Toward Approximate Planning in Very Large Stochastic Domains

Ann E. Nicholson* and **Leslie Pack Kaelbling†**
Department of Computer Science
Brown University, Providence, RI 02912
{aen,lpk}@cs.brown.edu

Abstract

In this paper we extend previous work on approximate planning in large stochastic domains by adding the ability to plan in automatically-generated abstract world views. The dynamics of the domain are represented compositionally using a Bayesian network. Sensitivity analysis is performed on the network to identify the aspects of the world upon which success is most highly dependent. An abstract world model is constructed by including only the most relevant aspects of the world. The world view can be refined over time, making the overall planner behave in most cases like an anytime algorithm. This paper is a preliminary report on this ongoing work.

1 Introduction

Many real-world domains cannot be effectively modeled deterministically: the effects of actions vary at random, but with some characterizable distribution. In stochastic domains such as these, a classical plan consisting of a sequence of actions is of little or no use because the appropriate action to take in later steps will depend on the stochastic outcome of previous steps.

The theory of Markov decision processes provides methods for constructing policies (mappings from states to actions) that achieve robust execution by conditioning all actions on the current state of the world. These methods, unfortunately, require enumeration of the state space. Dean *et al.* [Dean *et al.*, 1993] have addressed this problem by arranging for the planner to consider only parts of the state space that are likely to be traversed during the execution of a plan. In very large state spaces, however, there will be a huge number of dimensions of variability in the world, making even the likely state space quite large.

In this paper we introduce techniques for constructing abstract probabilistic world models that allow efficient approximate planning for very large stochastic domains. Domains are specified compositionally, in terms of state variables. The sensitivity of variables involved in the goal to other variables is analyzed, allowing approximate domain models at different levels of abstraction to be constructed. During the planning process, the approximation is refined until it is sufficient for the derivation of a plan at a specified level of reliability.

We begin by giving a brief review of the formal definitions of the Markov decision process (MDP) model, then we consider more compact compositional representations for domains, and present an efficient approximate planning algorithm based on those representations.

2 Markov Decision Processes

The work on Markov decision processes [Bellman, 1957, Howard, 1960], models the entire environment as a stochastic automaton. An MDP is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$, where $\mathcal{S}$ is a finite set of world states that can be reliably identified by the agent; $\mathcal{A}$ is a finite set of actions; T is a state transition model of the environment, which is a function mapping elements of $\mathcal{S} \times \mathcal{A}$ into discrete probability distributions over $\mathcal{S}$; and R is a *reward function* is a mapping from $\mathcal{S}$ to $\Re$, specifying the instantaneous reward that the agent derives from being in each state. We write $T(s_1, a, s_2)$ for the probability that the world will make a transition from state s_1 to state s_2 when action a is taken and we write $R(s)$ for the instantaneous reward derived from being in state s.

A *policy* π is a mapping from $\mathcal{S}$ to $\mathcal{A}$, specifying an action to be taken in each situation. An environment combined with a policy for choosing actions in that environment yields a Markov chain [Kemeny and Snell, 1976]. Given a policy π and a reward function R, the *value* of state $s \in \mathcal{S}$, $V_\pi(s)$, is the sum of the expected values of the rewards to be received at each future time step, discounted by how far into the future they occur. That is, $V_\pi(s) = \sum_{t=0}^{\infty} \gamma^t E(r_t)$, where r_t is the reward received on the tth step of executing policy π after starting in state s. The *discounting factor*, $0 \leq \gamma < 1$, controls the influence of rewards in the distant future. When $\gamma = 0$,

*Ann Nicholson's work was supported by the Advanced Research Projects Agency of the Department of Defense monitored by the Air Force under Contract No. F30602-91-C-0041 and by the National Science foundation in conjunction with the Advanced Research Projects Agency of the Department of Defense under Contract No. IRI-8905436.

