Planning Under Time Constraints in Stochastic Domains

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

Abstract

We provide a method, based on the theory of Markov decision processes, 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 require time 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 restrict the planner's attention to a set of world states that are likely to be encountered in satisfying the goal. Using this restricted set of states, the planner can generate more or less complete plans depending on the time it has available.

Our approach employs several iterative refinement routines for solving different aspects of the decision making problem. We describe the meta-level control problem of *deliberation scheduling*, allocating computational resources to these routines. We provide different models corresponding to optimization problems that capture the different circumstances and computational strategies for decision making under time constraints. We consider *precursor* models in which all decision making is performed prior to execution and *recurrent* models in which decision making is performed in parallel with execution, accounting for the states observed during execution and anticipating future states. We describe experimental results for both the precursor and recurrent problems that demonstrate planning times that grow slowly as a function of domain size and compare their performance to other relevant algorithms.

1 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 and Nilsson, 1971] 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 nondeterministic domains. Our approach addresses the uncertainty resulting from control error, but not sensor error; in most of the following, we assume certainty in observations, but discuss relaxing this assumption in Section 8.

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.

There are dynamic programming algorithms for computing the optimal policy given a stochastic model of the world. They are useful in small to medium-sized state-spaces, but become intractable on very large state-spaces. We address this difficulty by making some informal assumptions about the environments in which we are working that allow us to generate approximate solutions efficiently. In particular, we assume that the environment has the following properties:

- *high solution density*: it is relatively easy to find plausible (though perhaps not optimal) solutions
- *low dispersion rate*: from any given state, there are only a few states to which transitions can be made
- *continuity*: it is reasonable to estimate the values of states by considering the values of near-by states (where distance is measured as the expected number of steps between states)

Many large, realistic planning problems, such as those involving high-level navigation, have these properties.

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.

There are two basic types of operations on the restricted automaton. The first is called *envelope alteration* and serves to increase or decrease the number of states in the restricted automaton. The second is called *policy generation* and determines a policy for the system automaton using the restricted automaton. Note that, although the policy is constructed using the restricted automaton, it is a complete policy and applies to all of the states in the system automaton. For states outside of the envelope, the policy is defined by a set of *reflexes* that implement some default behavior for the agent.

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. In this paper, *deliberation scheduling* refers to the problem of allocating processor time to envelope alteration and policy generation. We gather statistics on how envelope alteration and policy generation improve performance and use these statistics to compile expectations for allocating computational resources in time-critical situations.

We consider several decision models for deliberation scheduling. In the simpler models, called *precursor-deliberation* models, we assume that the agent has one opportunity to generate a policy and that, having generated a policy, the agent must use that policy thereafter. In more complicated models, called *recurrent-deliberation* models, we assume that the agent periodically replans and executes the resulting policy in parallel with planning the next policy.

Our approach is motivated by the intuitively appealing work of Drummond and Bresina on 'anytime synthetic projection' [Drummond and Bresina, 1990]. In this paper, we reformulate their basic framework in terms of Markov decision processes (MDPs), cast the algorithmic issues in terms of approximations to specific optimization problems, provide a disciplined approach to allocating computational resources at run time, introduce techniques for specifying goals in stochastic domains, and describe how to extend the framework to deal with uncertainty in observation. In Section 9.1, we provide a more detailed comparison of our approach with that of Drummond and Bresina.

The MDP process model has been the basis of a large amount of work in the reinforcement-learning community. In the context of model-based learning, in which an MDP model of the world is learned, it is necessary to generate a policy from a model. Sutton's DYNA system [Sutton, 1990] explores this connection between planning and learning, and uses a version of value iteration to interleave model learning and policy updating. Barto, Bradtke, and Singh [Barto *et al.*, 1993] investigate this approach further, developing the RTDP (Real-Time Dynamic Programming) algorithm, which has the same fundamental motivations as this work. In Section 6.3.1 we describe the RTDP algorithm and in Section 9.2 we describe the relation of our work to other recent developments in reinforcement learning and real-time search.

The structure of this paper is as follows. We begin with an introduction to stochastic decision making in Section 2, then define the structures necessary to cope with large state spaces in Section 3. In Section 4 we present basic algorithms for a variety of envelope alterations and consider how iterative refinement versions of these algorithms may be used as anytime algorithms to provide expected improvements in value. Idealized decision models for precursor and recurrent deliberation are presented in Section 5; experimental results from the domain of robot path planning are given in Section 6. In Section 7 we show how the language of reward functions can be used to specify complex goals, including goals of achievement, maintenance of properties of the world, and prioritized combinations of primitive goals. In Section 8 we outline extensions of our approach: handling uncertainty in observation, based on the theory of partially observable Markov processes; exploring compositional representation of the state space and state transition model; and dealing with domain in which the number of actions is very large. The research presented in this paper is based on sequential decision making, stochastic control, and reinforcement learning; we review this work in Section 9.

