Planning With Deadlines in Stochastic Domains

Thomas Dean, Leslie Pack Kaelbling, Jak Kirman, Ann Nicholson
Department of Computer Science
Brown University, Providence, RI 02912
tld@cs.brown.edu

Abstract

We provide a method, based on the theory of Markov decision problems, for efficient planning in stochastic domains. Goals are encoded as reward functions, expressing the desirability of each world state; the planner must find a policy (mapping from states to actions) that maximizes future rewards. Standard goals of achievement, as well as goals of maintenance and prioritized combinations of goals, can be specified in this way. An optimal policy can be found using existing methods, but these methods are at best polynomial in the number of states in the domain, where the number of states is exponential in the number of propositions (or state variables). By using information about the starting state, the reward function, and the transition probabilities of the domain, we can restrict the planner's attention to a set of world states that are likely to be encountered in satisfying the goal. Furthermore, the planner can generate more or less complete plans depending on the time it has available. We describe experiments involving a mobile robotics application and consider the problem of scheduling different phases of the planning algorithm given time constraints.

Introduction

In a completely deterministic world, it is possible for a planner simply to generate a sequence of actions, knowing that if they are executed in the proper order, the goal will necessarily result. In nondeterministic worlds, planners must address the question of what to do when things do not go as expected.

The method of triangle tables [Fikes *et al.*, 1972] made plans that could be executed robustly in any circumstance along the nominal trajectory of world states, allowing for certain classes of failures and serendipitous events. It is often the case, however, that an execution error will move the world to a situation that has not been previously considered by the planner. Many systems (SIPE, for example [Wilkins, 1988]) can monitor for plan "failures" and initiate replanning. Replanning is often too slow to be useful in time-critical domains, however. Schoppers, in his universal plans [Schoppers, 1987], gives a method for generating a reaction for every possible situation that could transpire during plan execution; these plans are robust and fast to execute, but can be very large and expensive to generate. There is an inherent contradiction in all of these approaches. The world is assumed to be deterministic for the purpose of planning, but its nondeterminism is accounted for by performing execution monitoring or by generating reactions for world states not on the nominal planned trajectory.

In this paper, we address the problem of planning in nondeterministic domains by taking nondeterminism into account from the very start. There is already a well-explored body of theory and algorithms addressing the question of finding optimal policies (universal plans) for nondeterministic domains. Unfortunately, these methods are impractical in large state spaces. However, if we know the start state, and have a model of the nature of the world's nondeterminism, we can restrict the planner's attention to a set of world states that are likely to be encountered on the way to the goal. Furthermore, the planner can generate more or less complete plans depending on the time it has available. In this way, we provide efficient methods, based on existing techniques of finding optimal strategies, for planning under time constraints in non-deterministic domains. Our approach addresses uncertainty resulting from control error, but not sensor error; we assume certainty in observations.

We assume that the environment can be modeled as a stochastic automaton: a set of states, a set of actions, and a matrix of transition probabilities. In the simplest cases, achieving a goal corresponds to performing a sequence of actions that results in a state satisfying some proposition. Since we cannot guarantee the length of a sequence needed to achieve a given goal in a stochastic domain, we are interested in building planning systems that minimize the expected number of actions needed to reach a given goal.

In our approach, constructing a plan to achieve a goal corresponds to finding a *policy* (a mapping from states to actions) that maximizes expected performance. Performance is based on the expected accumulated reward over sequences of state transitions determined by the underlying stochastic automaton. The rewards are determined by a *reward function* (a mapping from states to the real numbers) specially formulated for a given goal. A good policy in our framework corresponds to a universal plan for achieving goals quickly on average.

In the following, we refer to the automaton modeling the environment as the *system* automaton. Instead of generating the optimal policy for the whole system automaton, we formulate a simpler or *restricted* stochastic automaton and then search for an optimal policy in this restricted automaton. The state space for the restricted automaton, called the *envelope*, is a subset of the states of the system automaton, augmented with a special state OUT that represents being in any state outside of the envelope.

The algorithm developed in this paper consists of two basic subroutines. *Envelope extension* adds states to the restricted automaton, making it approximate the system automaton more closely. *Policy generation* computes an optimal policy for the restricted automaton; a complete policy for the system automaton is constructed by augmenting the policy for the restricted automaton with a set of default actions or *reflexes* to be executed for states outside the envelope.

