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

Phone: 604-822-4632, 604-822-6254; **Fax:** 604-822-5485

Abstract: Partially-observable Markov decision processes provide a very general model for decision-theoretic planning problems, allowing the 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 information or *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 the 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 representation to structure the belief space for POMDP algorithms. This allows irrelevant distinctions to be ignored. Apart from speeding up optimal policy construction, we suggest that such representations can be exploited to great extent in the development of useful approximation methods. We also briefly discuss the difference in perspective adopted by influence diagram solution methods *vis à vis* POMDP techniques.

Keywords: decision theoretic planning, partially observable MDPs, influence diagrams, structured representation

1 Introduction

One of the most crucial aspects of intelligent agenthood is planning, or constructing a course of action appropriate for given circumstances. While planning models in AI have attempted to deal with conditions of incomplete knowledge about the true state of world, possible action failures and multiple goals, only recently has work on planning begun to quantify such uncertainties and account for competing objectives. This increased interest in *decision-theoretic planning* has lead to planning algorithms that essentially solve multistage decision problems of indefinite or infinite horizon. As a result, representations commonly used in the uncertain reasoning community, such as Bayes nets (BNs) influence diagrams (IDs), and their equivalents, have been adopted for planning purposes [6, 7, 14, 2] (see [1] for a brief survey).

A useful underlying semantic model for such DTP problems is that of *partially observable Markov decision processes* (POMDPs). This model, used in operations research [15, 10] and stochastic control [3], accounts for the tradeoffs between competing objectives, action costs, uncertainty of action effects and observations that provide incomplete information about the world. However, while 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 of the system grows exponentially with the number of variables needed to describe the problem.

Influence diagrams and Bayesian networks [8, 12] 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, via independencies among random variables. As such, problems can be specified much more compactly and naturally. In addition, algorithms for solving influence diagrams can exploit such independencies for computational gain. Unfortunately, one of the difficulties associated with ID solution methods is the association of a policy with the *observed history* of the agent: at any stage, the best decision is conditioned on all previous observations made and actions taken. Thus, while an exponential blow-up in the state space is partially alleviated, the size of a *single* policy grows exponentially with the horizon.

POMDP algorithms from the OR community adopt a very different perspective on the problem. The agent's observable history is not explicitly recorded; rather, a *belief state* or probability distribution over the state space is constructed that summarizes this history. At any stage, the agent's belief state is sufficient to

determine the expected value of any subsequent action choice [15]. By converting a partially observable process into a completely observable process over the belief space, history independent dynamic programming algorithms can be used. While this belief space is continuous, Sondik [15, 17] has shown that properties of this space can be exploited that give rise to finite algorithms for solving POMDPs (by partitioning the belief space appropriately). These algorithms are very computationally intensive, relying on representations of (parts of) value functions and belief states that require entries for every state of the system. In addition, the number of partitions needed may grow exponentially with the horizon. Because of this, current algorithms are only feasible for problems with tens of states [5].

In this paper we propose a method for policy construction based on the usual POMDP solution techniques, but that exploit representations to reduce the "effective" state space of a problem. We assume a structured state space (in contrast to usual POMDP formulations) generated by a set of random variables, that actions have relatively local effects on variables and observations and that rewards and costs are also structured. This allows BNs and IDs to represent large problems compactly. In addition, we use tree-structured conditional probability and utility matrices — or equivalently *rules* — to represent additional structure in the independence of variable *assignments*. We will use this representation to identify the *relevant* distinctions in the state space at any point in our POMDP algorithm. As a result, the values associated with a collection of states may be represented once.

For example, suppose that we have identified the fact that at a certain stage of the process that only variable P has an effect on subsequent expected value. Furthermore, assume that for a given action a only variables Q and R influence the probability of P under action a. In addition, suppose that if Q is true, the effect on a is independent of R. Then the expected value of action a at the previous stage is the same at all states satisfying Q, at all states satisfying $\neg Q \wedge R$ and $\neg Q \wedge \neg R$. In other words, the value of a depends only on the probability of these three propositions. Thus, only three values need be computed.[1] This structure can be identified readily from a Bayesian network representation of action a (we will use structured conditional probability matrices as well). Thus, by using the structure of the problem, we can identify the relevant distinctions at any stage of a problem, and perform computations *once* for the entire collection of states where the distinctions are irrelevant to utility calculations. We note that the algorithm we use exploits the partitioning ideas of the *Structured Policy Iteration* algorithm of

[1] We ignore possible observations in this motivating example.

[2]. This paper essentially extends these ideas to account for partial observations of the state, and applies them in a piecewise fashion to the components of a value function (as we describe below).

