Computing Optimal Policies for Partially Observable Decision Processes using Compact Representations

Craig Boutilier and David Poole
Department of Computer Science
University of British Columbia
Vancouver, BC V6T 1Z4, CANADA
cebly@cs.ubc.ca, poole@cs.ubc.ca

Abstract

Partially-observable Markov decision processes provide a general model for decision theoretic planning problems, allowing trade-offs between various courses of actions to be determined under conditions of uncertainty, and incorporating partial observations made by an agent. Dynamic programming algorithms based on the *belief state* of an agent can be used to construct optimal policies without explicit consideration of past history, but at high computational cost. In this paper, we discuss how structured representations of system dynamics can be incorporated in classic POMDP solution algorithms. We use Bayesian networks with structured conditional probability matrices to represent POMDPs, and use this model to structure the belief space for POMDP algorithms, allowing irrelevant distinctions to be ignored. Apart from speeding up optimal policy construction, we suggest that such representations can be exploited in the development of useful approximation methods.

1 Introduction

Recent interest in *decision-theoretic planning* (DTP) has been spurred by the need to extend planning algorithms to deal with quantified uncertainty regarding an agent's knowledge of the world and action effects, as well as competing objectives [9, 7, 4, 16] (see [2] for a brief survey). A useful underlying semantic model for such DTP problems is that of *partially observable Markov decision processes* (POMDPs) [6]. This model, used in operations research [17, 12] and stochastic control, accounts for the tradeoffs between competing objectives, action costs, uncertainty of action effects and observations that provide incomplete information about the world. However, while the model is very general, these problems are typically specified in terms of state transitions and observations associated with individual states—even specifying a problem in these terms is problematic given that the state space grows exponentially with the number of variables used to describe the problem.

Influence diagrams (IDs) and Bayesian networks (BNs) [10, 14] provide a much more natural way of specifying the dynamics of a system, including the effects of actions and observation probabilities, by exploiting problem structure and independencies among random variables. As such, problems can be specified much more compactly and naturally [8, 4, 16]. In addition, algorithms for solving IDs can exploit such regularities for computational gain in decision-making. Classic solution methods for POMDPs within the OR community, in contrast, have been developed primarily using explicit state-based representations which adds a sometimes unwanted computational burden. However, unlike ID algorithms, for which policies grow exponentially with the time horizon, POMDP algorithms offer concepts (in particular, that of *belief state*) that sometimes alleviate this difficulty.

In this paper we propose a method for optimal policy construction, based on standard POMDP algorithms, that exploits BN representations of actions and reward, as well as tree [4] or rule [16] representations within the BN itself. In this way, our technique exploits the advantages of classic POMDP and ID representations and provides leverage for approximation methods.

In Section 2, we define POMDPs and associated notions, at the same time showing how structured representations, based on BNs (augmented with *tree-structured conditional probability tables*), can be used to specify POMDPs. In Section 3, we describe a particular POMDP algorithm due to Monahan [12], based on the work of Sondik [17]. In Section 4, we describe how we can incorporate the structure captured by our representations to reduce the effective state space of the Monahan algorithm at any point in its computation. Our algorithm exploits ideas from the SPI algorithm of [4] for fully observable processes. In Section 5 we suggest that our method may enable good approximation schemes for POMDPs.

2 POMDPs and Structured Representations

In this section we build upon the classic presentation of POMDPs adopted in much of the OR community. We refer to [17, 11, 6] for further details and [12, 5] for a survey. We describe the main components of POMDPs and related concepts. However, by assuming that problems are specified in terms or propositional (or other random) variables, we are able to describe how structured representations, in particular, decision trees or if-then rules, can be used to describe these components compactly. We begin with a (running) example.

Example Imagine a robot that can check whether a user wants coffee and can get it by going to the shop across the

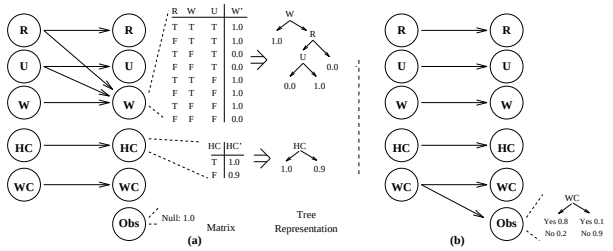

Figure 1: Action Networks for (a) GetC and (b) TestC

street. The robot is rewarded if the user wants coffee *WC* and has coffee *HC*, but is penalized if *HC* is false when *WC* is true. The robot will also get wet W if it is raining R when it goes for coffee, unless it has its umbrella U. We can imagine a number of other tasks here as well. Although the robot can check on the weather, grab its umbrella, etc., we focus on two actions: getting coffee *GetC* and checking whether the user wants coffee by means of a quick inspection *TestC*.

2.1 System Dynamics

We assume a finite set of propositions $\mathcal{P}$ that describe all relevant aspects of the system we wish to control. This induces a finite state space $\mathcal{S} = \mathbf{2}^{\mathcal{P}}$ consisting of all possible assignments of truth values to $\mathcal{P}$. There is a finite set of actions $\mathcal{A}$ available to the agent or controller, with each action causing a state transition. We assume the system can be modeled as a POMDP with a stationary dynamics (i.e., the effects of actions do not depend on the stage of the process). For simplicity we assume all actions can be taken (or attempted) at all states. While an action takes an agent from one state to another, the effects of actions cannot be predicted with certainty; hence (slightly abusing notation) we write $Pr(s_2|s_1, a)$ to denote the probability that s_2 is reached given that action a is performed in state s_1. This formulation assumes the Markov property for the system in question.