2 Markov Decision Models

Following the work on Markov decision processes [Bellman, 1957, Bertsekas, 1987], 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 $\Pr(s_1, a, s_2)$ 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 $\mathbb{R}$, 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*, $0 \leq \gamma < 1$, controls the influence of rewards in the distant future. When $\gamma = 0$, 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

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

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. 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. 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; this is explored in Section 7.

Given a state-transition model, a reward function, 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]. We use the policy iteration algorithm because it is guaranteed to converge in a finite number of steps—generally a small number of steps in the domains that we have experimented with—and thus simplifies debugging our computational experiments. The policy iteration algorithm works as follows:

1. Let π' be any policy on $\mathcal{S}$
2. While $\pi \neq \pi'$ do loop
   a. $\pi := \pi'$
   b. For all $s \in \mathcal{S}$, calculate $V_\pi(s)$ by solving the set of $|\mathcal{S}|$ linear equations in $|\mathcal{S}|$ unknowns given by equation 1
   c. For all $s \in \mathcal{S}$, if there is some action $a \in \mathcal{A}$ such that $[R(s) + \gamma \sum_{s' \in \mathcal{S}} \Pr(s, a, s') V_\pi(s')] > V_\pi(s)$, then $\pi'(s) := a$; otherwise $\pi'(s) := \pi(s)$
3. Return π

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, which requires time on the order of $|\mathcal{S}|^3$. 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. The algorithm is guaranteed to converge to an optimal policy in a number of iterations polynomial in $|\mathcal{S}|$.

3 Coping with Large State Spaces

As the size of our state spaces grows, even a polynomial-time algorithm such as policy iteration becomes too inefficient. We will assume that our environment is such that, for any given reward function and initial starting state, it is sufficient to consider a highly restricted subset of the entire state space in our planning.

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, $\mathcal{F}_\pi = \{s \in \mathcal{S} - \mathcal{E}_\pi \mid \exists s' \in \mathcal{E}_\pi, \Pr(s', \pi(s'), s) > 0\}$.

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

$$\Pr(s, a, \textsc{out}) = 1 - \sum_{s' \in \mathcal{E}} \Pr(s, a, s') \enspace .$$

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 $V(\textsc{out})$ 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.

4 Basic Algorithms

As a concession to complexity, in generating a policy, our algorithms consider only a subset of the state space of the stochastic process. The algorithms start with an initial policy and a restricted state space (or *envelope*), extend that envelope, and then compute a new policy. We would like it to be the case that the new policy π' is an improvement over (or at the very least no worse than) the old policy π in the sense that $V_{\pi'}(s_0) - V_\pi(s_0) \geq 0$.

In general, however, we cannot guarantee that the policy will improve without extending the state space to be the entire space of the system automaton, which results in computational problems. The best that we can hope for is that the algorithms improve in *expectation*. Suppose that the initial envelope is just the initial state and the initial policy

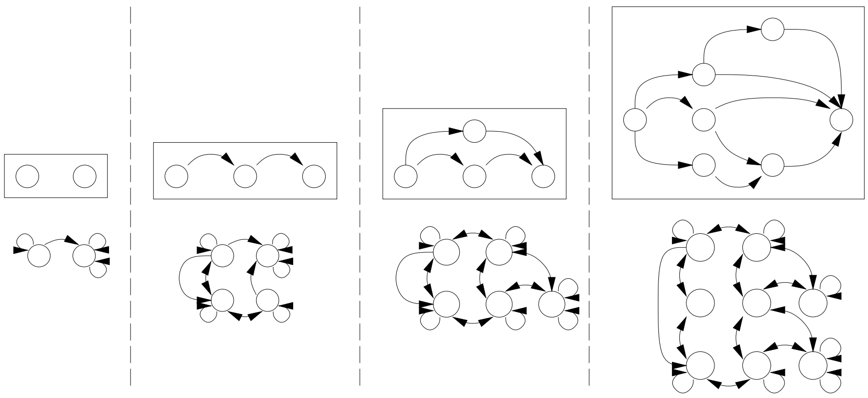

Figure 1: Sequence of restricted automata and associated paths through state space

is determined entirely by the reflexes. The difference $V_{\pi'}(s_0) - V_\pi(s_0)$ is a random variable, where π is the reflex policy and π' is the computed policy. We would like it to be the case that $E[V_{\pi'}(s_0) - V_\pi(s_0)] > 0$, where the expectation is taken over start states and goals drawn from some fixed distribution. Although it is possible to construct system automata for which even this improvement in expectation is impossible, we believe many moderately benign environments are well-behaved in this respect. In particular, *navigation* environments (excluding mazes) in which transitions are restricted by spatio-temporal constraints generally satisfy our requirements.

