, it may be the case that imprecision is *necessary* for the algorithms to converge on this problem, although it is difficult to test this without detailed formal analysis. Sondik's proof that his policy-iteration algorithm converges depends on controlled use of imprecision and we have not studied how that could best be used in the context of value iteration.

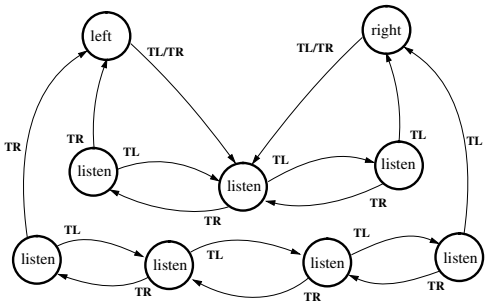

Fig. 16. The optimal infinite-horizon policy for the tiger problem can be drawn as a plan graph. This structure counts the relative number of times the tiger was heard on the left as compared to the right.

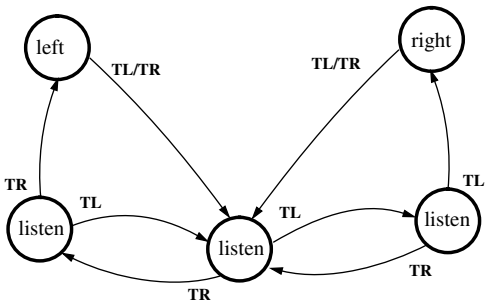

Fig. 17. Given the initial belief state of $(0.5, 0.5)$ for the tiger problem, some nodes of the plan graph can be trimmed.

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 plan graph, shown in Figure 17. The plan 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 plan graph representation does not require the agent to maintain an on-line representation of the belief state; the current node is a sufficient representation of the current belief. In order to execute a plan 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 plan 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. Furthermore, the extraction

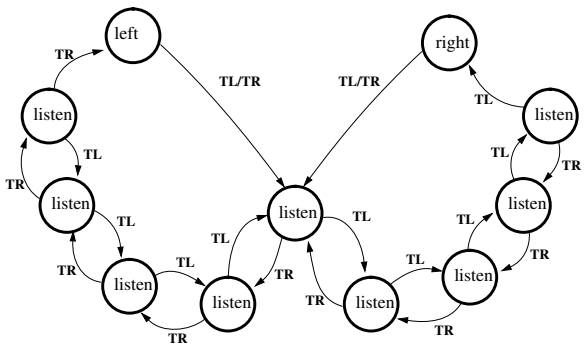

Fig. 18. More memory is needed in the tiger problem when listening reliability is reduced to 0.65.

of the controller can be done automatically from two successive equal value functions.

It is also important to note that there is no known *a priori* bound on the size of the optimal plan 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 plan graph, shown in Figure 18, is much larger, because the agent must hear the tiger on one side 5 times more than in the other before being sufficiently confident to act. As the observation reliability decreases, an increasing amount of memory is required.

6 Related Work

In this section, we examine how the assumptions of the POMDP model relate to earlier work on planning in AI. We consider only models with finite state and action spaces and static underlying dynamics, as these assumptions are consistent with the majority of work in this area. Our comparison focuses on issues of imperfect knowledge, uncertainty in initial state, the transition model, the observation model, the objective of planning, the representation of domains, and plan structures. The most closely related work to our own is that of Kushmerick, Hanks, and Weld [30] on the BURIDAN system, and Draper, Hanks and Weld [14] on the C-BURIDAN system.

6.1 Imperfect Knowledge

Plans generated using standard MDP algorithms and classical planning algorithms [10] assume that the underlying state of the process will be known with certainty during plan execution. In the MDP framework, the agent is informed of the current state each time it takes an action. In many classical planners (e.g., SNLP [39], UCPOP [45]), the current state can be calculated trivially from the known initial state and knowledge of the deterministic operators.

The assumption of perfect knowledge is not valid in many domains. Research on epistemic logic [43,44,52] relaxes this assumption by making it possible to reason about what is and is not known at a given time. Unfortunately, epistemic logics have not been used as a representation in automatic planning systems, perhaps because the richness of representation they provide makes efficient reasoning very difficult.

A step towards building a working planning system that reasons about knowledge is to relax the generality of the logic-based schemes. The approach of CNLP [46] uses three-valued propositions where, in addition to true and false, there is a value *unknown*, which represents the state when the truth of the proposition is not known. Operators can then refer to whether propositions have an *unknown* value in their preconditions and can have the value in their effects. This representation for imperfect knowledge is only appropriate when the designer of the system knows, in advance, what aspects of the state will be known and unknown. It is insufficient for multiple agents reasoning about each others' knowledge and for representing certain types of correlated uncertainty [20].