One can represent the transition probabilities associated with action a explicitly using a $|\mathcal{S}| \times |\mathcal{S}|$ probability matrix. However, the fact that $|\mathcal{S}|$ increases exponentially with the number of problem characteristics $|\mathcal{P}|$ generally requires more compact representation; thus we represent an action's effects using a "two-slice" (temporal) Bayes net [8]: we have one set of nodes representing the state prior to the action (one node for each variable P), another set representing the state after the action has been performed, and directed arcs representing causal influences between these sets (see Figure 1). We require that the induced graph be acyclic. For simplicity we assume also that arcs are directed only from pre-action to post-action nodes.[1] See [8, 4] for details.

The post-action nodes have the usual conditional probability tables (CPTs) describing the probability of their values given the values of their parents, under action a. We assume that these CPTs are represented using a decision tree, as in [4] (or if-then rules as in [15]). These are essentially compact function representations that exploit regularities in the CPTs. We will exploit the compactness and structure of such representations when producing optimal policies. We denote the tree for variable P under action a by *Tree*$(P'|a)$.[2]

Example Figure 1(a) illustrates the network for action *GetC*. The network structure shows, for instance, that the truth of W', whether the robot is wet after performing *GetC*, depends on the values of R, U and W prior to the action. The matrix for W' quantifies this dependence; and *Tree*$(W'|$*GetC*$)$ illustrates the more compact representation (the leaf nodes indicate the probability of W' after *GetC* given the conditions labeling its branch: left arcs denote true and right arcs false). We elaborate on the *Obs* variable below.

2.2 Observations

Since the system is partially observable, the planning agent may not be able to observe its exact state, introducing another source of uncertainty into action selection. However, we assume a set of possible *observations* $\mathcal{O}$ that provide evidence for the true nature of (various aspects of) the state. In general, the observation at any stage will depend stochastically on the state, the action performed and its outcome.

We assume a family of distributions over observations, For each s_i, s_j, a_k such that $Pr(s_j|s_i, a_k) > 0$, let $Pr(o_l|s_i, a_k, s_j)$ denote the probability of observing o_l when action a_k is executed at state s_i and results in state s_j. (As a special case, a fully observable system can be modeled by assuming $\mathcal{O} = \mathcal{S}$ and $Pr(o_l|s_i, a_k, s_j) = 1$ iff $o_l = s_j$.) We assume for simplicity that the observation probability depends only on the action and starting state, not the resulting state; that is, $Pr(o_l|s_i, a_k, s_h) = Pr(o_l|s_j, a_k, s_h)$ for each s_i, s_j.[3]

To represent observation probabilities compactly, we add a distinguished variable Obs to each action network that represents the observations possible after performing that action. We use *Obs*(a) to denote the set of possible observations given a.[4] The variables that influence the observation are indicated by directed arcs, and this effect is described, as above, using a decision tree. We note that complex observations may also be factored into distinct observation variables (e.g., should the agent get information pertaining to propositions P and Q by performing one action, two distinct variables Obs_1 and Obs_2 might be used); we ignore this possibility here.

[1] We often denote post-action variables by P' instead of P to prevent confusion. Causal influences between post-action variables should be viewed as *ramifications* and will complicate our algorithm slightly, but only in minor detail.

[2] The network structure is not strictly necessary: the parent of a post-action node can be determined from its CPT or decision tree (see, e.g., Poole's [15] rule-based representation of Bayes nets).

[3] This is a natural assumption for information-gathering actions, but others are possible; e.g., Sondik's [17] original presentation of POMDPs assumes the observation depends only on the resulting state. This assumption makes our algorithm somewhat simpler to describe; but it can generalized (see Section 4).

[4] These are similar to observation variables in influence diagrams [10]; however, there are no emanating information arcs.

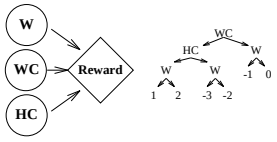

Figure 2: Reward Function Network

Example The variable *Obs* in Figure 1(a) takes on a single value (*Null*), obtained with certainty when *GetC* is executed (i.e., the action provides no feedback). More interesting is the action *TestC* shown in Figure 1(b). Although it has no effect on the state variables (we assume persistence), it is useful as an *information gathering action*: the value of the variable *Obs* (*Yes* or *No*) is strongly dependent on whether the user wants coffee. Should the value *Yes* be observed, our robot may be quite confident the user does, in fact, want coffee (see below).

2.3 Rewards

The final component needed to describe a POMDP is a real-valued *reward function* R that associates rewards or penalties with various states: $R(s)$ denotes the relative goodness of being in state s. We also assume a *cost function* $C(a, s)$ denoting the cost of taking action a in state s. The reward (cost) function can be represented in a structured fashion using a value node and decision tree describing the influence of various combinations of variables on rewards (as with tree-structured CPTs). Leaves of the tree represent the reward associated with the states consistent with the labeling of the corresponding branch.