Our basic algorithm consists of two stages: envelope alteration followed by policy generation. The algorithm takes an envelope and a policy as input and generates as output a new envelope and policy. We assume that the algorithm has access to the state transition matrix for the stochastic process. In general, we assume that the algorithm is applied in the manner of iterative refinement, with more than one invocation of the algorithm. We also treat envelope alteration and policy generation as separate, so we cast the overall process of policy formation in terms of some number of rounds of envelope alteration followed by policy generation, resulting in a sequence of policies. Figure 1 depicts a sequence of automata generated by iterative refinement along with corresponding paths through state space from the initial state to a goal state.

Policy generation is itself an iterative algorithm that improves an initial policy by estimating the value of policies with respect to the restricted-state-space stochastic process mentioned earlier. When run to completion, policy generation continues to iterate until it

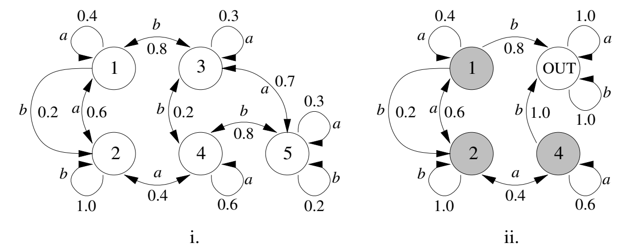

Figure 2: Stochastic process and a restricted version

finds a policy that it cannot improve with respect to its estimate of value; this policy is guaranteed to be optimal with respect to the restricted-state-space stochastic process.

Figure 2.i shows an example system automaton consisting of five states. Suppose that the initial state is 1, and state 4 satisfies the goal. The path $1 \xrightarrow{a} 2 \xrightarrow{a} 4$ goes from the initial state to a state satisfying the goal and the corresponding envelope is $\{1, 2, 4\}$. Figure 2.ii shows the restricted automaton for that envelope. Let $\pi(s)$ be the action specified by the policy π to be taken in state s; the optimal policy for the restricted automaton shown in Figure 2.ii is defined by $\pi(1) = \pi(2) = \pi(4) = a$ on the states of the envelope and the reflexes by $\pi(\mathrm{OUT}) = b$ (*i.e.*, $\forall s \notin \{1, 2, 4\}, \pi(s) = b$) (the reflex actions need not be the same in all states).

First we consider the high level algorithms for a precursor and a recurrent model of planning and execution. Execution of an explicit policy is trivial, so we describe only the algorithm for generating policies. We then look at the various component phases of the planning algorithms, which are shared between both models. A wider range of precursor and recurrent models are described in Section 5, where we consider how to schedule deliberation in both models.

4.1 High Level Algorithms

4.1.1 Precursor Deliberation Model

In the precursor deliberation model, 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. In the simplest precursor models, a deadline is specified indicating when planning stops and execution begins.

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
   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, in which case it will be optimal following step 2.

4.1.2 Recurrent Deliberation Model

A more sophisticated model of interaction between planning and execution is one in which the planner runs concurrently with the execution, sending new or expanded strategies to the executor as they are developed.

In recurrent-deliberation models, the agent has to repeatedly decide how to allocate time to deliberation, taking into account new information obtained during execution. The details of such models are discussed in Section 5; here we provide just a rough sketch. We assume two separate modules: one for planning and a second for execution. In the simplest model, the planner and executor operate in a rigid cycle with a period of fixed length of

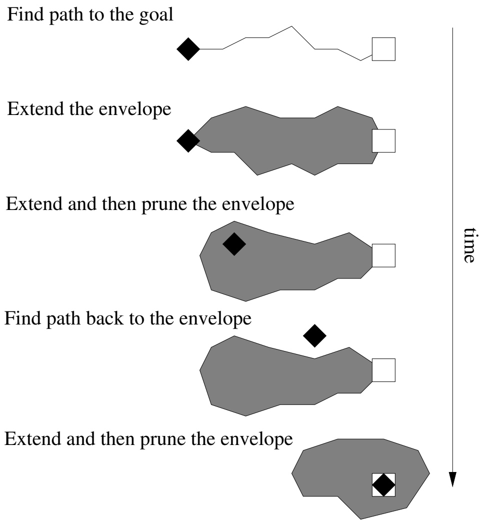

Figure 3: Typical sequence of changes to the envelope

time. At the beginning of each cycle, the planner is given the current state by the execution module; the planner spends the fixed length of time working on a new policy; at the end of the fixed time, the planner gives the new policy to the execution module. For the time being we assume that the execution module can identify the current state with certainty; in Section 8 we consider the case in which there is uncertainty in observation.

In the recurrent models, it is often necessary to remove states from the envelope in order to lower the computational costs of generating policies from the restricted automata. For instance, in the mobile-robot domain, it may be appropriate to remove states corresponding to portions of a path the robot has already traversed if there is little chance of returning to those states. Figure 3 shows a typical sequence of changes to the envelope corresponding to the state space for the restricted automaton. The current state is indicated by ◆ and the goal state is indicated by □.