Our algorithm solves the question of compactly representing belief states and value functions, without specifying or computing separate values for each state (the state space is exponential in the number of variables). It does not address the question of a potentially exponentially growing (in the horizon) partitioning of the state space. In general, this growth may be necessary to ensure optimality. That is, an optimal policy may be very fine-grained. Our algorithm does not sacrifice optimality, it merely exploits problem structure in optimal policy construction. Avoiding the exponential blow-up in the partitioning of the belief space necessarily involves approximation. While we do not address this question here, the use of structured representations that identify and quantify relevant distinctions provides tremendous leverage for approximation techniques. For instance, if certain distinctions are to be ignored, a structured value function representation allows one to quickly identify which variables (conditionally) have the most or least impact on value.

Section 2 briefly contrasts the perspectives adopted in the POMDP and ID formulation of decision problems. In Section 3, we describe POMDPs and our two-stage belief network representation of a POMDP. In Section 4, we describe a particular POMDP algorithm due to Monahan [10], based on the work of Sondik [15, 17]. We choose this algorithm because it is conceptually simple and easy to present in the restricted space available. In Section 5, 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. We provide some concluding remarks in Section 6.

2 POMDPs and Influence Diagrams

In this section we clarify some distinctions underlying the POMDP and ID approaches to the representation of partially observable decision problems.

History vs. Belief State: In a fully observable Markov decision problem, the decision maker or *agent* can accurately observe the state of the system being controlled. Under the Markov assumption this observation is the only information required to make an optimal action choice. In a partially observable setting, the the

agent only gets noisy, incomplete observations about the true nature of the state. An optimal action choice cannot be made on the basis of its current observations — the agent's past action and observations are relevant to its decision, for they influence its current state estimate.

There are two distinct approaches to incorporating this past history in the decision process.

1. The agent can condition its action choice explicitly on its past history. Under this approach, a decision is conditioned on all previous actions and the current and past observations. This model is adopted in the solution of influence diagrams [8], where it is called the "no forgetting" principle. Suppose we have a set of possible observations $\mathcal{O}$ that can be made at each stage and a set of actions $\mathcal{A}$. At stage s (after $s-1$ decisions have been taken), a policy maps each possible history into an action choice; i.e., $(\mathcal{O}^s \times \mathcal{A}^{s-1}) \rightarrow \mathcal{A}$. Using the dynamic programming principle, we can optimize the action for each history (element of $(\mathcal{O}^s \times \mathcal{A}^{s-1})$) independently. Since $\mathcal{O}$ and $\mathcal{A}$ are finite, at each stage s, $(\mathcal{O}^s \times \mathcal{A}^{s-1})$ is finite. Unfortunately, this results in exponential growth in policy specification (and computation).

2. The agent might instead maintain a *belief state*, or distribution over possible current system states, summarizing its history (as we elaborate below). Under this approach, the action choice is conditioned on the agent's current belief state. Given n system states, the set of belief states is $\Re^{n-1}$ (assigning a probability to $n-1$ of the states — the probability of the other state can be derived from this); and a policy is a function from belief states to actions (i.e., $\Re^{n-1} \rightarrow \mathcal{A}$). Again, we can make an optimal decision for each belief state independently. While the domain of the policy function does not grow with the stage, it is an $n-1$-dimensional infinite space. This approach to policy formulation is adopted by POMDP solution algorithms in the OR community, but requires some method of dealing with this uncountable belief space.

What is a state? Another distinction between POMDP and influence diagram specifications of a partially observable problem lies in the notion of a state. In the POMDP formulation, states are taken to be states of the core process, or system being controlled. Actions taken and observations made are viewed as external to the system and are not part of the system state. The system state

is sufficient to render the effects of actions independent of past history. Action choices and observations are modeled at a meta-level. Given some action choice and observation, Bayes rule is used to update the belief state: observations just change the probability distribution of the states.

In the BN/ID tradition, the observation made is a random variable and makes up part of the "state" in addition to the variables that define the core process (the observation is often *added* as an "extra" variable [12]). Taking the state space to be the joint probability space, in these models an observation simply makes some states impossible; conditioning on the observations involves setting these inconsistent "expanded" states to have probability zero, and renormalizing. The same representation is adopted for actions: the action choice at any stage is a variable that again constitutes part of the "expanded" state space. The value of this variable, however, is chosen by the agent and is "independent" of other variables.

3 POMDPs and Structured Representations

In this section we review the classic presentation of POMDPs adopted in much of the operations research community. We refer to [15, 17, 9, 5] for further details and [10, 4] for surveys. We then describe how structured representations adopted by the AI community can be used to represent factored state spaces.