Example Figure 2 shows the reward function for our problem, indicating that the reward for a particular state is influenced only by the truth of the propositions W, *WC* and *HC*. A similar representation for action cost can be used. In this example action costs are constant: a cost of 1.0 for *GetC* and 0.5 for *TestC* is assumed.

The sets of actions, states and observations, the associated transition and observation probabilities, and the reward and cost functions, make up a POMDP. We now turn our attention to the various concepts used in decision-making.

2.4 Policies

We focus on *finite-horizon problems* here: given a horizon of size n an agent executes n actions at stages 0 through $n-1$ of the process, ending up in a terminal state at stage n. The agent receives reward $R(s)$ for each state s passed through at stages 0 through n (its trajectory). A plan or *policy* is a function that determines the choice of action at any stage of the system's evolution. The *value* of a policy is the expected sum of rewards accumulated (incorporating both action costs and state rewards and penalties). A policy is optimal if no other policy has larger value.

In choosing the action to perform at stage k of the process, the agent can rely only on its knowledge of the initial state s_0 (whether it knows the state exactly, or had an initial distribution over states), and the history of actions it performed and observations it received prior to stage k. Different action-observation histories can lead an agent to choose different actions. Thus, a policy can be represented as a mapping from any initial state estimate, and k-stage history, to the action for stage $k+1$. This is roughly the approach adopted by solution techniques for IDs [10]. However, an elegant way to treat this problem is to maintain a current *belief state*, and treat policies as mapping over from belief states to actions.

2.5 Belief States

A *belief state* $\pi \in \Delta(S)$ is a probability distribution over states. The probability π_i assigned to state s_i by π is the degree of belief that the true (current) state of the system is s_i.

Given some state of belief π^k estimating the system state at stage k of the decision process, we can update our belief state based on the action a^k taken and observation o^k made at stage k to form a new belief state π^{k+1} characterizing the state of the system at stage $k+1$. Once we have π^{k+1} in hand, the fact that a^k, o^k and π^k gave rise to it can be forgotten. We use $T(\pi, a, o)$ to denote the *transformation* of the belief state π given that action a is performed and observation o is made: it is defined as

$$T(\pi, a, o)_i = \frac{\sum_{s_j \in \mathcal{S}} Pr(o|s_j, a, s_i) Pr(s_i|s_j, a) \pi_j}{\sum_{s_j, s_k \in \mathcal{S}} Pr(o|s_j, a, s_k) Pr(s_k|s_j, a) \pi_j}$$

$T(\pi, a, o)_i$ denotes the probability that the system is in state i once a, o are made, given prior belief state π.

The new belief state $T(\pi, a, o)$ summarizes all information necessary for subsequent decisions, accounting for all past observations, actions and their influence on the agent's estimate of the system state. This is the essential assumption behind classical POMDP techniques: at any stage of the decision process, assuming π^k accurately summarizes past actions and observations, the optimal decision can be based solely on π^k — history (now summarized) can be ignored [17]. Intuitively, we can think of this as converting a partially observable MDP over the original state space $\mathcal{S}$ into a *fully observable* MDP over the *belief space* $\mathcal{B}$ (the set of belief states π).

A belief state may be represented using a vector of $|\mathcal{S}|$ probabilities; but structured representations are possible. We do not pursue these here, since most POMDP solution algorithms do not use a belief state to construct a policy.

2.6 Value Functions

State Value Functions: A *state value function* $VS : \mathcal{S} \to \mathbb{R}$ associates a value $VS(s)$ with each state s. This reflects the expected sum of future rewards the agent will receive, assuming some fixed policy or sequence of actions in the future. In addition, a *state Q-function* $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ denotes the value $Q(s, a)$ of performing an action a in state s, assuming future value is dictated by a fixed course of action [18]. In particular, let VS^k and Q^k be the *k-stage-to-go* value and Q-functions. If the function VS^{k-1} is known, then Bellman's [1] optimality equation ensures that

$$Q^k(s_i, a) = C(a, s_i) + R(s_i) + \sum_{s_j \in \mathcal{S}} Pr(s_j|s_i, a) VS^{k-1}(s_j) \quad (1)$$

$$VS^k(s_i) = \max_{a \in \mathcal{A}} \{Q^k(s_i, a)\} \quad (2)$$

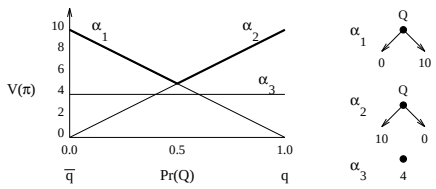

Figure 3: Piecewise Linear, Convex Value Function

Intuitively, once the agent has determined a course of action for the last $k-1$ stages of the process (giving rise to VS^{k-1}), Equation 1 determines the value of executing action a at any state. In the case of fully observable MDPs, this forms the basis of a dynamic programming algorithm that can be used to optimize the choice of action according to Equation 2.

We can represent value and Q-functions using decision trees in precisely the same manner as reward functions (e.g., Figure 2). Figure 5 illustrates just such value and Q-trees. In fact, as we will see below, we can apply these equations directly to such structured representations.