†Leslie Kaelbling's work was supported in part by a National Science Foundation National Young Investigator Award IRI-9257592 and in part by ONR Contract N00014-91-4052, ARPA Order 8225.

the value of a state is determined entirely by rewards received on the next step; we are generally interested in problems with a longer horizon and set γ to be near 1. Due to properties of the exponential, the definition of V can be rewritten as

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

We say that policy π *dominates* (is better than) π' if, for all $s \in \mathcal{S}$, $V_\pi(s) \geq V_{\pi'}(s)$, and for at least one $s \in \mathcal{S}$, $V_\pi(s) > V_{\pi'}(s)$. A policy is optimal if it is not dominated by any other policy. Given a Markov decision process and a value for γ, it is possible to compute the optimal policy using either the policy iteration algorithm [Howard, 1960] or the value iteration algorithm [Bellman, 1957].

One of the most common goals is to achieve a certain condition p *as soon as possible.* If we define the reward function as $R(s) = 0$ if p holds in state s and $R(s) = -1$ otherwise, and represent all goal states as being absorbing, then the optimal policy will result in the agent reaching a state satisfying p as soon as possible. A state is *absorbing* if all actions result in that same state with probability 1; that is, $\forall a \in \mathcal{A}$, $\Pr(s, a, s) = 1$. Making the goal states absorbing ensures that we go to the "nearest" state in which p holds, independent of the states that will follow.

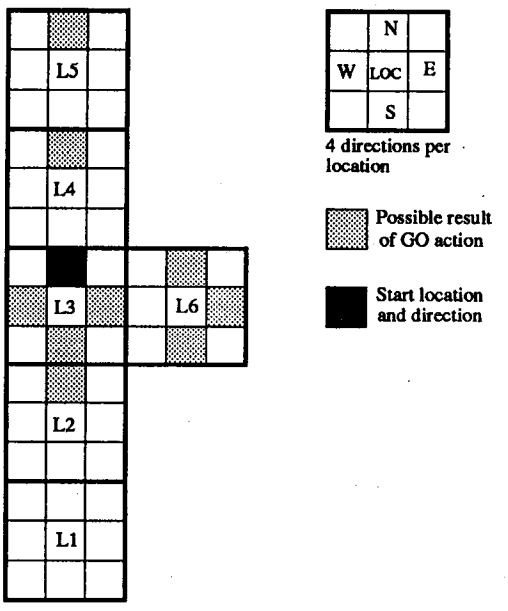

Figure 1: Section of the location floorplan used for examples in this paper. Suppose the robot is in location L3 facing north and executes action GO; the possible location/direction pairings it might end up in are highlighted.

3 Simulated Robotic Domain

In previous work we used high-level mobile-robot path planning as an exmaple domain. The floor plan is divided into a grid of locations, $\mathcal{L}$, with four directional states associated with each location, $\mathcal{D} = \{\text{N, S, E, W}\}$, corresponding to the direction the robot is facing. The robot is given a task to navigate from some starting location to some target location. The actions, $\mathcal{A}$, available to the robot are {STAY, GO, TURN-RIGHT, TURN-LEFT, TURN-ABOUT}. The transition probabilities for the outcome of each action could potentially be obtained empirically through experimentation with a real robot; in this example they are made up. The STAY action is guaranteed to succeed. The probability of success for GO and the turning actions in most locations is 0.8, with the remainder of the probability mass divided between undesired results such as overshooting, over-rotating, slipping sideways, etc. The reward function for the sequential decision problem associated with a given initial and target location assigns 0 to the states corresponding to the target location and -1 to all other states.

In this paper, the robotic domain is extended with a number of extra attributes. The robot has a battery voltage level, $\mathcal{B}$. If the battery voltage is zero, the robot cannot move. If the robot does move, the battery voltage is decreased by one. We also add the action CHARGE, which deterministically results in the battery voltage being raised to the maximum level. In addition, locations can be"cluttered", $\mathcal{C} = \{T, F\}$; if a location is cluttered, then the probability of the GO action taking the robot into the next location is reduced slightly. Finally, there are multiple independent domain variables, such as the temperature, which are not very relevant to the goal at hand (reaching a particular location). A small section of the example domain is shown in Figure 1; there are 6 locations, {L1, L2, L3, L4, L5, L6} and 3 levels of battery voltage {0, 1, 2}; L5 is the only cluttered location.