The recurrent planning algorithm, given a description of the environment, the policy π_c that is currently being followed by the agent, and the state of the agent at the beginning of the planning interval, s_c, is as follows:

While (not goal) do

1. Set s_c to be the current state for planning purposes

2. While (not end of current planning interval) do
   a. Extend the envelope $\mathcal{E}$
   b. Prune the envelope $\mathcal{E}$
   c. Generate an optimal policy π' for restricted automaton with state set $\mathcal{E} \cup \{\textsc{out}\}$
3. Set π_c to be the new policy π'

The details of the extension and pruning of the envelope will depend on the agent's expected state at the end of the planning interval. This can be determined from the state transitions, s_c and π_c by forward simulation.

4.2 Algorithm Components

In the following sections, we consider each component of the precursor and recurrent algorithms in more detail. Each of the components can be implemented as an anytime algorithm; in Section 5, we cast the problem of allocating computational resources to the components as an optimization problem and then describe decision-theoretic techniques to compute approximations.

4.2.1 Policy Generation

Given a restricted automaton with envelope $\mathcal{E}$, we use the policy iteration algorithm to generate the optimal policy. Although this is potentially an $O(|\mathcal{E}|^3)$ 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. This algorithm requires a number of iterations at most polynomial in the number of states; in practice, in our robot path planning domains with 660 to 16,000 world states, it has never taken more than 15 to 50 iterations respectively. When we use this as a subroutine in our planning algorithm, we generate a plausible policy for the first step, and then for 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 of the policy iteration algorithm to generate the optimal policy for the extended envelope (typically 2 or 3 iterations for the smaller domains, up to 10 for the larger domains). Occasionally, when a very dire consequence or an exceptional new path is discovered, the whole policy must be changed.

4.2.2 Initial Trajectory Planning

The high-level precursor and recurrent algorithms work no matter how the initial envelope is chosen, but if it is done with some intelligence, the early policies are much more useful, and the time taken to reach the goal is shorter. 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 a path from start to goal 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 to a goal state. We then check for any shortcuts within this path, deleting intermediate states if this does not decrease the probability of reaching the goal for the resultant path. We then attempt to improve the robustness of the path by adding a successor to a path state if it in turn has the next state in the path as its successor and the combined transition probabilities are higher than the original single transition probability.

We use this method to generate a small number of paths from the start to the goal, say 10, and choose the shortest path (which is usually the path with the highest probability) to form the initial envelope to the policy. We can then use the nominal path from the start to goal as the initial policy, which makes the optimization of the initial envelope much faster than if we began with a completely random policy for the envelope. More sophisticated techniques or more complicated heuristics 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. It also might be appropriate to use Kushmerick *et al.*'s method for generating plausible initial policies [Kushmerick *et al.*, 1993].

4.2.3 Envelope Alteration

Envelope alteration can be classified in terms of three basic operations on the envelope: *trajectory planning*, *envelope extension*, and *envelope pruning*. Trajectory planning consists of searching for a path from some initial state to a state satisfying the goal; this method need not make use of the current restricted automaton. Envelope extension adds states to the envelope. Envelope pruning removes states from the envelope and is generally used only in recurrent-deliberation models. Both envelope extension and envelope pruning will typically make use of the current restricted automaton; for example, envelope extension may add those the states outside of the envelope that the agent is most likely to reach given the current policy, and pruning may delete states that the agent is unlikely to end up in.

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, $\mathcal{F}_\pi$; 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 to be reached 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. This determines the probability of reaching each fringe state, starting from the current state and using the current policy; the fringe states are ranked by these probabilities.

We then have a choice of strategies. One possibility is to 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 path from the state back to some state in the envelope. We call this class of envelope extension methods *strengthening* (it is referred to as *robustification* by Drummond and Bresina). Strengthening may also be combined with trajectory planning by taking a fringe state and adding a path to a state that satisfies the goal.

Trajectory planning Trajectory planning is performed in much the same way as initial trajectory planning except that the notions of initial and goal states may vary. For instance, we often wish to find a path (trajectory) from some fringe state back to some state inside the envelope or back to a state satisfying the goal. Apart from this difference, the actual search techniques are exactly the same as in initial trajectory planning.

Pruning In the recurrent models, it is often necessary to remove states from the envelope in order to lower the computational costs of generating policies from the restricted automaton. One obvious method is to prune states from the current envelope on the grounds that the agent is unlikely to end up in those states and therefore need not consider them in formulating a policy. However we need to be careful about how this is done. It may be that a state has a low instantaneous reward, or is some kind of sink (*e.g.,* all non-self transitions have low probability). In these situations the current policy will direct the agent away from that state, resulting in a low probability of that state being reached. We do not always want to remove this kind of state; its presence in the envelope directs the agent away from an area it should avoid. We want to distinguish between states that have a low probability of being reached because they are in an area to be avoided but are still somehow between the agent and the goal, and those which the agent has gone past. In order to prune the latter category, for the results given in Section 6, we prune the N least likely states which also have a lower value than the value of the current state.