Belief State Value Functions: Unfortunately, in the case of POMDPs, determining the best action for individual states is not often helpful, for the agent will typically not know the exact state. However, the assignment of value to states via value and Q-functions can also be viewed as an assignment of value to belief states. In particular, any state value function *VS* induces a value function over belief states:

$$\alpha(\pi) = \pi \cdot VS = \sum_{s_i \in \mathcal{S}} \pi_i VS(s_i)$$

Following Monahan [12] we call these α-*functions*. The value of a belief state is the weighted sum of the individual state values; thus, such α-functions our linear functions over belief states. Q-functions can be applied similarly to belief states. Finally, we note that a value tree or Q-tree can be used to represent a linear value function over belief states; when interpreted this way, we call these α-trees. In the sequel, we assume that α-functions are represented by α-trees.

In determining optimal policies for POMDPs, we need to represent the optimal (k-stage-to-go) value functions $V : \Delta(\mathcal{S}) \rightarrow \mathbb{R}$ for belief states. Clearly, α-functions, being linear, are quite restrictive in expressiveness. However, a key observation of Sondik [17] is that optimal value functions are *piecewise linear and convex (p.l.c.)* over the belief space. In other words, we can represent the optimal (k-stage-to-go) value function for any POMDP as a set $\aleph$ of α-functions, with

$$V(\pi) = \max\{\alpha(\pi) : \alpha \in \aleph\}$$

(We will see exactly why this is so in the next section.)

As a graphical illustration of this p.l.c. representation, consider Figure 3. Assume a single proposition Q (two states q and $\overline{q}$) and the three α-functions $\alpha_1, \alpha_2, \alpha_3$, all represented as trees. Each α-tree determines a linear value function for any belief state (e.g., α_1 takes its highest value at belief state $\pi(q) = 0; \pi(\overline{q}) = 1$). The set $\{\alpha_1, \alpha_2, \alpha_3\}$ corresponds to the p.l.c. value function indicated by the thick line.

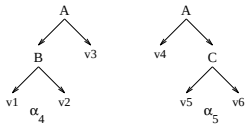

Figure 4: Structured Domination Testing

Dominated α-functions: Finally, we note that certain elements of a set $\aleph$ of α-functions may contribute nothing to the induced p.l.c. value function, namely, those elements that are *stochastically dominated*. For instance, α_3 in Figure 3 is dominated by one of α_1 or α_2 at all points in the belief space. Monahan [12] suggests that such dominated elements be detected by means of a simple linear program and eliminated from $\aleph$ (see also [5]). Once again, the use of α-trees can in many cases considerably reduce the size of these LPs, which normally involve variables for each state. For example, to consider whether the tree α_4 dominates α_5, as shown in Figure 4, the required LP need only have variables corresponding to the propositions AB, $A\overline{B}$, $\overline{A}C$ and $\overline{AC}$, rather than $|\mathcal{S}|$ variables.

3 Computation of Optimal Policies

We now describe how to use the ideas above to to determine optimal policies for POMDPs. We begin by presenting the intuitions underlying Monahan's [12] variant of Sondik's [17] algorithm, and how the p.l.c. nature of value functions is exploited. We describe how our compact tree representations can be exploited in the next section.

Given a POMDP, we want to determine a policy that selects, for any belief state π, and $k > 0$ within the problem horizon, the optimal action to be performed. Intuitively, $Pol(\pi, k) \in \mathcal{A}$ is the best action available to the agent assuming its state of belief is π and there are k stages of the process remaining. Unfortunately, representing such a function can be problematic, since the set of belief states $\mathcal{B}$ is a $|\mathcal{S}|$-dimensional continuous space. However, Sondik's key observation that k-stage-to-go value functions are p.l.c., and thus finitely representable, also provides a means to finitely represent policies (albeit indirectly). Intuitively, the determination of the "pieces" of the the k-stage-to-go value function will attach actions to each of these pieces. To determine the best action to be performed for a given belief state π, the action associated with the "maximal piece" of the value function for π will be the best action. Thus, actions are associated with various *regions* of the belief space, regions determined by the value function itself.

To see this, we first note that with zero stages-to-go the agent has no action choice to make, and the expected value of being in any belief state is given by the α-function determined by immediate reward R; that is, $V^0(\pi) = \pi \cdot R$. Thus, V^0 is a linear function of π. We call this single α-function α^0.

The computation of V^1 depends only on V^0 and illustrates why the value functions remain p.l.c. The value of performing any action a in a given *state* s is given by $Q(a, s)$, as defined in Equation 1, using R (or V^0) as the terminal value.

Since the agent has no choice of action at stage 0, any observations it makes subsequent to performing this action can have no influence on its behavior or the expected value of the action. Hence, this Q-function can be interpreted as an α-function (say α_a^1) over belief states in the obvious way. The *value* of π with one stage remaining requires that we choose an action a that has maximal Q-value; in other words,

$$V^1(\pi) = \max\{\alpha_a^1(\pi) : a \in \mathcal{A}\}$$

However, since each of the α_a^1 is linear, V^1 is p.l.c. and has a finite representation — the set of α^1-functions themselves. We dub this set $\aleph^1$.

It is worth noting that the optimal action choice given π with one stage-to-go, while not represented explicitly, is easily determined from $\aleph^1$: if α_a^1 is the member of $\aleph^1$ that maximizes $\alpha_a^1(\pi)$, then a is the best action choice. For any $\alpha_a^k \in \aleph^k$, we say a is the *action associated with* α_a^k. In this algorithm, a policy is represented implicitly in this way.