The algorithm is implemented as an *anytime* algorithm [Dean and Boddy, 1988], one that can be interrupted at any point during execution to return an answer whose value at least in certain classes of stochastic processes improves in expectation as a function of the computation time. We gather statistics on how envelope extension and policy generation improve performance and use these statistics to compile expectations for allocating computing resources in time-critical situations.

In this paper, we focus primarily on the details of the algorithm and the results of a series of computational experiments that provide some indication of its merit. Subsequent papers will expand on the representation for goals and deal with more complicated models of interaction that require more sophisticated methods for allocating computational resources.

Planning Algorithm

Definitions We model the entire environment as a stochastic automaton. Let $\mathcal{S}$ be the finite set of world states; we assume that they can be reliably identified by the agent. Let $\mathcal{A}$ be the finite set of actions; every action can be taken in every state. The transition model of the environment is a function mapping elements of $\mathcal{S} \times \mathcal{A}$ into discrete probability distributions over $\mathcal{S}$. We write $\text{Pr}(s_1, a, s_2)$ for the probability that the world will make a transition from state s_1 to state s_2 when action a is taken.

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, 1960].

A *reward function* is a mapping from $\mathcal{S}$ to $\Re$, specifying the instantaneous reward that the agent derives from being in each state. 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*, γ, is between 0 and 1 and controls the influence of rewards in the distant future. Due to properties of the exponential, the definition of V can be rewritten as

$$V_\pi(s) = R(s) + \gamma \sum_{s' \in \mathcal{S}} \text{Pr}(s, \pi(s), s') V_\pi(s') \quad . \qquad (1)$$

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.

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. *Absorbing* means that all actions result in the same state with probability 1; $\forall a \in \mathcal{A}$, $\text{Pr}(s, a, s) = 1$. Making the goal states absorbing ensures that we go to the "nearest" state in which p holds, independent of the states that will follow. The language of reward functions is quite rich, allowing us to specify much more complex goals, including the maintenance of properties of the world and prioritized combinations of primitive goals.

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, F_π, 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, $F_\pi = \{s \in \mathcal{S} \mid \exists s' \in \mathcal{E}_\pi \; s.t. \; \text{Pr}(s', \pi(s'), s) > 0\}$.

To construct a restricted automaton, 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

$$\text{Pr}(s, a, \text{OUT}) = 1 - \sum_{s' \in \mathcal{E}} \text{Pr}(s, a, s') \quad .$$

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 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.

Overall Structure We assume, initially, that there are two separate phases of operation: planning and execution. 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. More sophisticated models of interaction between planning and execution are possible, including one in which the planner runs concurrently with the execution, sending down new or expanded strategies as they are developed. Questions of how to schedule deliberation are discussed in the following section (see also [Dean *et al.*, 1993]). Execution of an explicit policy is trivial, so we describe only the algorithm for generating policies.

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

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

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 envelopé. The algorithm terminates when a deadline has been reached or when the envelope has been expanded to include the entire state space. In the following sections, we consider each subcomponent of this algorithm in more detail.

Generating an initial envelope This high-level algorithm works no matter how the initial envelope is chosen, but if it is done with some intelligence, the early policies are much more useful. In our examples, we consider the goal of being in a state satisfying p as soon as possible. For such simple goals of achievement, a good initial envelope is one containing a chain of states from the initial state, s_0, to some state satisfying p such that, for each state, there is some action with a non-zero probability of moving to the next state in the chain.

In the implemented system, we generate an initial envelope by doing a depth-first search from s_0 considering the most probable outcome for each action in decreasing order of probability. This yields a set of states that can be traversed with fairly high probability to a goal state. More sophisticated techniques could be used to generate a good initial envelope; our strategy is to spend as little time as possible doing this, so that a plausible policy is available as soon as possible.