4 Domain Models

The typical algorithms for working with MDP's represent the state transition function as a matrix and the reward function as a vector, in order to find policies that work on the entire state space. This approach works well in small state spaces, but can very quickly become intractable. In order for very large domains to be amenable to planning, they must have internal regularities that allow them to be represented more compactly. Approaches that take advantage of these regularities typically give up generality; but in their full generality, large planning problems are intractable.

The models used in traditional AI planning, made up of operator descriptions, provide an important starting point for developing compact models of stochastic domains. They start by considering the world state as being composed of a number of state variables (typically propositional), each of which can take on values independently. The state transition function is given compositionally, by specifying, in different rules, how different aspects of the environment change given different actions. Thus, the color of walls need only be mentioned when describing change that occurs as a result of the paint operator.

Work on the Buridan planner [Kushmerick *et al.*, 1993] has extended traditional operator descriptions to describe stochastic transitions. It allows the post-

conditions of an operator to be a probability distribution over possible outcomes, expressed as equivalence classes of world states. This representation is compact, but it can be difficult to describe a situation in which, after doing action a, the new value of variable X depends only on the old value of X and the new value of variable Y depends only on the old value of Y; these *independence* relations can also afford us with savings in model complexity.

A simplified version of the Bayesian network formalism [Pearl, 1988] is well suited to specifying stochastic state transition and reward models. For each action, we use a two-slice network, in which nodes in the first slice represent values of state variables at time t and nodes in the second slice represent values of state variables at time $t+1$. In addition, there is a node in the second slice that represents instantaneous reward at time $t+1$. The value of this node depends deterministically on the values of the nodes to which it is connected; it plays the role of a value node in an influence diagram.[1] Arcs in the diagram indicate probabilistic dependence between variables and are represented, in detail, by conditional probability distributions over values of the target variable given values of the source variables. Figure 2 shows a Bayesian network model for the results of performing the actions GO and one for the turning actions (turn right, left and about all have the same network structure) in the example robot domain described in the previous section, whose states are characterized by the robot's location, direction, battery voltage, and the clutter of the current location.

When there are no arcs between nodes in the second slice, we can say that the attributes of the state at time $t+1$ are *conditionally independent* given the state at time t. It is possible for attributes at time $t+1$ to be dependent on one another, as long as there is no cycle of dependency.

We assume, then, that the state set $\mathcal{S}$ is describable as the cross product of a set of m sets of values V_i, one for each world attribute i. An individual state is described by the vector $\langle v_1, \ldots, v_m \rangle$. The state-transition function of the MDP can be derived from the Bayesian network in the conditionally-independent case as follows:

$$T(\langle x_1, \ldots, x_m \rangle, a, \langle y_1, \ldots, y_m \rangle) = \prod_i \Pr(y_i | a, x_1, \cdots, x_n) \ .$$

For any given y_i term, it is only necessary to condition on x_j if there is an arc from node j in the first slice to node i in the second slice. If the second-slice attributes are not conditionally independent, then because they are acyclic, it is possible to find an ordering of them that allows the following calculation:

$$T(\langle x_1, \ldots, x_m \rangle, a, \langle y_1, \ldots, y_m \rangle) =$$

[1] If it is useful to think of reward as being multi-dimensional, it is possible to have a collection of reward nodes, each with expected value given as a conditional function of some of the state nodes. The reward for any particular state will be the (possibly weighted) sum of the values of the reward nodes.