Determining V^2 requires that we take observations into account. To begin, we note that to determine the value of action a with 2 stages-to-go, we allow for the fact that the action b chosen with 1 stage-to-go (and therefore the function α_b^1 representing future value) can depend on the observation o made after a. This dependence is accounted for using *observational strategies*: the action chosen with k-stages-to-go can depend on the observation made following the execution of action a with $k+1$ stages-to-go. Specifically, given a set $\aleph^k$ of α-functions denoting possible future value, an *observational strategy* is a function $OS : \mathcal{A} \times Obs \rightarrow \aleph^k$. For any $o \in Obs(a)$, we use $\alpha_{a,o}^k$ to denote $OS(a,o)$. We write OS_a to denote the restriction of OS to a particular action a.

For a given action a, the value of performing a with $k+1$ stages-to-go, given an observational strategy OS_a, is given by

$$\begin{aligned} & Q_{OS}(a, s_i) \\ &= C(a, s_i) + R(s_i) + \sum_{o \in Obs(a)} \sum_{s_j \in \mathcal{S}} Pr(s_j | o, s_i, a) \alpha_{a,o}^k(s_j) \\ &= \sum_{o \in Obs(a)} Pr(o | s_i, a) Q_{\alpha_{a,o}^k}(a, s_i) \end{aligned} \quad (3)$$

where $Q_{\alpha_{a,o}^k}$ is the Q-function given by Equation 1, using $\alpha_{a,o}^k$ as the terminal value function. Each Q_{OS} also determines an α-function over belief states. From this we derive the true Q-function, by maximizing over observational strategies:

$$Q(a, s) = \max_{OS_a} \{Q_{OS_a}(a, s)\}$$

The state value function is determined using the Q-functions by Equation 2, leaving us with a definition of the $k+1$ stage value function over belief states:

$$V^{k+1}(\pi) = \pi \cdot V = \sum_{s_i} \pi_i \cdot \max_a \max_{OS_a} \{Q_{OS_a}(a, s)\}$$

Thus, V^{k+1} is p.l.c. and can be represented by the set of α-functions $\{Q_{OS_a} : a \in \mathcal{A}\}$. We let $\aleph^{k+1}$ be this set of α-functions, defined in terms of $\aleph_k$ (since each OS_a maps an observation into an element of $\aleph_k$).

This gives the basic intuitions underlying Monahan's variant of Sondik's algorithm. To determine the set of k-stage-to-go value functions V^k for $k \leq n$, where n is some finite horizon, we simply iterate the following algorithm for n steps:

1. Let $\aleph^0 = \{R\}$
2. For $0 \leq k < n$, compute $\aleph^{k+1} = \{Q_{OS_a} : a \in \mathcal{A}\}$

As mentioned above, the optimal action choice at stage k for any π is determined by the computing the $\alpha_a^k \in \aleph_k$ that maximizes value of π and adopting the the action associated with α^k. We emphasize that the policy is not explicitly represented.

Generally, many of the generated α-functions in $\aleph_k$ will be irrelevant: they never influence the optimal policy because they are dominated by other elements of $\aleph_k$. Monahan's algorithm includes a pruning phase at each iteration that removes dominated components from $\aleph_k$ (see Section 2.6).

4 Structured Computation of α-Functions

We now turn our attention to using the structured representations described in Section 2 in Monahan's algorithm. The aim is to obviate the need to compute and represent the values of each state—each coefficient in the α-functions—individually. Beginning with a tree representation of the reward and cost functions, we use the tree-structured representation of CPTs in our action and observation descriptions to ensure that as much structure as possible is preserved in the generation of the elements of $\aleph_{k+1}$ from $\aleph_k$. Thus, we treat each $\aleph_k$ as a collection of α-*trees*, and show in two steps how to generate a the set of trees $\aleph_{k+1}$ from the set $\aleph_k$.

4.1 Generation of a Single α-tree for an action

The generation of an α-tree in $\aleph_{k+1}$, using a particular action a and strategy OS_a, requires we compute the function Q_{OS_a}, given by Equation 3, in structured form. We note that this computation naturally breaks into two parts: first, we compute the function $Q_{\alpha_{a,o}^k}$ for the individual observations $o \in Obs(a)$; and then we piece them together, taking the sum of the $Q_{\alpha_{a,o}^k}$-functions, weighted by the probability of observing o. We focus on the construction of $Q_{\alpha_{a,o}^k}$ first.

The function $Q_{\alpha_{a,o}^k}$ describes the value of performing a fixed action a at a state s, assuming the value of subsequent states is given by $\alpha_{a,o}$. For clarity, we let α denote the α-tree for $\alpha_{a,o}$ and Q_a denote the tree-structured function $Q_{\alpha_{a,o}^k}$ we wish to construct (i.e., a and o are fixed). Our method for generating the new Q-tree exploits the ideas described in [4], and is closely related to [15] (we refer to [4] for further details). Roughly, given a structured value function α, the conditions under which two states can have different expected future value given by α (under action a) can be easily determined by appeal to the action network for a. In particular, although an action may have different effects at two states, if the differences pertain only to variables (or variable assignments) that are *not relevant to the value function* α,