Formulating knowledge as predicate values that are either known or unknown makes it impossible to reason about gradations of knowledge. For example, an agent that is fairly certain that it knows the combination to a lock might be willing to try to unlock it before seeking out more precise knowledge. Reasoning about levels of knowledge is quite common and natural in the POMDP framework. As long as an agent's state of knowledge can be expressed as a probability distribution over possible states of the world, the POMDP perspective applies.

[10] By "classical planning" we mean linear or partial-order planners using STRIPS-LIKE OPERATORS.

6.2 Initial State

Many classical planning systems (SNLP, UCPOP, CNLP) require the starting state to be known during the planning phase. An exception is the U-PLAN [38] system, which creates a separate plan for each possible initial state with the aim of making these plans easy to merge to form a single plan. Conditional planners typically have some aspects of the initial state unknown. If these aspects are important to the planning process, they are tested during execution.

In the POMDP framework, the starting state is not required to be known precisely and can instead be represented as a probability distribution over possible states. BURIDAN and C-BURIDAN also use probability distributions over states as an internal representation of uncertainty, so they can deal with initial-state uncertainty in much the same way.

6.3 Transition Model

In classical planning systems, operators have deterministic effects. The plans constructed are brittle, since they apply to a specific starting state and require the trajectory through the states to go exactly as expected. Many domains are not easily modeled with deterministic actions, since an action can have different results, even when applied in exactly the same state.

Extensions to classical planning, such as CNLP [46] and CASSANDRA [48] have considered operators with nondeterministic effects. For each operator, there is a set of possible next states that could occur. A drawback of this approach is that it gives no information about the relative likelihood of the possible outcomes. These systems plan for every possible contingency to ensure that the resulting plan is guaranteed to lead to a goal state.

Another approach used in modeling nondeterministic actions is to define a probability distribution over the possible next states. This makes it possible to reason about which of the resulting states are more likely and makes it possible to assess whether a plan is *likely* to reach the goal even if it is not guaranteed to do so. This type of action model is used in MDPs and POMDPs as well as in BURIDAN and C-BURIDAN. Other work [5,15,19] has used representations that can be used to compute probability distributions over future states.

6.4 Observation Model

When the starting state is known and actions are deterministic, there is no need to get feedback from the environment when executing a plan. However, if the starting state is unknown or the actions have nondeterministic effects, more effective plans can be built by exploiting feedback, or observations, from the environment concerning the identity of the current state.

If observations reveal the precise identity of the current state, the planning model is called "completely observable." The MDP model, as well as some planning systems such as CNLP and PLINTH [18,19] assume complete observability. Other systems, such as BURIDAN and MAXPLAN [37], have no observation model and can attack "completely unobservable" problems. Classical planning systems typically have no observation model, but the fact that the initial state is known and operators are deterministic means that they can also be thought of as solving completely observable problems.

Completely observable and completely unobservable models are particularly clean but are unrealistic. The POMDP and C-BURIDAN frameworks model *partially* observable environments, in that observations provide some information about the underlying state, but not enough to guarantee that it will be known with certainty. This model provides for a great deal of expressiveness (both completely observable and completely unobservable models can be viewed as special cases), but is quite difficult to solve. It is an interesting and powerful model because it allows systems to reason about taking actions to gather knowledge that will be important for later decision making.

6.5 Objective

The job of a planner is to find a plan that satisfies a particular objective; most often, the objective is a goal of achievement, that is, to arrive at some state that is in a set of problem-specific goal states. When probabilistic information is available concerning the initial state and transitions, a more general objective can be used—reaching a goal state with sufficient probability (see, for example, work on BURIDAN and C-BURIDAN).

A popular alternative to goal attainment is maximizing total expected discounted reward (total-reward criterion). Under this objective, each action results in an immediate reward that is a function of the current state. The exponentially discounted sum of these rewards over the execution of a plan (finite or infinite horizon) constitutes the value of the plan. This objective is used extensively in most work with MDPs and POMDPs, including ours.

Several authors (for example, Koenig [27]) have pointed out that, given a completely observable problem stated as one of goal achievement, reward functions can be constructed so that a policy that maximizes reward can be used to maximize the probability of goal attainment in the original problem. This shows that the total-reward criterion is no less general than goal achievement in completely observable domains. The same holds for finite-horizon partially observable domains.