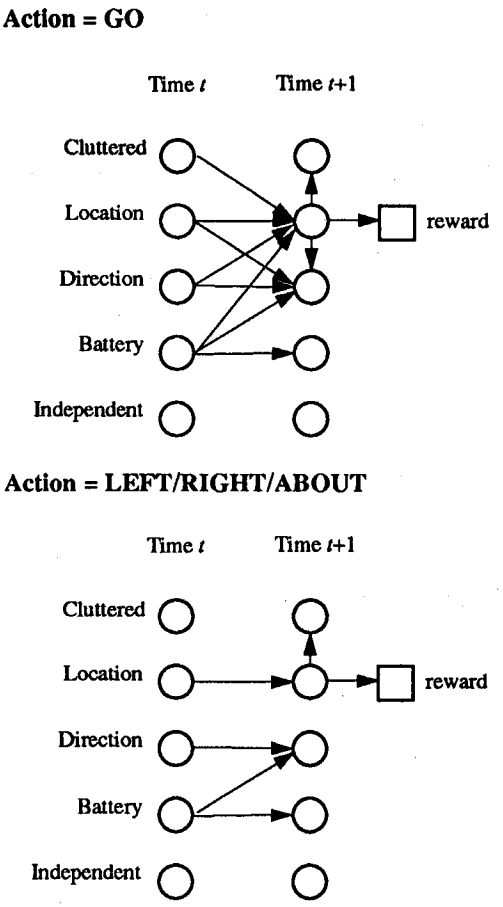

Figure 2: Bayesian network model of simple robotic action

$$\Pr(y^1|a, x_1, \cdots, x_n)$$
$$\prod_{i>1} \Pr(y^i|a, x_1, \cdots, x_n, y^1, \cdots, y^{i-1}) \ .$$

Although it would be possible to compute the entire state transition function from the Bayesian network representation and store it in a table, it will be intractable to do so for domains of the size we are interested. Thus, we just compute instances of the T and R functions when they are required.

5 Abstract World Views

Even with a compact representation of the dynamics of the entire world, we will rarely want or need to work with the whole model. Given different goals, different time constraints, or different current world states, we might want to take very different *views of the world.* In this section, we discuss the construction of different world views by specifying only a subset of the possible attributes in the complete world model. In some cases, these abstract views will capture all of the world dynamics relevant to the problem at hand. In other cases, they will serve as tractable approximations to more complex models.

The idea of using abstract versions of a problem space for efficient planning is by no means new. Polya mentions it as a problem solving heuristic in *How to Solve It* [Polya, 1945]. It was articulated formally and applied to planning by Sacerdoti in his work on ABSTRIPS [Sacerdoti, 1974]. ABSTRIPS planned first in abstract domains that left out preconditions of low criticality. These preconditions (and operators to satisfy them) were gradually added back in to completely flesh out a skeletal plan that was built without them. Our use of abstraction has much the same character, leaving out domain attributes of low criticality and potentially adding them back in over time. We are, however, interested in working domains so large that all of the potentially relevant attributes can never be considered.

5.1 Constructing Abstract World Views

An abstract world view is derived from an MDP and is, itself, an MDP. It is constructed by projecting out some of the dimensions of world state description, so that each state in the abstract view stands for an equivalence class of states in the original world model.

Given an original world $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$ with elements of $\mathcal{S}$ having the form $\langle x_1, \ldots, x_m \rangle$, an abstraction can be specified by giving a set of indices ι to be projected out. The abstract world view is $\langle \mathcal{S}', \mathcal{A}, T', R' \rangle$. The new state set, $\mathcal{S}'$ is constructed from $\mathcal{S}$ by removing from each element attributes whose indices are in ι. In this work, the action set remains the same; in the domains we are currently considering, this set is small and does not introduce any efficiency problems.[2]

Since the original T and R functions are specified with a Bayesian network, we retain this representation, and construct a new Bayesian network for the abstract view. Projecting out a dimension is equivalent to deleting the two corresponding nodes from the network, one from each slice. Let us assume we are projecting out dimension k from a network that does not have links among second-slice nodes. Then we must modify the conditional probability distributions associated with any arcs coming from node k in the first slice. For every node i in the second slice that has a link from k, we define a new conditional probability distribution

$$\Pr(y_i|x_1, \cdots, x_{k-1}, x_{k+1}, \cdots, x_m) = \sum_{x_k} \Pr(x_k) \Pr(y_i|x_1, \cdots, x_m) \ .$$

Now we are faced with the problem of needing to know a prior distribution on the values of dimension k. In this work, we assume a uniform distribution.[3] A slightly more complex form of this basic technique applies to the reward nodes and to the case in which there are links among the second-slice nodes.

5.2 Sensitivity Analysis

In order to determine which abstract views of a world may be most useful for solving a particular problem, we can perform a sensitivity analysis on the Bayesian network model. Sensitivity analysis determines the degree to which the conditional probability distribution of a particular node in the second slice will be affected by the distribution of a node in the first slice. Knowing this will allow us to determine useful abstract views.