Generating an optimal policy Howard's *policy iteration* algorithm is guaranteed to generate the optimal policy for the restricted automaton. The algorithm works as follows: [1]

```
1. Let $\pi'$ be any policy on $\mathcal{E}$
2. While $\pi \neq \pi'$ do loop
   2.1 $\pi := \pi'$
   2.2 For all $s \in \mathcal{E}$, calculate $V_\pi(s)$ by solving the set of $|\mathcal{E}|$ linear equations in $|\mathcal{E}|$ unknowns given by equation 1
   2.3 For all $s \in \mathcal{E}$, if there is some action $a \in \mathcal{A}$ s.t. $[R(s) + \gamma \sum_{s' \in \mathcal{E} \cup \{\text{OUT}\}} \text{PR}(s, s', a) V_\pi(s')] > V_\pi(s)$, then $\pi'(s) := a$; otherwise $\pi'(s) := \pi(s)$
3. Return $\pi$
```

The algorithm iterates, generating at every step a policy that strictly dominates the previous policy, and terminates when a policy can no longer be improved, yielding an optimal policy. In every iteration, the values of the states under the current policy are computed. This is done by solving a system of equations; although this is potentially an $O(|\mathcal{E}|^{2.8})$ operation, most realistic environments cannot transition from every state to every other, so the transition matrix is sparse, allowing much more efficient solution of the equations. The algorithm then improves the policy by looking for states s in which doing some action a other than $\pi(s)$ for one step, then continuing with π, would result in higher expected reward than simply executing π. When such a state is found, the policy is changed so that it always chooses action a in that state.

This algorithm requires a number of iterations at most polynomial in the number of states; in practice for for an instance of our domain with 6000 world states, it has never taken more than 16 iterations. When we use this as a subroutine in our planning algorithm, we generate a random policy for the first step, and then for

[1] Since $V(\text{OUT})$ is fixed, and the OUT state is absorbing, it does not need to be explicitly included in the policy calculations.

all subsequent steps we use the old policy as the starting point for policy iteration. Because, in general, the policy does not change radically when the envelope is extended, it requires very few iterations (typically 2 or 3) of the policy iteration algorithm to generate the optimal policy for the extended envelope. Occasionally, when a very dire consequence or an exceptional new path is discovered, the whole policy must be changed.

Extending the envelope There are a number of possible strategies for extending the envelope; the most appropriate depends on the domain. The aim of the envelope extension is to judiciously broaden the subset of the world states, by including states that are outside the envelope of the current policy but that may be reached upon executing the policy. One simple strategy is to add the entire fringe of the current policy, F_π; this would result in adding states uniformly around the current envelope. It will often be the case, however, that some of the states in the fringe are very unlikely given the current policy.

A more reasonable strategy, similar to one advocated by Drummond and Bresina [Drummond and Bresina, 1990], is to look for the N most likely fringe states. We do this by simulating the restricted automaton and accumulating the probabilities of falling out into each fringe state. We then have a choice of strategies. We can add each of the N most likely fringe states. Alternatively, for goals of achievement, we can take each element of this subset of the fringe states and find a chain of states that leads back to some state in the envelope. In the experiments described in the following sections, fringe states are added rather than whole paths back to the envelope.

Example In our approach, unlike that of Drummond and Bresina, extending the current policy is coupled tightly and naturally to *changing* the policy as required to keep it optimal with respect to the restricted view of the world. The following example illustrates how such changes are made using the algorithm as described.

The example domain is mobile-robot path planning. The floor plan is divided into a grid of 166 locations, $\mathcal{L}$, with four directional states associated with each location, $\mathcal{D} = \{\mathrm{N, S, E, W}\}$, corresponding to the direction the robot is facing, resulting in a total of 664 world states. The actions available to the robot are {STAY, GO, TURN-RIGHT, TURN-LEFT, TURN-ABOUT}. The transition probabilities for the outcome of each action may be obtained empirically. In our experimental simulation the STAY action is guaranteed to succeed. The probability of success for GO and turning actions in most locations were 0.8, with the remainder of the probability mass divided between undesired results such as overshooting, over-rotating, slipping sideways, etc. The world also contains *sinks*, locations that are difficult or impossible to leave. On average each state has 15.6 successors.