3.1 Explicit State Space Presentation

We assume the dynamics of the system to be controlled can be modeled as a POMDP with a stationary (stage-independent) dynamics. We assume a finite set of states $\mathcal{S}$ that capture all relevant aspects of the system, and a set of actions $\mathcal{A}$ available to the agent or controller. For simplicity we assume all actions can be taken (i.e., 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. These transition probabilities can be encoded in an $|\mathcal{S}| \times |\mathcal{S}|$ matrix for each action. This formulation assumes the Markov property for the system in question.

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* $\mathcal{O}$ an agent can make. These observations provide evidence for the true nature of various aspects of the state. In general, the observation made at any stage will depend stochastically on the state, the action performed and the outcome of the action at that stage. We assume a family of distributions over observations for each s_i, s_j, a_k such that $Pr(s_j|s_i, a_k) > 0$. We 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 taken and the 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.

Finally, we assume 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* $C(s, a)$ denoting the cost of taking action a in state s. (For our purposes, it suffices to consider rewards and penalties separately from action costs.) A plan or *policy* is a function that determines the choice of action at any stage of the system's evolution. A policy is optimal (for a specified horizon or number of stages) if it maximizes the expected value of the system trajectory it induces; that is, it maximizes the expected rewards accumulated (where "reward" incorporates both action costs and state rewards and penalties).

The appropriate action choice requires that an agent predict the expected effects of possible actions and the expected value of the states visited for a given (sequence of) action choice(s). Although the utility and the possible actions depend only on the current state, the whole history is relevant to our beliefs about the current state. As mentioned above and adopted in ID algorithms, a policy can be represented as a mapping from any given initial state estimate and sequence of actions and observations over the previous n stages to the action for stage $n + 1$. However, an especially elegant way to treat this problem is to maintain a current belief state as described above.

Intuitively, a *belief state* π is a tuple $\langle \pi_1, \cdots \pi_n \rangle$ where π_i denotes the probability that the system is in state s_i. Because of the Markov assumption, this belief state is sufficient to predict the expected effects of any action on the state of the system. Thus, given some state of belief π^k characterizing our estimate of 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

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

$T(\pi, a, o)_i$ is the probability that the system is in state i after action a is taken and observation o made, given prior belief state π.

It is easy to see that $T(\pi, a, o)$ summarizes all necessary information for subsequent decisions. 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. 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). We will review policy construction techniques in the next section.

3.2 Two-Stage Belief Network

Although one can formulate a problem as an explicit POMDP, it is unreasonable to expect a problem to be specified in such a manner, since state spaces grow exponentially with the number of variables relevant to the problem at hand. The explicit formulation requires on two specify a $|\mathcal{S}| \times |\mathcal{S}|$ probability matrix for each action describing transition probabilities, a $|\mathcal{S}| \times |\mathcal{O}|$ probability matrix for each action a describing observation probabilities, and $|\mathcal{S}|$ action cost and reward vectors. Regularities in action effects and reward structure will usually permit more natural and concise representations.

Consider the following simple example: we have a robot that can check whether a user wants coffee and can get it by going to the shop across the 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 to get 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 it umbrella, etc., we focus on two actions: getting coffee *GetC* and checking whether the user wants coffee by means of a quick inspection *TestC*.

We represent the dynamics of a state using a "two-slice" temporal Bayes net [6]: we have one set of nodes representing the state of the world at prior to the

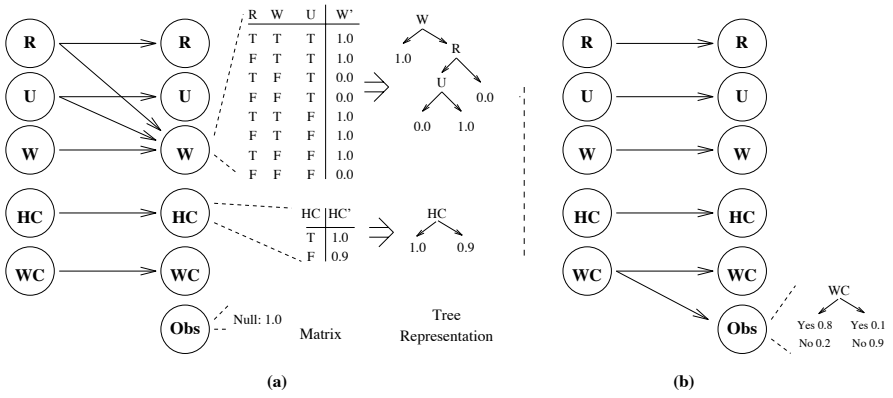

Figure 1: Action Networks for (a) GetC and (b) TestC

action (one node for each state variable P), another set representing the world after the action has been performed, and directed arcs representing causal influences between the these sets. In addition, we have a distinguished variable representing the possible observations that can be made after performing the action. The variables that influence the observation are indicated by directed arcs. We use $Obs(a)$ to denote the set of observations possible given a.