4.3 Example: Strengthening in the Precursor Model

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 precursor algorithm described above.

The example domain is high-level mobile-robot path planning. It was chosen so that it would be easy to understand the policies generated by our algorithms. 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, representing the layout of the fourth floor of the Brown University Computer Science department (see Figure 4). The robot is given a task to navigate from some starting location to some target location. 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 is 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

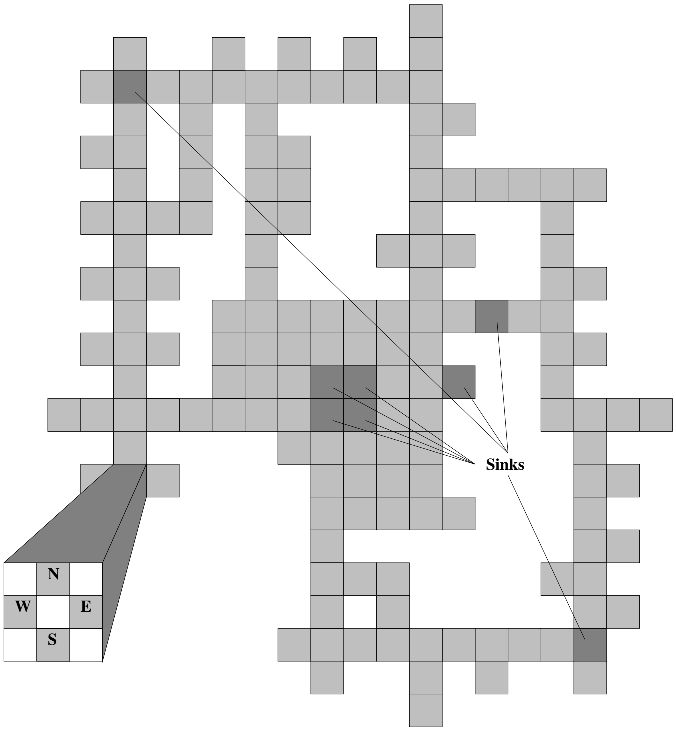

Figure 4: Plan of the Brown CS department fourth floor used for the mobile-robot domain. Each square is a location corresponding to four states, one for each heading of the robot, as shown in the expanded square in the lower left. Sinks are indicated by darker shading. The larger domains are constructed by replicating this map in a grid layout.

sinks, locations that are difficult or impossible to leave. In the mobile-robot domain, a sink might correspond to a stairwell that the robot could fall into. The reward function for the sequential decision problem associated with a given initial and target location assigns 0 to the four states corresponding to the target location and -1 to all other states. On average each state has 3.5 possible successors (8.5 for the GO action). this is the type of low dispersion rate domain we identified earlier as suitable for our approach.

Figure 5 shows a subset of the domain corresponding to the locations surrounding a stairwell. The stairwell states taken as a whole correspond to what we call a *complete sink*; there are no nonzero transitions out of a complete sink. The stairwell states are only accessible from one direction, the 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 5 (a) shows the optimal policy for a small early envelope; Figures 5(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.

5 Deliberation Scheduling

Deliberation scheduling is the problem of allocating processor time to envelope alteration and policy generation. It is natural to think of deliberation scheduling in terms of optimization even if the combinatorics dictate that an optimal solution is not computationally feasible. Having said this, it still remains to determine what optimization problem we are trying to solve. We have to specify exactly what options are allowed and what information is available; such a characterization is generally referred to as a *decision model.*

In formulating a precise decision model, we are following standard practice in the decision sciences: make explicit the options for action and the variables and objective functions that influence performance. For example, if you are going to make assumptions regarding how far in the future you are willing to consider during planning, then it is important to make those assumptions explicit. The problem of formulating a precise decision model for a time-critical decision problem is even more important since the decision model determines the computational cost of optimal deliberation scheduling. In this section, we provide some insight into the space of possible decision models and describe some of the particular decision models that influenced the design of our prototype planning system.

In the following, we present a number of decision models. It should be pointed out that for each instance of the problems that we consider, there is a large number of possible decision models. By specifying different decision models, we can make deliberation scheduling easy or hard. Our selection of which decision models to investigate is guided by our interest in providing insight into the problems of time-critical decision making and our anticipation of the combinatorial problems involved in deliberation scheduling. In this section, we ignore the time spent in deliberation scheduling; for practical reasons, however, we are interested in decision models for which the on-line time spent in deliberation scheduling is negligible.