Interestingly, a more complicated transformation holds in the opposite direction: any total expected discounted reward problem (completely observable or finite horizon) can be transformed into a goal-achievement problem of similar size [12,69]. Roughly, the transformation simulates the discount factor by introducing an absorbing state with a small probability of being entered on each step. Rewards are then simulated by normalizing all reward values to be between zero and one and then "siphoning off" some of the probability of absorption equal to the amount of normalized reward. The (perhaps counterintuitive) conclusion is that goal-attainment problems and reward-type problems are computationally equivalent.

There is a qualitative difference in the kinds of problems typically addressed with POMDP models and those addressed with planning models. Quite frequently, POMDPs are used to model situations in which the agent is expected to go on behaving indefinitely, rather than simply until a goal is achieved. Given the inter-representability results between goal-probability problems and discounted-optimality problems, it is hard to make technical sense of this difference. In fact, many POMDP models should probably be addressed in an average-reward context [17]. Using a discounted-optimal policy in a truly infinite-duration setting is a convenient approximation, similar to the use of a situation-action mapping from a finite-horizon policy in receding horizon control.

Littman [35] catalogs some alternatives to the total-reward criterion, all of which are based on the idea that the objective value for a plan is based on a summary of immediate rewards over the duration of a run. Koenig and Simmons [28] examine risk-sensitive planning and showed how planners for the total-reward criterion could be used to optimize risk-sensitive behavior. Haddawy et al. [21] looked at a broad family of decision-theoretic objectives that make it possible to specify trade-offs between partially satisfying goals quickly and satisfying them completely. Bacchus, Boutilier, and Grove [2] show how some richer objectives based on evaluations of sequences of actions can actually be converted to total-reward problems. Other objectives considered in planning systems, aside from simple goals of achievement, include goals of maintenance and goals of prevention [15]; these types of goals can typically be represented using immediate rewards as well.

6.6 Representation of Problems

The propositional representations most often used in planning have a number of advantages over the flat state-space representations associated with MDPs and POMDPs. The main advantage comes from their compactness—just as with operator schemata, which can represent many individual actions in a single operator, propositional representations can be exponentially more concise than a fully expanded state-based transition matrix for an MDP.

Algorithms for manipulating compact (or factored) POMDPs have begun to appear [14,6]—this is a promising area for future research. At present, however, there is no evidence that these algorithms result in improved planning time significantly over the use of a "flat" representation of the state space.

6.7 Plan Structures

Planning systems differ in the structure of the plans they produce. It is important that a planner be able to express the optimal plan if one exists for a given domain. We briefly review some popular plan structures along with domains in which they are sufficient for expressing optimal behavior.

Traditional plans are simple sequences of actions. They are sufficient when the initial state is known and all actions are deterministic. A slightly more elaborate structure is the partially ordered plan (generated, for example, by SNLP and UCPOP), or the parallel plan [4]. In this type of plan, actions can be left unordered if all orderings are equivalent under the performance metric.

When actions are stochastic, partially ordered plans can still be used (as in BURIDAN), but contingent plans can be more effective. The simplest kind of contingent or branching plan is one that has a tree structure (as generated by CNLP or PLINTH). In such a plan, some of the actions have different possible outcomes that can be observed, and the flow of execution of the plan is conditioned on the outcome. Branching plans are sufficient for representing optimal plans for finite-horizon domains. Directed acyclic graphs (DAGs) can represent the same class of plans, but potentially do so much more succinctly, because separate branches can share structure. C-BURIDAN uses a representation of contingent plans that also allows for structure sharing (although of a different type than our DAG-structured plans). Our work on POMDPs finds DAG-structured plans for finite-horizon problems.

For infinite-horizon problems, it is necessary to introduce loops into the plan representation [57,31]. (Loops might also be useful in long finite-horizon POMDPs for representational succinctness.) A simple loop-based plan representation de-

picts a plan as a labeled directed graph. Each node of the graph is labeled with an action and there is one labeled outgoing edge for each possible outcome of the action. It is possible to generate this type of *plan graph* for some POMDPs [47,59,8,22,23].

For completely observable problems with a high branching factor, a more convenient representation is a *policy*, which maps the current state (situation) to a choice of action. Because there is an action choice specified for all possible initial states, policies are also called universal plans [53]. This representation is not appropriate for POMDPs, since the underlying state is not fully observable. However, POMDP policies can be viewed as universal plans over belief space.