Given a Bayesian network model to represent the dynamics of a particular action, $a \in \mathcal{A}$, let $\sigma_a(N, N')$ be the sensitivity of node N in the second slice to a parent node N'. It is, intuitively, the difference between the distribution of values at node N when the values at node N' are taken into account and the distribution of values at node N when node N' is assumed to take on all of its values according to some fixed distribution.

Let $\mathcal{N}$ be the set of *parents* of N (nodes with links to N). We can compute the sensitivity of N to any node $N' \in \mathcal{N}$. First, let Φ be the set of possible valuations of the nodes nodes $\mathcal{N} - \{N'\}$, and let n' range over the values of N'. Then

$$\sigma_a(N, N') = \frac{1}{|\Phi|} \sum_{\phi \in \Phi} \sum_{n'} \Pr(n') D(\Pr(N|n', \phi), \Pr(N|\phi)) \ ,$$

where D is a measure of distance between two distributions. As a distance measure, we use the *information-theoretical distance* (also known as the Kullback-Leibler information distance)[Rényi, 1984] as follows:

$$D(P, Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)} \ .$$

It measures the distance between two distributions by taking the expectation (according to the first distribution) of the difference of the log probabilities at every

[2] We might want, eventually, to represent $\mathcal{A}$ compositionally in order, for example, to be able to characterize agents with independent, simultaneous output modalities, such as moving and speaking.

[3] Of course, this might be a very bad approximation, because we don't know the relative probabilities of being in the elements of the equivalence class.

N	N'	$\sigma(N, N')$
$\mathcal{L}$	$\mathcal{L}$	0.220377
$\mathcal{L}$	$\mathcal{D}$	0.014027
$\mathcal{L}$	$\mathcal{B}$	0.009335
$\mathcal{L}$	$\mathcal{C}$	0.004814
$\mathcal{D}$	$\mathcal{L}$	0.015208
$\mathcal{D}$	$\mathcal{D}$	0.170770
$\mathcal{D}$	$\mathcal{B}$	0.079002
$\mathcal{D}$	$\mathcal{C}$	0
$\mathcal{B}$	$\mathcal{L}$	0
$\mathcal{B}$	$\mathcal{D}$	0
$\mathcal{B}$	$\mathcal{B}$	0.174207
$\mathcal{B}$	$\mathcal{C}$	0
$\mathcal{C}$	$\mathcal{L}$	0.016360
$\mathcal{C}$	$\mathcal{D}$	0
$\mathcal{C}$	$\mathcal{B}$	0
$\mathcal{C}$	$\mathcal{C}$	0

Table 1: Sensitivity of domain variables to each other for the robot example

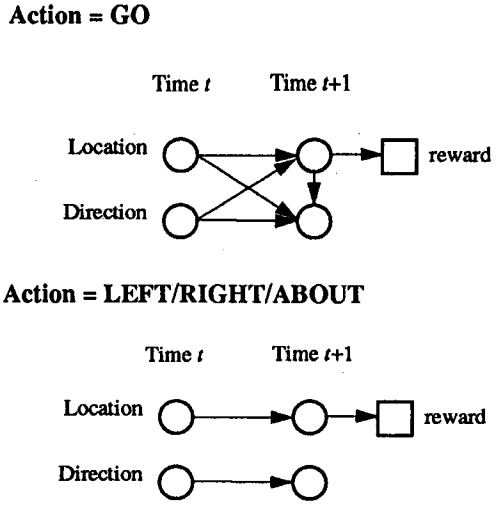

Figure 3: Abstracted Bayesian network model of simple robotic actions

point. This measure is suited to the situation in which P is the true value of a quantity and Q is an estimate; in this case, it makes sense to take an expected difference with respect to the distribution P. This value is always non-negative; if the two nodes are independent, then the sensitivity will be zero. The more the values of N' affect the values of N, the higher the sensitivity value will be.

The overall sensitivity $\sigma(N, N')$ of domain variable N to domain variable N' is the average over all actions:

$$\sigma(N, N') = \frac{1}{|\mathcal{A}|} \sum_a \sigma_a(N, N')$$

5.3 Example of World View Abstraction

We performed the sensitivity analysis on the example robot domain represented by the Bayesian network model given in Figure 2, and obtained the sensitivities shown in Table 1.