In the simpler *precursor-deliberation* models, we assume that the agent has one oppor-

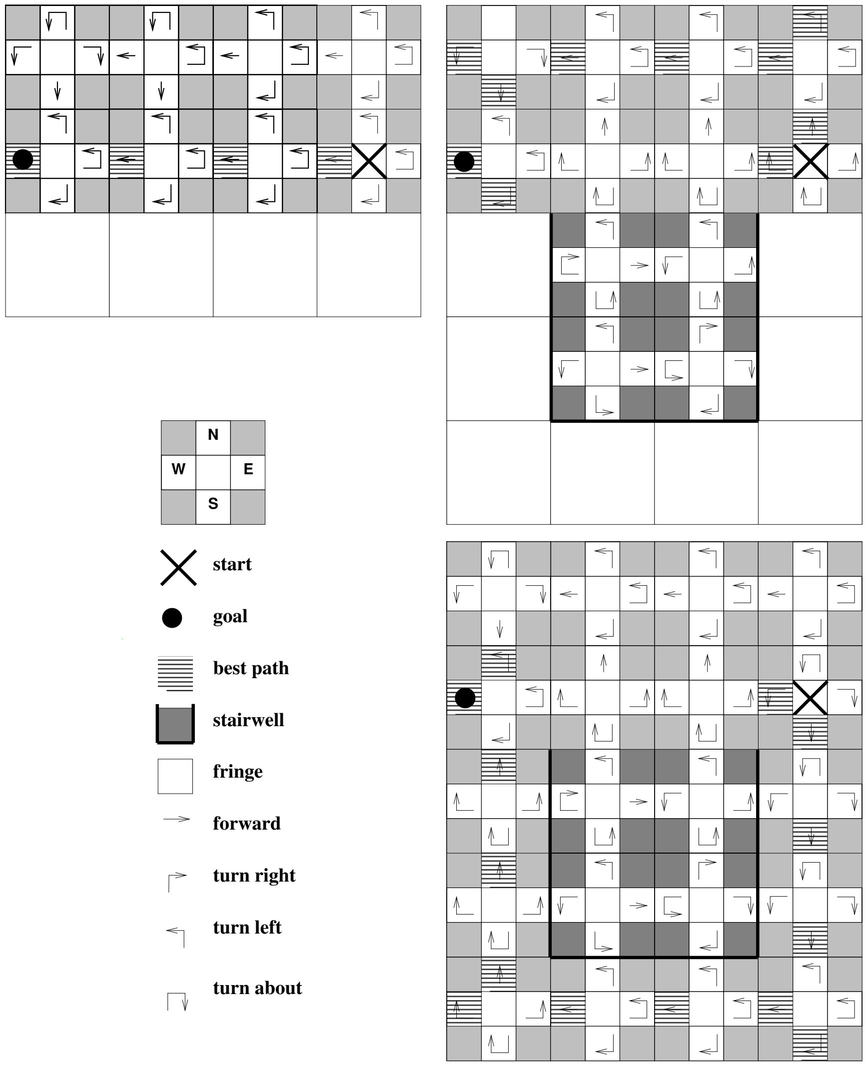

Figure 5: Example of policy change for different envelopes near a complete sink. The direction of the arrow indicates the current policy for that state. (a) Sink not in the envelope: the policy chooses the straightforward shortest path. (b) Sink included: the policy skirts north around it. (c) All states surrounding the stairwell included: the barriers on the south, east and west sides allow the policy to take a longer but safer path. For this run γ = 0.999999 and V(OUT) = -4000.

tunity to generate a policy and that having generated a policy the agent has to stick to that policy thereafter. Precursor-deliberation models include those in which

1. a deadline is given in advance specifying when to stop deliberating and start acting according to the generated policy (an algorithm for this was given in Section 4.1.1)

2. the agent is given an unlimited amount of time to respond with a time cost of delay specified as a fixed function

In the following two models, there is a trigger event that occurs indicating that the agent must begin following its policy immediately with no further refinement.

3. the trigger event can occur at any time in a fixed interval with a uniform distribution

4. the trigger event is governed by a more complicated distribution, *e.g.,* a normal distribution centered on an expected time

Models (3) and (4) are not considered in this paper; models (1) and (2) are treated in Section 5.1.

In the more complicated *recurrent-deliberation* models, we assume that the agent periodically replans. Recurrent-deliberation models include those in which

5. the agent performs further envelope alteration and policy generation if and only if it 'falls out' of the envelope defined by the current restricted automaton

6. the agent performs further envelope alteration and policy generation in parallel with execution, tailoring the restricted automaton and its corresponding policy to states anticipated in the near future (the algorithm for this was given in Section 4.1.2)

Model (6) is treated in Section 5.2. Model (5) corresponds to a standard AI strategy of replanning or plan repair on failure; we consider this model empirically in Section 6.3.