Figure 1 shows a subset of our domain, the locations surrounding a stairwell, which is a *complete sink*, *i.e.*, there are no non-zero transitions out of it; also, it is only accessible from one direction, north. In this figure there are four small squares associated with each location, one for each possible heading; thus each small square corresponds to a state, the direction of the arrow shows the policy for the robot in that location and with that heading. Figure 1 (a) shows the optimal policy for a small early envelope; Figures 1(b) and (c) show two subsequent envelopes where the policy changes to direct the robot to circumvent the stairwell, reflecting aversion to the risk involved in taking the shortest path.

Deliberation Scheduling

Given the two-stage algorithm for generating policies provided in the previous section, we would like the agent to allocate processor time to the two stages when faced with a time critical situation. Determining such allocations is called *deliberation scheduling* [Dean and Boddy, 1988]. In this paper, we consider situations in which the agent is given a deadline and an initial state and has until the deadline to produce a policy after which no further adjustments to the policy are allowed. The interval of time from the current time until the deadline is called the *deliberation interval.* We address more complicated situations in [Dean *et al.*, 1993].

Deliberation scheduling relies on compiling statistics to produce expectations regarding performance improvements that are used to guide scheduling. In general, we cannot guarantee that our algorithm will produce a sequence of policies, $\hat{\pi}_0, \hat{\pi}_1, \hat{\pi}_2, \ldots$, that increase in value, *e.g.*, $V_{\hat{\pi}_0}(s_0) < V_{\hat{\pi}_1}(s_0) < V_{\hat{\pi}_2}(s_0) \cdots$, where the $\hat{\pi}_i$ are complete policies constructed by adding reflexes to the partial policies generated by our algorithm. The best we can hope for is that the algorithm produces a sequence of policies whose values increase in expectation, *e.g.*, $E[V_{\hat{\pi}_0}(s_0)] < E[V_{\hat{\pi}_1}(s_0)] < E[V_{\hat{\pi}_2}(s_0)] \cdots$, where here the initial state s_0 is considered a random variable. In allocating processor time, we are concerned with the expected improvement, $E[V_{\hat{\pi}_{i+1}}(s_0) - V_{\hat{\pi}_i}(s_0)]$, relative to a given allocation of processor time.

If envelope extension did not make use of the current policy, we could just partition the deliberation interval into two subintervals, the first spent in envelope extension and the second in policy generation. However, since the two stages are mutually dependent, we have to consider performing multiple rounds where each round involves some amount of envelope extension followed by some amount of policy generation.

Let t_{EE_i} (t_{PG_i}) be the time allocated to envelope extension (policy generation) in the ith round of the algorithm and $\mathcal{E}_i$ be the envelope following the ith round envelope extension. To obtain an optimal deliberation schedule, we would have to consider the expected value