It is interesting to note that there are infinite-horizon POMDPs for which no finite-state plan is sufficient. Simple 2-state examples can be constructed for which optimal behavior requires counting (i.e., a simple stack machine); there is reason to believe that general pushdown automata and perhaps even Turing machines are necessary to represent optimal plans in general. This argues that, in the limit, a plan is actually a program. Several techniques have been proposed recently for searching for good program-like controllers in POMDPs [68,29]. We restrict our attention to the simpler finite-horizon case and a small set of infinite-horizon problems that have optimal finite-state plans.

7 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 [34] suggest that 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 [33]. 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 [50,60] 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 [11] and McCallum [40,41].

A Appendix

Theorem 1 *Let U_a be a non-empty set of useful policy trees, and $\mathcal{Q}_t^a$ be the complete set of useful policy trees. Then $U_a \neq \mathcal{Q}_t^a$ if and only if there is some tree $p \in U_a$, observation $o^* \in \Omega$, and subtree $p' \in \mathcal{V}_{t-1}$ for which there is some belief state b such that*

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

for all $\tilde{p} \in U_a$, where p_{new} is a t-step policy tree that agrees with p in its action and all its subtrees except for observation o^*, for which $o^*(p_{\text{new}}) = p'$.

Note that we are defining two trees to be equal if they have the same value function; this makes it unnecessary to deal with the effect of ties in the set U_a.

PROOF. The "if" direction is easy since the b can be used to identify a policy tree missing from U_a.

The "only if" direction can be rephrased as: If $U_a \neq \mathcal{Q}_t^a$ then there is a belief state b, a $p \in U_a$, and a p_{new} such that p_{new} has a larger value than any other $\tilde{p} \in U_a$ at b.

Start by picking some $p^* \in \mathcal{Q}_t^a - U_a$ and choose any b such that p^* has the highest value at b (there must be such a b since p^* is useful). Let

$$p = \operatorname*{argmax}_{p' \in U_a} V_{p'}(b).$$

Now, p^* is the policy tree in $\mathcal{Q}_t^a - U_a$ that has the highest value at b, and p is the policy tree in U_a that has the highest value at b. By construction, $V_{p^*}(b) > V_p(b)$.

Now, if p and p^* differ in only one subtree, then we are done, p^* can serve as a p_{new} in the theorem.

If p and p^* differ in more than one subtree, we will identify another policy tree that can act as p_{new}. Choose an observation $o^* \in \Omega$ such that

$$\sum_s b(s) \left(\sum_{s' \in \mathcal{S}} T(s, a(p^*), s') V_{o^*(p^*)}(s') \right) > \sum_s b(s) \left(\sum_{s' \in \mathcal{S}} T(s, a(p^*), s') V_{o^*(p)}(s') \right) \quad .$$

There must be a o^* satisfying this inequality since otherwise we get the contradiction

$$\begin{aligned}
& V_{p^*}(b) \\
&= \sum_s b(s) \left(R(s, a(p^*)) + \gamma \sum_{s' \in \mathcal{S}} T(s, a(p^*), s') \sum_{o_i \in \Omega} O(s', a(p^*), o_i) V_{o_i(p^*)}(s') \right) \\
&\leq \sum_s b(s) \left(R(s, a(p)) + \gamma \sum_{s' \in \mathcal{S}} T(s, a(p), s') \sum_{o_i \in \Omega} O(s', a(p), o_i) V_{o_i(p)}(s') \right) \\
&= V_p(b) \quad .
\end{aligned}$$

Define p_{new} to be identical to p except that in the place of subtree $o^*(p)$, we put $o^*(p^*)$. From this, it follows that

$$\begin{aligned}
& V_{p_{\text{new}}}(b) \\
&= \sum_s b(s) \left(R(s, a(p_{\text{new}})) + \gamma \sum_{s' \in \mathcal{S}} T(s, a(p_{\text{new}}), s') \sum_{o_i \in \Omega} O(s', a(p_{\text{new}}), o_i) V_{o_i(p_{\text{new}})}(s') \right) \\
&> \sum_s b(s) \left(R(s, a(p)) + \gamma \sum_{s' \in \mathcal{S}} T(s, a(p), s') \sum_{o_i \in \Omega} O(s', a(p), o_i) V_{o_i(p)}(s') \right) \\
&= V_p(b) \\
&\geq V_{\tilde{p}}(b)
\end{aligned}$$

for all $\tilde{p} \in U_a$. Therefore, the policy trees p and p_{new}, the observation o^*, $p' = o^*(p^*)$, and the belief state b satisfy the conditions of the theorem.