Figure 1 illustrates this network representations for the actions *GetC* (a) and *TestC* (b). Although, we only require that the graph be acyclic, for ease of exposition we assume here that the only arcs are directed from pre-action variables to post-action variables.[2] The post-action nodes have the usual matrices describing the probability of their values given the values of their parents, under action A. We assume that these conditional probability matrices are represented using a tree-structure (or if-then rules) as done in [16, 13]. This representation is exploited to great effect in [2] in the solution of completely observable MDPs. We will adopt the same ideas below. The tree representation of the matrix for variable W is illustrated in Figure 1(a), illustrating how asymmetries are exploited. The tree associated with proposition P in the network for an action A is denoted $Tree(P|A)$. The observation variable in this action has a single value – *GetC* provides no information about the true state of the world. The *TestC* action has

[2]Causal influences between post-action variables should be viewed as *ramifications* and will complicate our algorithm slightly, but only in minor detail, not in conceptual understanding.

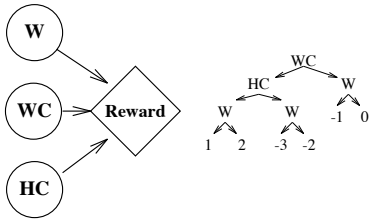

Figure 2: Reward Function Network

no effect on the core state variables, but does provide information about the user's coffee preference: an observation *Yes* indicates *WC* and *No* indicates $\overline{WC}$ (with the specified errors).

It is easy to see how a set of action networks can be put together into a single network with a decision node, corresponding to an influence diagram. We add a decision node denoting the choice of action, with arcs to each post-action node. The conditional probability tree for each such node P has the action choice as a root, and $Tree(P|A)$ is attached to the branch at that root. The set of values for the observation variable is the union of possible observations from the individual action networks, and its tree also first branches on the action choice and then contains the individual observation trees. See [1] for a brief survey of such representations.

The state reward function can be represented in a similar structured fashion using a single value node and tree-structured matrix (see Figure 2) that allows features that do not influence reward to be ignored. The leaves of the tree denote the reward associated with the states determined by the branch.[3] A similar representation for action cost can be used. We will assume here that action costs are constant (we assume a cost of 1.0 for *GetC* and 0.5 for *TestC*).

4 Monahan's POMDP Algorithm

Using a belief vector to summarize the current state estimate without explicitly incorporating history allows policies to be expressed independent of history — our

[3]Note that in all of these network representations, the network is not strictly necessary — the conditional probability trees themselves allow determination of parents [13].

decisions are contingent only on the current state of belief. However, we have in fact potentially made the specification of a policy much more difficult. The space of possible belief states $\mathcal{B}$ is uncountably infinite. Computing, or even associating, an action choice with each belief state is impossible on this explicit formulation. However, as observed by Sondik [15, 17], optimal policies may be computed and specified finitely by performing computations and associating actions with various *regions* of the belief space. The crucial component of any POMDP algorithm is the discovery of an appropriate set of regions for an optimal policy.

Most POMDP algorithms do not, in fact, explicitly compute the regions of $\mathcal{B}$ for which the optimal decision may vary. Rather, most keep track of a finite set of α-*vectors* for each stage of the decision process.[4] An α-vector is a vector of size $|\mathcal{S}|$ that specifies a value for each state. Each α-vector has an associated action; intuitively, this vector defines the value of performing that action in each state given some fixed future value function. Given any $\pi \in \mathcal{B}$, the expected value of π given a vector α is the dot product $\pi \cdot \alpha$. Again, a crucial observation of Sondik is the fact that the k-stage *value function* V for any policy is piecewise linear and convex, and that each linear component of V can be represented as an α-vector. Since the true value function V is piecewise linear and convex, we can represent the linear components as a set of α-vectors $\{\alpha_i\}$, and the true value of π is the maximum of the products $\pi \cdot \alpha_i$. As we see below, each vector has an action associated with it: the optimal action choice at any stage given π will be the action associated with the maximizing vector.