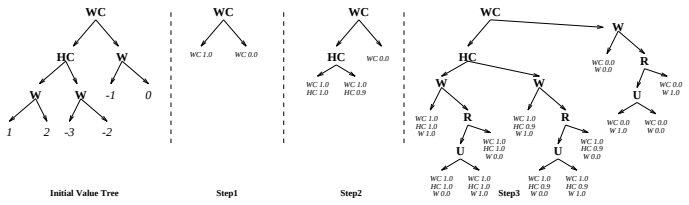

Figure 5: Generating Explanation of Future Value

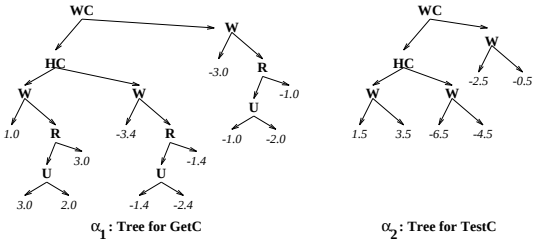

Figure 6: α-trees with 1 Stage-to-go

then those states have identical expected future value and need not be distinguished in the function Q_a. We construct the tree α^k so that only these relevant distinctions are made.

Construction of Q_a proceeds abductively: given the tree α, we want to generate the conditions that, *prior to the performance* of action a, could cause the outcome probabilities (with respect to the partitions induced by α) to vary. We proceed in a stepwise fashion, "explaining" each of the interior nodes of α in turn, beginning with the root node and proceeding recursively with its children. It is important to remember that all of the propositions in α refer to the state at stage k, and that each of the propositions in Q_a refer to stage $k+1$. These propositions are related to each other via the state-transition trees for action a. Space precludes a full exposition of the method—we refer to [4] for details of this method (applied to fully observable MDPs)—so we present a simple example.

Example To illustrate this process, consider the following example, illustrated in Figure 5. We take the immediate reward function (see Figure 2) to be a tree α^0 (the initial value tree), and we wish to generate the expected future value tree for stage 1 assuming action *GetC* is taken and that α^0 determines value at stage 0. We begin by explaining the conditions that influence the probability of *WC′* under *GetC* (Step 1 of Figure 5). This causes *Tree*(*WC′*|*GetC*) to be inserted into the tree α: as indicated by Figure 1, *WC′* is not affected by the action *GetC*, and thus remains true or false with certainty. The leaves of this partial tree denote the probability of *WC′* being true *after* the action given its value (*WC*) *before* the action. We then explain *HC′* (Step 2). Since the initial value tree asserts that *HC* is only relevant when *WC* is true, the new subtree *Tree*(*HC′*|*GetC*) is added only to the left branch of the existing tree, since *WC′* has probability zero on the right.

Again, the probabilities labeling the leaves describe the probability of the variable in question *after the action*, while the labels on interior nodes of the branches relate the conditions *before the action* under which these probabilities are valid. This becomes clear in Step 3, where we consider the conditions (prior to *GetC*) that affect the occurrence of W' (wet) after *GetC*: the relation (*Tree*(W'|*GetC*)) is complex, depending on whether the robot had an umbrella and whether it was raining. This final tree has all the information needed to compute expected *future* value at each leaf—the probabilities at each leaf uniquely determine the probability of landing in any partition of initial value tree under *GetC*.

Finally, we note that to get the true expected value (not just future value), we must add to each of these trees both the current state value $R(s)$ and the action cost $C(a,s)$. This will generally require the simple addition of cost/reward to the values labeling the leaves of the current tree, though occasionally a small number of additional distinctions may be required. Figure 6 shows the expected (total) value tree for *GetC* obtained by adding $R(s)$ and $C(a,s)$ to the future value tree of Figure 5. Figure 6 also shows the tree for *TestC*.

4.2 Incorporating Observations

To account for observations, every element of $\aleph^k$ must correspond to a given action choice a and an observation strategy that assigns a vector in $\aleph^{k-1}$ to each $o \in Obs(a)$. We now consider the problem of generating the actual α-tree corresponding to action a and the strategy assigning $\alpha_j \in \aleph^{k-1}$ to the observation o_j.

Since the conditions that influence the probability of a given observation affect expected future value (since they affect the subsequent choice of α-vector with $k-1$ stages-to-go), the new tree α must contain these distinctions. Thus α is partially specified by $Tree(Obs|a)$, the observation tree corresponding to action a. Recall that the branches of this tree correspond to the conditions relevant to observation probability, and the leaves are labeled with the probability of making any observation o_j. To the leaves of $Tree(Obs|a)$ we add the *weighted sum of the explanation trees* (see also [16]). More specifically, at each leaf of $Tree(Obs|a)$ we have a set of possible (nonzero probability) observations; for exposition, assume for some leaf these are o_i and o_j. Under the conditions corresponding to that leaf, we expect to observe o_i and o_j with the given probabilities *Pr*(o_i) and *Pr*(o_j), respectively. We thus expect to receive the value associated with the explanation tree for α_i with probability *Pr*(o_i), and that for α_j with probability *Pr*(o_j). We thus take the weighted sum of these trees and add the resulting merged tree to the appropriate leaf node in $Tree(Obs|a)$.[5]

[5] Computing the weighted sum of these trees is relatively straightforward. We first multiply the value of each leaf node in a given tree by its corresponding probability. To add these weighted trees together involves constructing a *smallest* single tree that forms a partition of the state space that subsumes each of the explanation trees. This can be implemented using a simple tree merging opera-