References

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

[2] Fahiem Bacchus, Craig Boutilier, and Adam Grove. Rewarding behaviors. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 1160–1167. AAAI Press/The MIT Press, 1996.

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

[4] Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1–2):279–298, 1997.

[5] Jim Blythe. Planning with external events. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 94–101, 1994.

[6] Craig Boutilier and David Poole. Computing optimal policies for partially observable decision processes using compact representations. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 1168–1175. AAAI Press/The MIT Press, 1996.

[7] Anthony Cassandra, Michael L. Littman, and Nevin L. Zhang. Incremental Pruning: A simple, fast, exact method for partially observable Markov decision processes. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–97)*, pages 54–61, San Francisco, CA, 1997. Morgan Kaufmann Publishers.

[8] 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*, pages 1023–1028, Seattle, WA, 1994.

[9] Anthony Rocco Cassandra. *Exact and Approximate Algorithms for Partially Observable Markov Decision Problems.* PhD thesis, Department of Computer Science, Brown University, May 1998.

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

[11] 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.

[12] Anne Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203–224, February 1992.

[13] Thomas Dean, Leslie Pack Kaelbling, Jak Kirman, and Ann Nicholson. Planning under time constraints in stochastic domains. *Artificial Intelligence*, 76(1-2):35–74, 1995.

[14] Denise Draper, Steve Hanks, and Dan Weld. Probabilistic planning with information gathering and contingent execution. Technical Report 93-12-04, University of Washington, Seattle, WA, December 1993.

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

[16] James N. Eagle. The optimal search for a moving target when the search path is constrained. *Operations Research*, 32(5):1107–1115, 1984.

[17] Emmanuel Fernández-Gaucherand, Aristotle Arapostathis, and Steven I. Marcus. On the average cost optimality equation and the structure of optimal policies for partially observable Markov processes. *Annals of Operations Research*, 29:471–512, 1991.

[18] Robert P. Goldman and Mark S. Boddy. Conditional linear planning. In Kristian Hammond, editor, *The Second International Conference on Artificial Intelligence Planning Systems*, pages 80–85. The AAAI Press / The MIT Press, 1994.

[19] Robert P. Goldman and Mark S. Boddy. Epsilon-safe planning. In *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence (UAI94)*, pages 253–261, Seattle, WA, 1994.

[20] Robert P. Goldman and Mark S. Boddy. Representing uncertainty in simple planners. In *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning (KR94)*, pages 238–245, 1994.

[21] Peter Haddawy and Steve Hanks. Utility models for goal-directed decision-theoretic planners. Technical Report 93-06-04, Department of Computer Science and Engineering, University of Washington, June 1993.

[22] Eric A. Hansen. Cost-effective sensing during plan execution. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 1029–1035. AAAI Press/The MIT Press, 1994.

[23] Eric A. Hansen. An improved policy iteration algorithm for partially observable MDPs. In *Advances in Neural Information Processing Systems 10*, 1998. In press.

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

[25] Ronald A. Howard. Information value theory. *IEEE Transactions on Systems Science and Cybernetics*, SSC-2(1):22–26, August 1966.

[26] 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, March 1960.

[27] Sven Koenig. Optimal probabilistic and decision-theoretic planning using Markovian decision theory. Technical Report UCB/CSD 92/685, Berkeley, May 1992.

[28] Sven Koenig and Reid G. Simmons. Risk-sensitive planning with probabilistic decision graphs. In *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning*, pages 363–373, 1994.

[29] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press, 1992.

[30] Nicholas Kushmerick, Steve Hanks, and Daniel S. Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, 76(1-2):239–286, September 1995.

[31] Shieu-Hong Lin and Thomas Dean. Generating optimal policies for high-level plans with conditional branches and loops. In *Proceedings of the Third European Workshop on Planning*, pages 205–218, 1995.

[32] Michael L. Littman. Memoryless policies: Theoretical limitations and practical results. In Dave Cliff, Philip Husbands, Jean-Arcady Meyer, and Stewart W. Wilson, editors, *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, Cambridge, MA, 1994. The MIT Press.