The computation of an optimal policy requires that we generate a set of vectors for stage k, given the set of vectors for stage $k+1$. In other words, a dynamic programming style computation can be used to determine the representation of the stage k value function (hence, policy) given the stage $k+1$ value function. This is the method adopted by all POMDP algorithms (although the precise technique for enumerating the set of required vectors varies). Let $\aleph_{k+1}$ be the set of α-vectors associated with stage $k+1$. Following Monahan [10], the set of vectors necessary at stage k is generated as follows: for each action a, associate some $\alpha^l \in \aleph_{k+1}$ with each observation o_l in $\mathit{Obs}(a)$. We generate a new α-vector as follows (where α_i denotes its ith component). Judicious application of Bayes rule allows the following expression to be used, whose quantities can be read directly from the

[4] Or a single set of such vectors, used for every stage, if a stationary policy is sufficient.

problem specification:

$$\alpha_i = C(a, s_i) + R(s_i) + \sum_l \sum_{s_j \in \mathcal{S}} \mathit{Pr}(o_l | s_i, a s_j) \mathit{Pr}(s_j | s_i, a) \alpha_j^l$$

Intuitively, we can think of this new vector as specifying the value of any belief state π at stage k assuming that action a will be chosen and that if o_l is observed, the value for stage $k+1$ is determined by the stage $k+1$ vector α^l. For each action, we generate a set of new vectors of size $|\mathit{Obs}(a)|^{|\aleph_{k+1}|}$; and the set $\aleph_k$ is the union of the sets generated by each action. The action *associated* with each vector is the action a used to generate it. The optimal action choice at stage k for any π is determined by the computing the $\alpha \in \aleph_k$ that maximizes $\pi \cdot \alpha$ and adopting the the action associated with α. We emphasize that the policy is not explicitly represented: the algorithm constructs a set of vectors for each stage of a finite horizon problem from which the optimal action choice for any belief state can be easily determined (for infinite horizon problems, a single set of vectors can be obtained).

Of course, it may be that many of these generated vectors are irrelevant in the sense that they are dominated by other vectors at that stage. By eliminating such dominated vectors before proceeding to compute vectors for earlier stages, the number of subsequent vectors generated (and computation time) can be drastically reduced. Dominated vectors are "easily" identified using a set of linear programs (with variables for each state), one for program for each vector [10].

We note that for a finite horizon n-stage problem, we begin this process by setting $\aleph_n$ to contain the immediate (state) reward vector. For an infinite horizon discounted problem, under certain assumptions, we run the process until convergence is reached; thus the initial vector choice is less crucial (see [17]). We will focus on finite stage problems in the sequel, though infinite horizon problems offer no special difficulties.

5 Exploiting Structure in Solving a POMDP

The crucial step in Monahan's algorithm is the generation of the set $\aleph_k$ of α-vectors for stage k given the set of vectors $\aleph_{k+1}$. We note that each α-vector has size $|\mathcal{S}|$, and that the new vectors are generated pointwise. Thus, even the representation and construction of a single vector can be computationally prohibitive: as the number of variables grows, this approach quickly becomes infeasible. However,

given that a problem can be represented compactly using a network representation, one should expect that the set of α-vectors can also be compactly represented. If the values associated with various states in such a vector are the same, we can exploit this fact by clustering states together that have the same value. A very simple example is the initial α-vector (for stage n) which is simply the immediate state reward function. For a given problem, this vector may be compactly expressed in tree form as in Figure 2, using 6 values instead of 32.

We propose a method for optimal policy construction that, in general, eliminates the need to represent α-vectors pointwise, and the need to construct new vectors in the standard pointwise fashion. Our method is based on Monahan's algorithm, but exploits the fact that: a) at any stage of the process, the α-vectors representing the value function for that stage may be structured, making only *relevant* distinctions in the state space; and b) the problem representation allows us to preserve this structure in the generation of new α-vectors.

Each linear component of the piecewise-linear value functions used in the classic POMDP representations is represented as an α-vector consisting of the values associated (in that linear segment) with each state. We represent these segments using α-*trees*. These are trees whose interior nodes are state variables – the edges descending from each variable correspond to possible assignments to that variable – and whose leaves are real values. This is precisely the representation we used for the immediate reward function above. An α-tree partitions the state space in the obvious way: each branch denotes the set of states satisfying the variable assignment determined by that branch. This represents the (unique) α-vector where the value of a state s is simply the value associated with the partition containing that state.