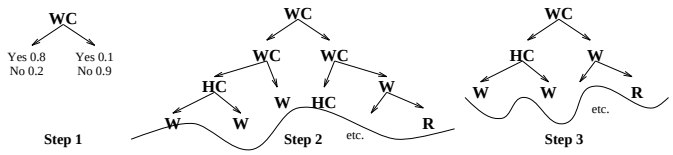

Figure 7: New α-tree for Stage $n-2$

Example Consider the following example illustrated in Figure 7. We assume that trees α_1 and α_2, the trees for *GetC* and *TestC* in Figure 6, are elements of $\aleph^1$. We consider generating the new tree α to be placed in $\aleph^2$ that corresponds to the action *TestC* and invokes the strategy that associates α_1 with the observation *Yes* and α_2 with the observation *No*. We begin by using the observation tree for *TestC*: the observation probability depends only on *WC* (see Step 1 of Figure 7). We then consider the weighted combination of the trees α_1 and α_2 at each leaf: to the leaf *WC* we add the tree $0.8\alpha_1 + 0.2\alpha_2$ and to $\overline{WC}$ we add $0.1\alpha_1 + 0.9\alpha_2$. This gives the "redundant" tree in the middle of Figure 7. We can prune away the inconsistent branches and collapse the redundant nodes to obtain the final tree α, shown to the right.

We note that this simple combination of trees is due in part to the dependence of observations on only the pre-action state (as is the "separation" in Equation 3). This allows the direct use of $Tree(Obs|a)$ in assessing the influence of observations on the values of pre-action states. However, should observations depend instead on the post-action state as is usual in the POMDP literature [17, 6], our algorithm is complicated only in slight detail. In this case, $Tree(Obs|a)$ refers to variables in the state following the action, (recall we are interested in the values of states prior to the action). Generating the probability of the observations based on pre-action variables is, however, a simple matter: we simply generate an explanation for the observation in a manner similar to that described in Section 4.1 (though, in fact, much less complicated). The standard explanation trees are then merged together within this slightly more complicated tree instead of $Tree(Obs|a)$.

4.3 Generation of $\aleph_k$ and Pruning

The algorithm for construction of the structured value function proceeds exactly as Monahan's algorithm in the previous section. The substantial difference is that we start with a tree-structured initial reward function as the sole α-tree at stage 0, and generate collections $\aleph^k$ of α-trees rather than simple (e.g., vector-represented) α-functions. The process described above involves some overhead in the construction of explanation trees and piecing them together with observation probabilities. We note, however, that we need not generate the trees for $Q_{\alpha^k_{a,o}}$ for each observation strategy individually. This tree depends only on a and $\alpha^k_{a,o}$, not on o. Thus, we need only construct $|\mathcal{A}||\aleph^k|$ such trees; the $|\mathcal{A}||\mathcal{O}||\aleph^k|$ different trees in $\aleph^{k+1}$ are simply different weighted combinations of these (corresponding to different observational strategies). Further savings are possible in piecing together certain strategies (e.g., if OS_a associates the same vector with each observation, the explanation tree for a can be used directly).

One can prune away dominated α-trees from $\aleph^k$, as suggested by Monahan. As described in Section 2.6, this too exploits the structured nature of the α-trees.

Finally we note that most POMDP algorithms are more clever about generating the set of possible α-vectors. For example, Sondik's algorithm does not enumerate all possible combinations of observational strategies and then eliminate useless vectors. We focus here on Monahan's approach because it is conceptually simple and allows us to illustrate the exact nature of structured vector representations and how they can be exploited computationally. We are currently investigating how algorithms that use more direct vector generation can be adapted to our representation. The Witness algorithm [6] appears to be a promising candidate in this respect, for the LPs used to generate "promising" α-vectors are amenable to the representations described here.

4.4 Executing Policies

Given $\aleph^k$ and a belief state π, we can determine the optimal action with k stages-to-go by choosing an $\alpha \in \aleph^k$ such that $\pi \cdot \alpha$ is maximal, and carrying out the action associated with α. We can then make our observations, and use Bayes rule to update our belief state. We are then ready to repeat and choose an action for the next stage.

The structured representation of value functions, which will generally be compact, can aid policy execution as well. This will be especially true if the belief state is itself represented in a structured way. The expected value of belief state π is the sum of the values at the leaves of the α-tree multiplied by the probabilities of the leaves. The probability at each leaf is the probability of the conjunction of propositions that lead to it (which can be derived from the belief state). Moreover, this also specifies which probabilities are *required* as part of the belief state (and which may be ignored). For instance, if it is discovered in the generation of the value function that certain variables are never relevant to value, these distinctions need not be made in the belief state of the agent.

5 Approximation Methods

While the standard vector representation of α-functions requires vectors of exponential size (in the number of propositions), computing with decision trees allows one to keep the size of the representation relatively small (with potentially exponential reduction in representation size). However, our example illustrates the natural tendency for these trees to become more "fine-grained" as the horizon increases. Depending on the problem, the number of leaves can approach (or reach) the size of the state space. In such cases, the overhead involved in constructing trees may outweigh the marginal decrease in effective state-space size.

tion (for example, see [4] where similar tree merging is used for a different purpose). In terms of rules [16], this effect is obtained by explaining the conjunction of the roots of the trees.