The location is most sensitive to the previous location by an order of magnitude, then in order of decreasing sensitivity to the direction, the battery level and the clutter of the current location. If more locations are cluttered, or if the battery has more capacity, the relative sensitivity of location to the battery level and the clutter is reversed. The direction is most sensitive to the previous direction and the battery level; the lower sensitivity to the location reflects the low probability outcome where there the robot slips and turns while performing a GO action. As expected for the domain specification, the battery level is only sensitive to the previous battery level, and the clutter variable is only sensitive to the location.

The location domain variable is the one we are most interested in, as it is the only one influencing the reward node. The sensitivity analysis suggests constructing an abstract world view containing the location and direction domain variables. Applying this abstraction to the world modeled in Figure 2 results in the abstract world view shown in Figure 3.

6 Planning With Approximate World Models

In this section we outline the basic approximate planning algorithm of Dean *et al.* [Dean *et al.*, 1993], then show how we can extend the notion of planning in restricted state spaces by planning in restricted versions of *abstract world views*. The work by Dean *et al.* presented two planning models: a *precursor* model in which all planning is performed before any execution begins and a *recurrent model* in which planning and execution take place in parallel. The techniques described in this paper are equally applicable to both models, but for simplicity in presentation we consider only the precursor model.

An important property of this planning algorithm is that it works by constructing successive approximations to the optimal policy. Although there are rare cases in which approximations can get worse over time, in most cases this algorithm behaves like an *anytime algorithm* [Dean and Boddy, 1988], generating better and better policies the longer it has to compute. It can be interrupted at any time and a policy will be available. The extensions we are making to the original algorithm preserve this property.

6.1 Original Algorithm

The original planning algorithm attempts to reduce the complexity of generating a policy by considering only a subset of the state space of the MDP. The algorithm starts with an initial policy and a restricted state space (or *envelope*), extends that envelope, and then computes a new policy. In the precuror model, the planner constructs a policy that is followed by the agent until a new goal must be pursued or until the agent falls out of the current envelope. In the simple case, a deadline is specified indicating when planning stops and execution begins.

A *partial policy* is a mapping from a subset of $\mathcal{S}$ into actions; the domain of a partial policy π is called its *envelope*, $\mathcal{E}_\pi$. The *fringe* of a partial policy, $\mathcal{F}_\pi$, is the

set of states that are not in the envelope of the policy, but that may be reached in one step of policy execution from some state in the envelope. That is, $\mathcal{F}_\pi = \{s \in \mathcal{S} - \mathcal{E}_\pi \mid \exists s' \in \mathcal{E}_\pi \; s.t. \; T(s', \pi(s'), s) > 0\}$.

To construct a restricted MDP, we take an envelope $\mathcal{E}$ of states and add the distinguished state OUT. For any states s and s' in $\mathcal{E}$ and action a in $\mathcal{A}$, the transition probabilities remain the same. Further, for every $s \in \mathcal{E}$ and $a \in \mathcal{A}$, we define the probability of going out of the envelope as

$$T(s, a, \text{OUT}) = 1 - \sum_{s' \in \mathcal{E}} T(s, a, s') \ .$$

The OUT state is absorbing.

The cost of falling out of the envelope is a parameter that depends on the domain. If it is possible to re-invoke the planner when the agent falls out of the envelope, then one approach is to assign $V(\text{OUT})$ to be the estimated value of the state into which the agent fell minus some function of the time required to construct a new partial policy. Under the reward function described earlier, the value of a state is negative, and its magnitude is the expected number of steps to the goal; if time spent planning is to be penalized, it can simply be added to the magnitude of the value of the OUT state with a suitable weighting function.