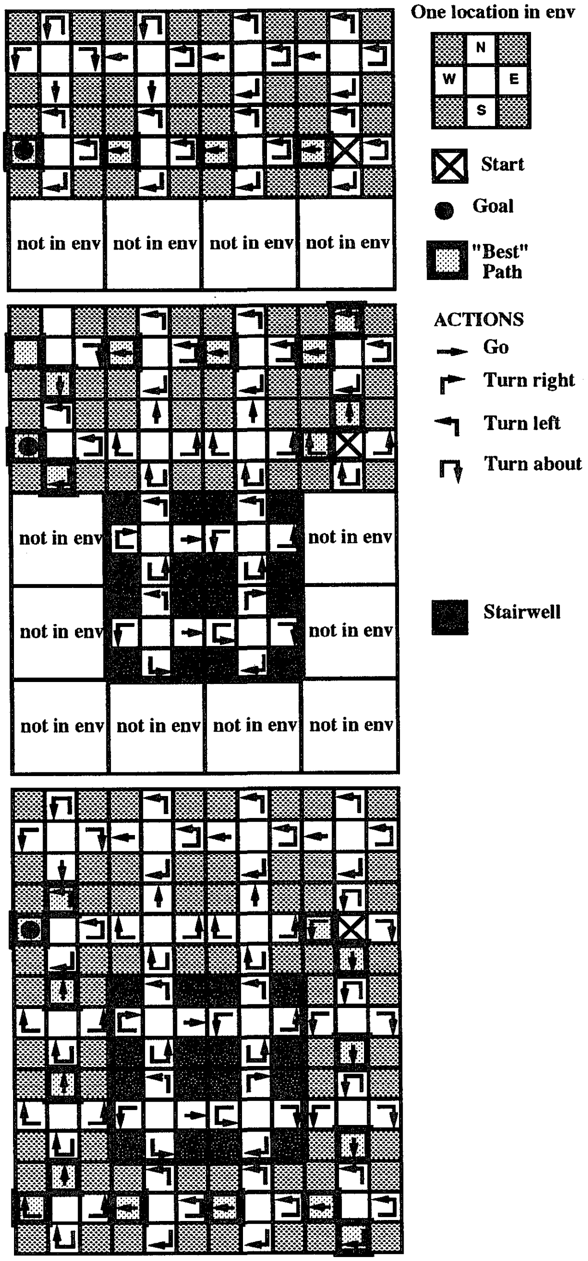

Figure 1: Example of policy change for different envelopes near a complete sink. The direction of the arrow indicates the current policy for that state. (a) Sink not in the envelope: the policy chooses the straightforward shortest path. (b) Sink included: the policy skirts north around it. (c) All states surrounding the stairwell included: the barriers on the south, east and west sides allow the policy take a longer but safer path. For this run $\gamma = 0.999999$ and $V(\text{OUT}) = -4000$.

of the final policy given $k = 1, 2, \ldots$ rounds and all possible allocations to t_{EE_i} and t_{PG_i} for $1 \leq i \leq k$. We suspect that finding the optimal deliberation schedule is NP-hard. To expedite deliberation scheduling, we use a greedy algorithm based on the following statistics.

1. The expected improvement starting with an envelope of size m and adding n states: $E[V_{\hat{\pi}_{i+1}}(s_0) - V_{\hat{\pi}_i}(s_0) \mid m = \mid \mathcal{E}_i \mid, m + n = \mid \mathcal{E}_{i+1} \mid]$.
2. The expected time required to extend by n states an envelope of size m and compute the optimal policy for the resulting restricted automaton: $E[t_{EE_i + PG_i} \mid m = \mid \mathcal{E}_i \mid, m + n = \mid \mathcal{E}_{i+1} \mid]$.

After each round of envelope extension followed by policy generation we have an envelope of some size m; we find that n maximizing the ratio of (1) and (2), add n states, and perform another round, time permitting. If the deadline occurs during envelope extension, then the algorithm returns the policy from the last round. If the deadline occurs during policy generation, then the algorithm returns the policy from the last iteration of policy iteration.

Results

In this section, we present results from the iterative refinement algorithm using the table lookup deliberation scheduling strategy and statistics described in the previous sections. We generated 1.6 million data points to compute the required statistics for the same robot-path-planning domain. The start and goal states were chosen randomly for executions of the planning algorithm using a greedy deliberation strategy, where N, the number of fringe states added for each phase of envelope extension, was determined from the deliberation scheduling statistics.

We compared the performance of (1) our planning algorithm using the greedy deliberation strategy to (2) policy iteration optimizing the policy for the whole domain. Our results show that the planning algorithm using the greedy deliberation strategy supplies a good policy early, and typically converges to a policy that is close to optimal before the whole domain policy iteration method does. Figure 2 shows average results from 620 runs, where a single run involves a particular start state and goal state. The graph shows the average improvement of the start state under the policy available at time t, $V_{\hat{\pi}}(s_0)$, as a function of time. In order to compare results from different start/goal runs, we show the average ratio of the value of the current policy to the value of the optimal policy for the whole domain, plotted against the ratio of actual time to the time, T_{opt}, that the policy iteration takes to reach that optimal value.

The greedy deliberation strategy performs significantly better than the standard optimization method. We also considered very simple strategies such as adding a small fixed N each iteration, and adding the

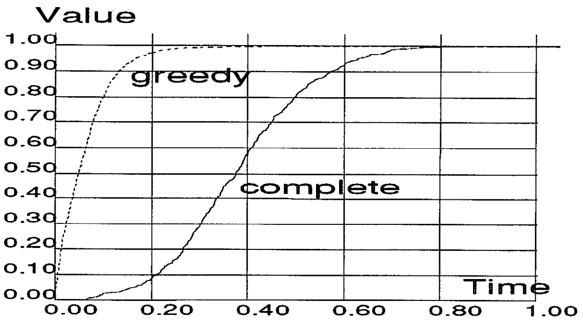

Figure 2: Comparison of planning algorithm using greedy deliberation strategy (dashed line) with the policy iteration optimization method (solid line).

whole fringe each iteration, which performed fairly well for this domain, but not as well as the greedy policy. Further experimentation is required to draw definitive conclusions about the comparative performance of these deliberation strategies for particular domains.