5.1 Generation of a Single α-tree

The algorithm we present below allows one to construct the set of α-trees required at stage k given the set $\aleph_{k+1}$ of trees associated with stage $k+1$. Before presenting that algorithm in detail, we first describe the main intuitions underlying our method by considering the construction of a single tree. We consider first the subproblem of generating the value tree corresponding to performing a single action at stage k assuming that future expected value is given by a fixed $\alpha \in \aleph_{k+1}$. We then consider how to combine such trees to form the stage k α-vector corresponding to a particular strategy that associates one element of $\aleph_{k+1}$ with each $o \in \mathit{Obs}(a)$.

Suppose we wish to determine the value of performing action a at stage k,

and that the expected future value (from stage $k+1$ on) is determined by a fixed tree $\alpha^{k+1} \in \aleph_{k+1}$. Our method for generating the new α-tree α^k exploits the *abductive repartitioning algorithm* described in [2], and is closely related to Poole's probabilistic Horn abduction [13] (we refer to [2] for further details). Roughly, given a structured value function α^{k+1}, the conditions under which two states can have different expected future value at stage k (under action a) can be easily determined by appeal to the action representation for a. In particular, although an action may have different effects at two states, if the differences in those effects are only related to variables (or variable assignments) that are *not relevant to the value function* α^{k+1}, then those states have identical expected future value and need not be distinguished in the function α^k. We construct the tree α^k in a such a way that only these relevant distinctions are made.

Construction of α^k proceeds abductively: given the tree α^{k+1}, 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^{k+1}) to vary. We proceed in a stepwise fashion, "explaining" each of the interior nodes of α^{k+1} in turn, beginning with the root node and proceeding recursively with its children.

The α^k tree is initially empty. We explain the root node (say P) of α^{k+1} by generating the conditions under which the values of this variable can have different probabilities under action a (the "explanation" of P). Note that this explanation is essentially lifted from the problem description, since it is nothing more than the probability tree $Tree(P|a)$ in the action network for a. This tree is the initial *partial* α^k tree. Every partial tree has leaves that are labeled with the probabilities of all variables explained so far. Such explanations continue for each variable in the tree α^{k+1}, and each explanation is added to the leaf nodes of the current partial tree. We note that explanation for a variable X is not added to every branch of the current partial tree, however. The tree α^{k+1} describes the conditions under which X is relevant to future value. The explanation for X is thus only relevant in circumstances that allow these conditions to be met with positive probability, and $Tree(X|a)$ need only be added to those branches where these conditions are possible. Since the explanation for any ancestor of X in α^{k+1} is generated before X, the relevant probabilities already label the leaves of the current partial tree. Finally, once all variables are explained, the expected future reward for each branch of α^k can be easily computed: the probability labels at each branch determine the probability of ending up in any partition of the tree α^{k+1} under a.

We note that redundant and inconsistent branches in the explanations added to the partial tree are also deleted. For example, if the variable labeling a node

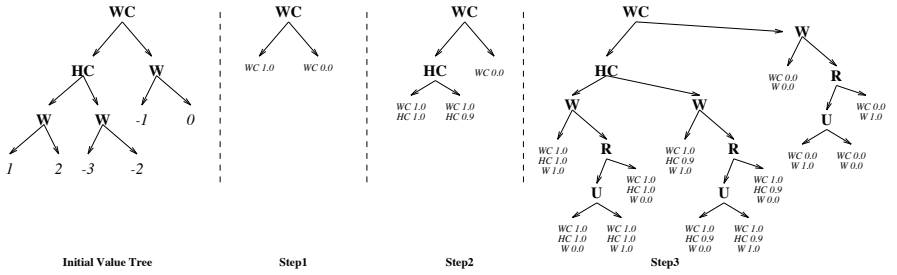

Figure 3: Generating Explanation of Future Value

of $Tree(X|a)$ occurs earlier in the partial tree for V^{i+1}, that node in $Tree(X|a)$ can be deleted (since the assignments to that node in $Tree(X|a)$ must be either redundant or inconsistent). Thus, much shrinkage is possible.[5]

To illustrate this process, consider the following example, illustrated in Figure 3. We take the immediate reward function to be a tree α^{k+1} (the initial value tree), and we wish to generate the expected future value tree for stage k assuming action *GetC* is taken and assuming that α^{k+1} determines value at stage $k+1$. We begin by explaining the conditions that influence the probability of *WC* under *GetC* (Step 1). This causes $Tree(\textit{WC}|\textit{GetC})$ to be inserted into the new tree. We then explain *HC* (Step 2). Since the initial value tree asserts that *HC* is only relevant when *WC* is true, this new tree is added only to the left branch of the existing tree, since *WC* has probability zero on the right. Note that these probabilities describe the probability of the variable in question *after the action*, while the branches relate conditions that affect these probabilities before the action. This becomes clear in Step 3, where we consider the conditions (prior to *GetC*) that affect the occurrence of W (wet) after *GetC*. Note that 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 $C(a, s)$. While in general this may cause the explanation trees to grow (e.g.,

[5]It is this fact that keeps trees as compact as possible and tends to make the set of trees stabilize rather quickly. We refer to [2] for details (see also Figure 5).