[33] Michael L. Littman, Anthony R. Cassandra, and Leslie Pack Kaelbling. Learning policies for partially observable environments: Scaling up. In Armand Prieditis and Stuart Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning*, pages 362–370. Morgan Kaufmann, San Francisco, CA, 1995. Reprinted in *Readings in Agents*, M. H. Huhns and M. P. Singh, eds., Morgan Kaufmann, 1998.

[34] Michael L. Littman, Anthony R. Cassandra, and Leslie Pack Kaelbling. Efficient dynamic-programming updates in partially observable Markov decision processes. Technical Report CS-95-19, Brown University, Providence, RI, 1996.

[35] Michael Lederman Littman. *Algorithms for Sequential Decision Making.* PhD thesis, Department of Computer Science, Brown University, February 1996. Also Technical Report CS-96-09.

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

[37] Stephen M. Majercik and Michael L. Littman. MAXPLAN: A new approach to probabilistic planning. Technical Report CS-1998-01, Department of Computer Science, Duke University, 1998. Submitted for review.

[38] T. M. Mansell. A method for planning given uncertain and incomplete information. In *Proceedings of the 9th Conference on Uncertainty in Artificial Intelligence*, pages 350–358. Morgan Kaufmann Publishers, July 1993.

[39] David McAllester and David Rosenblitt. Systematic nonlinear planning. In *Proceedings of the 9th National Conference on Artificial Intelligence*, pages 634–639, 1991.

[40] R. Andrew McCallum. Overcoming incomplete perception with utile distinction memory. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 190–196, Amherst, Massachusetts, 1993. Morgan Kaufmann.

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

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

[43] 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*, pages 319–358. Ablex Publishing Company, Norwood, New Jersey, 1985.

[44] L. Morgenstern. Knowledge preconditions for actions and plans. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, pages 867–874, 1987.

[45] J. S. Penberthy and D. Weld. UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of the third international conference on principles of knowledge representation and reasoning*, pages 103–114, 1992.

[46] Mark A. Peot and David E. Smith. Conditional nonlinear planning. In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, pages 189–197, 1992.

[47] Loren K. Platzman. A feasible computational approach to infinite-horizon partially-observed Markov decision problems. Technical report, Georgia Institute of Technology, Atlanta, GA, January 1981.

[48] Louise Pryor and Gregg Collins. Planning for contingencies: A decision-based approach. *Journal of Artificial Intelligence Research*, 4:287–339, 1996.

[49] Martin L. Puterman. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, 1994.

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

[51] 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.

[52] R. B. Scherl and H. J. Levesque. The frame problem and knowledge-producing actions. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 689–697, 1993.

[53] Marcel J. Schoppers. Universal plans for reactive robots in unpredictable environments. In *Proceedings of the International Joint Conference on Artificial Intelligence 10*, pages 1039–1046, 1987.

[54] Alexander Schrijver. *Theory of Linear and Integer Programming*. Wiley-Interscience, New York, NY, 1986.

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

[56] 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.

[57] David E. Smith and Mike Williamson. Representation and evaluation of plans with loops. Working notes for the 1995 Stanford Spring Symposium on Extended Theories of Action, 1995.

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

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

[60] 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, San Mateo, California, 1993. Morgan Kaufmann.

[61] Jonathan Tash and Stuart Russell. Control strategies for a stochastic planner. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 1079–1085, 1994.

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

[63] C. C. White and D. Harrington. Application of Jensen's inequality for adaptive suboptimal design. *Journal of Optimization Theory and Applications*, 32(1):89–99, 1980.

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

[65] Chelsea C. White, III and William T. Scherer. Solution procedures for partially observed Markov decision processes. *Operations Research*, 37(5):791–797, September-October 1989.

[66] Ronald J. Williams and Leemon C. Baird, III. Tight performance bounds on greedy policies based on imperfect value functions. Technical Report NU-CCS-93-14, Northeastern University, College of Computer Science, Boston, MA, November 1993.

[67] Nevin L. Zhang and Wenju Liu. Planning in stochastic domains: Problem characteristics and approximation. Technical Report HKUST-CS96-31, Department of Computer Science, Hong Kong University of Science and Technology, 1996.

[68] Jieyu Zhao and Jürgen H. Schmidhuber. Incremental self-improvement for life-time multi-agent reinforcement learning. In Pattie Maes, Maja J. Mataric, Jean-Arcady Meyer, Jordan Pollack, and Stewart W. Wilson, editors, *From Animals to Animats: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pages 516–525. The MIT Press, 1996.

[69] Uri Zwick and Mike Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1–2):343–359, 1996.