Related Work and Conclusions

Our primary interest is in applying the sequential decision making techniques of Bellman [Bellman, 1957] and Howard [Howard, 1960] in time-critical applications. Our initial motivation for the methods discussed here came from the 'anytime synthetic projection' work of Drummond and Bresina. [Drummond and Bresina, 1990]. We improve on the Drummond and Bresina work by providing (i) coherent semantics for goals in stochastic domains, (ii) theoretically sound probabilistic foundations, (iii) and decision-theoretic methods for controlling inference.

The approach described in this paper represents a particular instance of time-dependent planning [Dean and Boddy, 1988] and borrows from, among others, Horvitz' [Horvitz, 1988] approach to flexible computation. Boddy [Boddy, 1991] describes solutions to related problems involving dynamic programming. Hansson and Mayer's BPS (Bayesian Problem Solver) [Hansson and Mayer, 1989] supports general state-space search with decision-theoretic control of inference; it may be that BPS could be used as the basis for envelope extension thus providing more fine-grained decision-theoretic control. Christiansen and Goldberg [Christiansen and Goldberg, 1990] also address the problem of planning in stochastic domains.

The approach is applicable to stochastic domains with certain characteristics; typically there are multiple paths to the goal and the domain is relatively benign. If there is only one path to the goal all the work will be done by the procedure finding the initial envelope, and extending the envelope only improves the policy if the new states can be recovered from. Our future research plans involve extending the approach in several directions: allowing more complex goals; performing more complicated deliberation scheduling such as integrating online deliberation in parallel with the execution of policies; relaxing the assumption of observation certainty to handle sensor error.

Acknowledgements. Thomas Dean's work was supported in part by a National Science Foundation Presidential Young Investigator Award IRI-8957601, 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.

References

Bellman, R. 1957. *Dynamic Programming.* Princeton University Press.

Boddy, M. 1991. Anytime problem solving using dynamic programming. In *Proceedings AAAI-91.* AAAI. 738–743.

Christiansen, A., and Goldberg, K. 1990. Robotic manipulation planning with stochastic actions. In *DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control.* San Diego,California.

Dean, T., and Boddy, M. 1988. An analysis of time-dependent planning. In *Proceedings AAAI-88.* AAAI. 49–54.

Dean, T.; Kaelbling, L.; Kirman, J.; and Nicholson, A. 1993. Deliberation scheduling for time-critical sequential decision making. Submitted to *Ninth Conference on Uncertainty in Artificial Intelligence.*

Drummond, M., and Bresina, J. 1990. Anytime synthetic projection: Maximizing the probability of goal satisfaction. In *Proceedings AAAI-90.* AAAI. 138–144.

Fikes, R. E.; Hart, P. E.; and Nilsson, N. J. 1972. Learning and executing generalized robot plans. *Artificial Intelligence* 3:251–288.

Hansson, O., and Mayer, A. 1989. Heuristic search as evidential reasoning. In *Proceedings of the Fifth Workshop on Uncertainty in AI.* 152–161.

Horvitz, E. J. 1988. Reasoning under varying and uncertain resource constraints. In *Proceedings AAAI-88.* AAAI. 111–116.

Howard, R. A. 1960. *Dynamic Programming and Markov Processes.* MIT Press, Cambridge, Massachusetts.

Kemeny, J. G. and Snell, J. L. 1960. *Finite Markov Chains.* D. Van Nostrand, New York.

Schoppers, M. J. 1987. Universal plans for reactive robots in unpredictable environments. In *Proceedings IJCAI 10.* IJCAII. 1039–1046.

Wilkins, D. E. 1988. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan-Kaufmann, Los Altos, California.