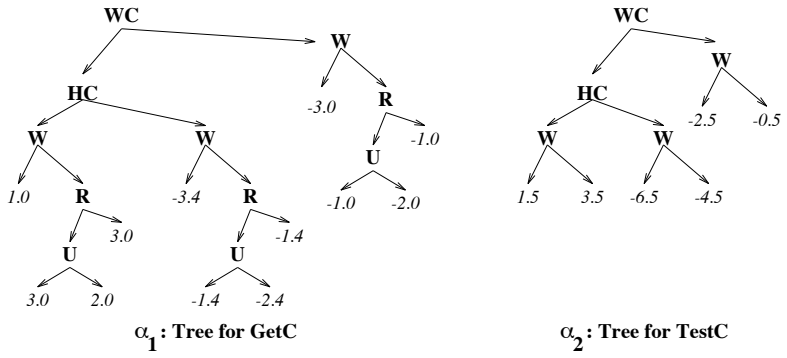

Figure 4: New α-trees for Stage $n-1$

to make further distinctions that influence action cost), the operation will actually be trivial under some common assumptions. For example, if the tree for $R(s)$ is used as the initial α-tree for stage n (as is expected), these distinctions should be preserved in all subsequent value trees. Furthermore, if action cost is constant, this requires a simple addition of this constant to all leaves. Figure 4 shows the expected (total) value tree for the action *GetC* obtained by adding $R(s)$ and $C(a,s)$ to the future value tree obtained from Figure 3. Figure 4 also shows the expected value tree for the action *TestC* under the same assumptions.

These value trees are not generally α-vectors suitable for stage k, for they do not correspond to a fixed observational strategy (unless the strategy assigns the same vector to each observation). To account for observations, we note that every element of $\aleph_k$ corresponds to a given action choice a and an observation strategy that assigns a vector in $\aleph_{k+1}$ to each $o \in \mathit{Obs}(a)$. 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 at stage $k+1$), the new tree α must contain these distinctions. Thus α is partially specified by $Tree(Obs|a)$, the observation tree corresponding to action a. Note 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*. More specifically, at each leaf of $Tree(Obs|a)$ we have a set of possible (nonzero

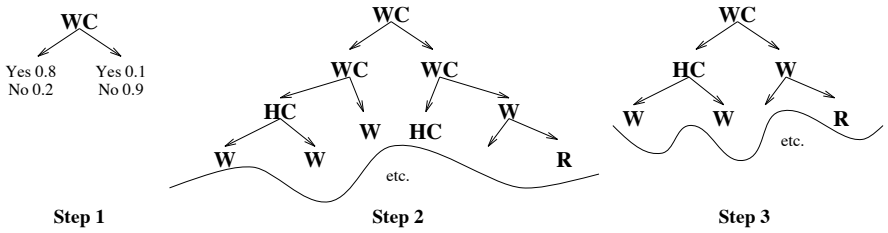

Figure 5: New α-tree for Stage $n-2$

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 $Tree(Obs|a)$.[6]

To make this concrete, consider the following example illustrated in Figure 5. We assume that trees α_1 and α_2, the trees for *GetC* and *TestC* in Figure 4, are elements of $\aleph_k$.[7] We consider generating the new tree α to be placed in $\aleph_{k-1}$ 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 5). 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 5. Of course, we can prune away the inconsistent branches and collapse the redundant nodes to obtain the final tree α, shown at the right of the figure.

[6]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 operation (for example, see [2] where similar tree merging is used for a different purpose). We are exploring heuristic strategies for determining good variable orderings in the merged trees.

[7]In fact, for our particular problem, assuming n stages, these are elements of $\aleph_{n-1}$. Thus, this example shows the generation of an element in $\aleph_{n-2}$.

5.2 Generation of $\aleph_k$

The process described above involves some overhead in the construction of explanation trees and piecing them together with observation probabilities (we do note however that putting together these trees really involves patching together partial trees that exist in the action networks, and is thus not as computationally intensive as it may appear). More substantially, if we were to follow Monahan's algorithm directly, we would have to repeat this process for each action-strategy combination. However, we note that generating the explanation trees for a fixed action and α-tree is independent of the actual strategy adopted; the strategy only tells us how to piece together these explanation trees. This leads to the following informal algorithm for generating $\aleph_k$ from $\aleph_{k+1}$:

(a) For each $\langle a, \alpha \rangle \in \mathcal{A} \times \aleph_{k+1}$, construct the value tree for a, α.