However, an additional advantage of tree (or related) representations is the ease with which one can impose approximation schemes. If the α-tree makes certain distinctions of marginal value, the tree can be pruned by deleting interior nodes corresponding to these distinctions. Replacing the subtrees rooted at U in tree α_1 of Figure 6 by a midpoint value introduces a (maximum) error of 0.5 in the resulting approximate value function. This may be acceptable given the shrinkage in the representation size it provides. This contraction has the effect of reducing the size of new trees generated for subsequent stages, as well. In addition, the error introduced can be tightly controlled, bounded and traded against computation time.[6] In this sense, tree-based representations provide an attractive basis for approximation in large discrete problems.

A major difficulty with Monahan's algorithm is the fact that the number of (unpruned) α-functions grows exponentially with the horizon: $\aleph^k$ contains $(|\mathcal{A}||\mathcal{O}|)^k$ pieces. Of course, pruning dominated α-functions can help, but does not reduce worst-case complexity. The methods above address the size of α-trees, but not (apart from pruning) their number.

A second advantage of the tree-based representation, and approximation schemes based upon it, is the possibility of greatly reducing the number of α-trees needed at each stage. By blurring or ignoring certain distinctions, the number of dominated vectors (hence the amount of pruning) may be increased. In addition, "approximate domination" testing can be aided: for example, if one tree has strictly worse values than another except for slightly better values in one small region of the state space, it could be pruned away. Again, the compactness of the α-trees can be exploited in such tests, as in Section 2.6. Indeed, this complements certain work that reduces the number of α-functions, such as [13].[7] These suggestions are, admittedly, not developed completely at this point. However, a firm grasp of optimal decision making with structured representations provides a sound basis for further investigation of structured approximation methods.

6 Concluding Remarks

We have sketched an algorithm for constructing optimal policies for POMDPs that exploits problem structure (as exhibited by rules or decision trees) to reduce the effective state space at various points in computation. The crucial aspect of this approach is the ability to construct the conditions relevant at a certain stage of the process given the relevant distinctions at the following stage. This merging of planning and optimization techniques (and related approaches) should provide significant improvements in policy construction algorithms.

Of great interest are extensions of this work to algorithms that enumerate "vectors" (in our case, trees) in a more direct fashion (rather than by exhaustive enumeration and elimination), as well as empirical evaluation of the overhead of tree construction. In addition, the development of approximation methods such as those alluded to above is an important step.

Acknowledgements: Thanks to Tony Cassandra, Leslie Kaelbling, Michael Littman and Ron Parr for their helpful discussion and comments. This research supported by NSERC Grants OGP0121843 and OGPOO44121.

[6] See [3] on this type of pruning.

[7] In [13], a continuous approximation of the value function is adjusted via gradient descent on the Bellman error; but there is one adjustable parameter per state. A (dynamic) tree-based representation of the value function may be exploited here.

References

[1] R. E. Bellman. *Dynamic Programming*. Princeton, 1957.

[2] C. Boutilier, T. Dean, and S. Hanks. Planning under uncertainty: Structural assumptions and computational leverage. *3rd Eur. Workshop on Planning*, Assisi, 1995.

[3] C. Boutilier and R. Dearden. Approximating value trees in structured dynamic programming. *ML-96*, to appear, 1996.

[4] C. Boutilier, R. Dearden, and M. Goldszmidt. Exploiting structure in policy construction. *IJCAI-95*, pp.1104–1111, Montreal, 1995.

[5] A. R. Cassandra. Optimal policies for partially observable Markov decision processes. TR CS-94-14, Brown Univ., Providence, 1994.

[6] A. R. Cassandra, L. P. Kaelbling, and M. L. Littman. Acting optimally in partially observable stochastic domains. *AAAI-94*, pp.1023–1028, Seattle, 1994.

[7] T. Dean, L. P. Kaelbling, J. Kirman, and A. Nicholson. Planning with deadlines in stochastic domains. *AAAI-93*, pp.574–579, Washington, D.C., 1993.

[8] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Comp. Intel.*, 5(3):142–150, 1989.

[9] T. Dean and M. Wellman. *Planning and Control*. Morgan Kaufmann, 1991.

[10] R. A. Howard and J. E. Matheson. Influence diagrams. R. A. Howard and J. Matheson, eds., *The Principles and Applications of Decision Analysis*, pp.720–762, 1981.

[11] W. S. Lovejoy. Computationally feasible bounds for partially observed Markov processes. *Op. Res.*, 39:162-175, 1991.

[12] G. E. Monahan. A survey of partially observable Markov decision processes: Theory, models and algorithms. *Mgmt. Sci.*, 28:1–16, 1982.

[13] R. Parr and S. Russell. Approximating optimal policies for partially observable stochastic domains. *IJCAI-95*, pp.1088–1094, Montreal, 1995.

[14] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[15] D. Poole. Probabilistic Horn abduction and Bayesian networks. *Art. Intel.*, 64(1):81–129, 1993.

[16] D. Poole. Exploiting the rule structure for decision making within the independent choice logic. *UAI-95*, pp.454–463, Montreal, 1995.

[17] R. D. Smallwood and E. J. Sondik. The optimal control of partially observable Markov processes over a finite horizon. *Op. Res.*, 21:1071–1088, 1973.

[18] C. J. C. H. Watkins and P. Dayan. Q-Learning. *Mach. Learning*, 8:279–292, 1992.