5.1 Precursor-Deliberation Models

In the following we consider four cases of precursor deliberation with known deadlines and one case of precursor deliberation with unlimited time to respond and a cost for delay. Let

t_{TOT} be the total amount of time from the current time until the deadline. If there are k rounds of envelope alteration and policy generation, then we have

$$t_{EA_1} + t_{PG_1} + \cdots + t_{EA_k} + t_{PG_k} = t_{TOT},$$

where t_{EA_i} (t_{PG_i}) is the time spent in the ith round of envelope alteration (policy generation). Let π_i represent the policy after the ith round of envelope alteration followed by policy generation. We say that policy generation is *inflexible* if the ith round of policy generation is always *run to completion* on the restricted automaton available at the ith round.

Single round; inflexible policy generation; deadlines In the simplest case, policy generation does not inform envelope alteration and so we might as well do all of the envelope alteration before policy generation, and

$$t_{EA_1} + t_{PG_1} = t_{TOT}.$$

In order to schedule time for EA_1 and PG_1, we need:

1. the expected improvement of the value of a random initial state between the reflex policy and the policy resulting from policy generation given a fixed amount of time allocated to envelope alteration, $\mathrm{E}[V_{\pi_1}(s_0) - V_{\pi_0}(s_0) | t_{EA_1}]$;
2. the expected size of the envelope given the time allocated to the first round of envelope alteration, $\mathrm{E}[\,|\mathcal{E}_1|\,|t_{EA_1}]$; and
3. the expected time required for policy generation given the size of the envelope after the first round of envelope alteration, $\mathrm{E}[t_{PG_1}|\,|\mathcal{E}_1|\,]$.

Each of (1), (2) and (3) can be determined empirically, and, at least in principle, the optimal allocations to envelope alteration and policy generation can be determined. If we assume no variance in run times and envelope sizes, then optimal deliberation scheduling corresponds to finding that t_{EA_1} maximizing $\mathrm{E}[V_{\pi_1}(s_0) - V_{\pi_0}(s_0)|t_{EA_1}]$ subject to the constraint that

$$t_{EA_1} + \mathrm{E}[t_{PG_1}|\mathrm{E}[\,|\mathcal{E}_1|\,|t_{EA_1}]] \leq t_{TOT}.$$

Note that, because policy generation is itself an iterative refinement algorithm, we can interrupt it at any point to obtain a policy. Although the particular model considered here assumes inflexible policy generation for the purpose of deliberation scheduling, we might use a more flexible approach to handling of deadlines at run time.

In the case of nondegenerate distributions over run times and envelope sizes, optimal deliberation scheduling would require consideration of cases in which the actual run times violate the specified deadline. This is relatively straightforward to model, but considerably more difficult to implement.

Multiple rounds; inflexible policy generation; deadlines Assume that policy generation can profitably inform envelope alteration, *i.e.,* that the policy after round i provides guidance in extending the environment during round $i+1$. In this case, we have k rounds and

$$t_{EA_1} + t_{PG_1} + \cdots + t_{EA_k} + t_{PG_k} = t_{TOT}.$$

Recall that the fringe states for a given envelope and policy correspond to those states outside the envelope that can be reached with a non zero probability in a single step by following the policy starting from some state within the envelope. Let the most likely *falling-out* state with respect to a given envelope and policy correspond to that fringe state that is most likely to be the first fringe state reached by following the policy starting in the initial state. We might consider a very simple method of envelope alteration in which we just add the most likely falling-out state and then the next most likely and so on. Suppose that adding each additional state takes a fixed amount of time. Let

$$\mathrm{E}[V_{\pi_i}(s_0) - V_{\pi_{i-1}}(s_0) | \, |\mathcal{E}_{i-1}| = m, |\mathcal{E}_i| = m+n]$$

denote the expected improvement in the value of the initial state after the ith round of envelope alteration and policy generation given that there are n states added to the m states that were already in the envelope after the $i-1$ round.

Again, the expectations described above can be obtained empirically. Coupled with the sort of expectations described for the previous single-round case (*e.g.,* $\mathrm{E}[t_{PG_i} | \, |\mathcal{E}_i|]$) , one could, at least in principle, determine the optimal number of rounds k and the allocations to t_{EA_i} and t_{PG_i} for $1 \leq j \leq k$. In practice, we use slightly different statistics and heuristic methods for deliberation scheduling to avoid the combinatorics.

Single round; flexible policy generation; deadlines Actually, this case is simpler in concept than the case with inflexible policy generation assuming that we can compile the following statistics.

$$\mathrm{E}[V_{\pi_1}(s_0) - V_{\pi_0}(s_0)|t_{EA_1}, t_{PG_1}]$$

Multiple rounds; flexible policy generation; deadlines Again, with additional statistics, *e.g.*,

$$\mathrm{E}[V_{\pi_i}(s_0) - V_{\pi_{i-1}}(s_0)|\,|\mathcal{E}_{i-1}| = m, |\mathcal{E}_i| = m + n, t_{PG_{i-1}}],$$

this case is not much more difficult than the earlier cases.