(b) For each action a and strategy s associating elements of $Obs(a)$ with elements of $\aleph_{k+1}$, construct the new α-tree for s. This proceeds by adding the weighted sum (determined by s) of expected value trees (determined in the previous step) to the leaf nodes of the observation tree for a, and deleting redundant nodes and inconsistent branches. Add each such α-tree to $\aleph_k$.

(c) Prune the set $\aleph_k$ by eliminating unnecessary α-trees.

For a fixed action, the explanation tree for a given element of $\aleph_{k+1}$ is thus only computed once. Further savings are possible in piecing together certain strategies (e.g., if s associates the same vector with each observation, the value tree in part (a) can be used directly).

The final stage of the algorithm involves pruning the set of possible α-trees to eliminate those that are dominated (or more accurately, do not dominate some other vector). In other words, if there is no belief state π such that $\pi \cdot \alpha$ is greater than $\pi \cdot \alpha'$ for each $\alpha' \in \aleph_k$, then this vector need never be chosen and has no bearing on the policy. Elimination of such vectors can greatly reduce the number of vectors generated for earlier stages. Monahan suggests a simple set of linear programs for eliminating useless vectors. The tree representation of these vectors allows this elimination phase to become more tractable as well. The LPs used to eliminate vectors have variables corresponding to each state. With our representation, we can reduce the number of variables required to formulate these LPs: we need only make distinctions in the state space that are made by some tree

in the collection $\aleph_k$. This corresponds to finding the smallest subsuming tree for the set $\aleph_k$ and constructing an LP with one variable for each leaf in this tree (see the full paper).

Finally we note that most POMDP algorithms are much more clever about generating the set of possible α-vectors. For example, Sondik's algorithm (and subsequent approaches) do not attempt to 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.

6 Concluding Remarks

We have sketched an algorithm for constructing optimal policies for POMDPs that exploits problem structure (as exhibited by rules of decision trees) to reduce the effective state space at various points in the 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 coalescence of of AI planning and OR optimization techniques (and related approaches) should provide significant improvements in policy construction algorithms. Space limitations preclude a more complete presentation of the algorithm and the example; we refer to the full paper for details.

Directions of current and future research include: applying our ideas to algorithms that enumerate vectors more directly (rather than by exhaustive enumeration and elimination); and empirical evaluation of problem characteristics that dictate whether history-based or belief-state-based policy construction is most effective. We believe than our ideas offers tremendous leverage for approximation methods. The structured representation will allow branches to be collapsed in a way that minimizes lost value, allows vectors to be compared easily, allowing elimination of "similar" vectors, and so on. This should alleviate to a great extent the combinatorial explosion of required vectors as the horizon increases (see also [11]).

Acknowledgements: Thanks to Richard Dearden, Moisés Goldszmidt, and . . . for helpful discussions on this topic. This research was supported by NSERC Research Grants OGP0121843 and OGPOO44121.

References

[1] C. Boutilier, T. Dean, and S. Hanks. Planning under uncertainty: Structural assumptions and computational leverage. (manuscript), 1995.

[2] C. Boutilier, R. Dearden, and M. Goldszmidt. Exploiting structure in policy construction. In *AAAI Spr. Symp. on Extending Theories of Action*, Stanford, 1995.

[3] P. E. Caines. *Linear stochastic systems*. Wiley, New York, 1988.

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

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

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

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

[8] R. A. Howard and J. E. Matheson. Influence diagrams. In R. A. Howard and J. Matheson, editors, *The Principles and Applications of Decision Analysis*, pages 720–762. Strategic Decisions Group, CA, 1981.

[9] W. S. Lovejoy. Computationally feasible bounds for partially observed Markov decision processes. *Operations Research*, 39(1):162–175, 1989.

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

[11] R. Parr and S. Russell. Approximating optimal policies for partially observable stochastic domains. (unpublished manuscript), 1995.

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

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

[14] D. Poole and K. Kanazawa. A decision-theoretic abductive basis for planning. *AAAI Spr. Symp. on Decision-Theoretic Planning*, Stanford, 1994.

[15] 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.

[16] J. E. Smith, S. Holtzman, and J. E. Matheson. Structuring conditional relationships in influence diagrams. *Op. Res.*, 41(2):280–297, 1993.

[17] E. J. Sondik. The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. *Op. Res.*, 26:282–304, 1978.