The high level planning algorithm, given a description of the environment and start state s_0 or a distribution over start states, is as follows:

1. Generate an initial envelope $\mathcal{E}$
2. While $(\mathcal{E} \neq \mathcal{S})$ and (not deadline) do
   a. Extend the envelope $\mathcal{E}$
   b. Generate an optimal policy π for restricted automaton with state set $\mathcal{E} \cup \{\text{OUT}\}$
3. Return π

The algorithm first finds a small subset of world states and calculates an optimal policy over those states. Then it gradually adds new states in order to make the policy robust by decreasing the chance of falling out of the envelope. After new states are added, the optimal policy over the new envelope is calculated. Note the interdependence of these steps: the choice of which states to add during envelope extension may depend on the current policy, and the policy generated as a result of optimization may be quite different depending on which states were added to the envelope. The algorithm terminates when a deadline has been reached or when the envelope has been expanded to include the entire state space.

6.2 Using Abstract World Views

Even though the planning technique described above considers a potentially very small subset of the state space, it considers fully-specified world states, which may make distinctions that are not important to the planning task at hand. By restricting the world view before applying the envelope-based algorithm, we can potentially drastically reduce the number of states in the envelope. With an abstract world view, each of the states in the envelope stands for an equivalence class of world states that can be treated the same for the current purposes. The extended planning algorithm can be described as follows:

1. Generate an initial world view $\langle \mathcal{S}', \mathcal{A}, T', R' \rangle$
2. Generate an initial envelope $\mathcal{E} \subset \mathcal{S}'$
3. While $(\mathcal{E} \neq \mathcal{S}')$ and (not deadline) do
   - Extend the envelope $\mathcal{E}$
   - Generate an optimal policy π for restricted automaton with state set $\mathcal{E} \cup \{\text{OUT}\}$
   - If the current world view is insufficient then
     - Expand the world view
     - Go to step 2
4. Return π

This algorithm works by initially making a fairly gross approximation to the real world dynamics, which allows it to quickly derive a partial policy that is of some utility, though perhaps not as good as desired. If time remains, the world view is refined and new policies are constructed within the refined world view. The best policy from the previous world view is always retained, so that if time runs out before a good policy can be found in the new world view, the previous policy can be returned for execution.

Off-line, a sensitivity analysis can be performed, revealing the sensitivity of the reward node to each of the state attribute nodes. The initial world view is constructed by including only those state attributes to which the reward node is most sensitive (determined by a threshold). If this view proves to be insufficient for planning, then a new view is constructed by adding the state attribute with the next most sensitivity.

The question, then, is how to determine that the current world view is insufficient for planning. We are currently investigating several possible criteria. When the estimated value of the starting state is not improving through envelope expansion or when the quality of the best solution solution does not exceed some threshold that may indicate that a less abstract world view is needed. It may also be possible to learn when to change views by statistical learning in the same way that we learn envelope extension strategies in the original work.

7 Future Work

Obviously, our first goal is to get the entire system implemented and running, and to perform empirical tests of the performance of this planning algorithm. In particular, we expect to have to experiment with criteria for deciding when to refine the current abstract world view.

As the algorithm currently stands, when we add a dimension, we add it to the entire state space. We are considering methods that would allow different abstractions over different parts of the state space, capturing the idea that whether or not a particular door is open is only relevant in certain parts of a large domain.

Finally, we are interested in learning the domain model and perhaps also the sensitivity information, eventually building an integrated learning and planning system.

References

[Bellman, 1957] Bellman, Richard 1957. *Dynamic Programming.* Princeton University Press, Princeton, New Jersey.

[Dean and Boddy, 1988] Dean, Thomas and Boddy, Mark 1988. An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, Minneapolis-St. Paul, Minnesota.

[Dean *et al.*, 1993] Dean, Thomas; Kaelbling, Leslie Pack; Kirman, Jak; and Nicholson, Ann 1993. Planning with deadlines in stochastic domains. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, Washington, DC.

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

[Kemeny and Snell, 1976] Kemeny, John G. and Snell, J. Laurie 1976. *Finite Markov Chains.* Springer-Verlag, New York.

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

[Pearl, 1988] Pearl, Judea 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, San Mateo, California.

[Polya, 1945] Polya, George 1945. *How to Solve It.* Princeton university Press, Princeton, New Jersey.

[Rényi, 1984] Rényi, Alfréd 1984. *A Diary on Information Theory.* John Wiley and Sons, New York, New York.

[Sacerdoti, 1974] Sacerdoti, Earl D. 1974. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence* 5:115–135.