Single round; inflexible policy generation; cost of delay Deliberation models that assume no fixed deadline but specify a time cost of delay as a fixed function can be handled similarly to the cases considered above. For instance, in the case of single round, inflexible policy generation, if we assume no variance in run times and envelope sizes, optimal deliberation scheduling corresponds to finding that t_{EA_1} maximizing the sum of $\mathrm{E}[V_{\pi_1}(s_0) - V_{\pi_0}(s_0)|t_{EA_1}]$ and $\mathrm{Cost}(t_{EA_1} + \mathrm{E}[t_{PG_1}|\mathrm{E}[\,|\mathcal{E}_1|\,|t_{EA_1}]])$.

5.2 Recurrent-Deliberation Models

In recurrent deliberation models, the agent has to decide repeatedly how to allocate time to deliberation, taking into account new information obtained during execution. In this section, we consider a particular model for recurrent deliberation in which the agent allocates time to deliberation only at prescribed intervals. We assume that the agent has separate planning and execution modules that run in parallel and communicate by message passing; the planning module sends partial policies to the execution module and the execution module sends observed states to the planning module.

We call the models considered in this section the *discrete, weakly-coupled, recurrent deliberation* models. *Discrete* because each tick of the clock corresponds to exactly one state transition; *recurrent* because the execution module gets a new policy from the planning module periodically; *weakly coupled* in that the two modules communicate by having the execution module send the planning module the current state and the planning module send the execution module the latest policy.

As mentioned earlier, in the recurrent models, it is often necessary to remove states from the envelope in order to lower the computational costs of generating policies from the restricted automata. In general, there are many more possible strategies for deploying envelope alteration and policy generation in recurrent models than in the case of precursor models. To cope with the attendant combinatorics, we raise the level of abstraction slightly and assume that we are given a small set of strategies that have been determined empirically to improve policies significantly in various circumstances. Each strategy corresponds to some fixed schedule for allocating processor time to envelope alteration and policy generation routines. In addition, we assume for simplicity of exposition that the decision problem is a simple goals of achievement, in which goal states have a reward of 0 and all others -1.

Discrete; weakly-coupled; fixed intervals We first consider the case in which communication between the two modules occurs exactly once every n execution steps or *ticks*; at times $n, 2n, 3n, \ldots$, the planning module sends off the policy generated in the last n ticks, receives the current state from the execution module, and begins deliberating on the next policy. Strategies would be tuned to a particular n-tick planning cycle. One strategy might be to use a particular pruning algorithm to remove a specified number of states and then use whatever remains of the n ticks to generate a new policy. In this regime, deliberation scheduling consists of choosing which strategy to use at the beginning of each n-tick interval.

Before we get into the details of the decision model, consider some complications that arise in recurrent deliberation problems. At any given moment, the agent is executing a policy, π, defined on the current envelope and augmented with a set of reflexes for states falling outside the envelope. The agent begins executing π in state s. At the end of the current n-tick interval, the execution module is given a new policy π', and the planning module is given the current state s'. It is possible that s' is not included in the envelope for π'; if the reflexes do not drive the robot inside the envelope then the agent's behavior throughout the next n-tick interval will be determined entirely by the reflexes. Figure 6 shows a possible run depicting intervals in which the system is executing reflexively and intervals in which it is using the current policy; for this example, we assume reflexes that enable an agent to remain in the same state indefinitely.

Let $\delta_n(s, \pi, s')$ be the probability of ending up in s' starting from s and following π

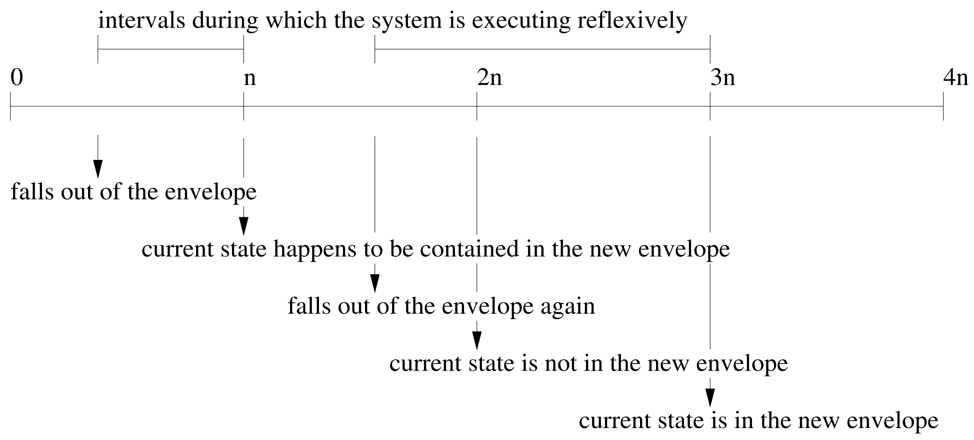

Figure 6: Recurrent deliberation

for n steps. Suppose that we are given a set of deliberation strategies $\{F_1, F_2, \ldots\}$. As is usual in such combinatorial problems with indefinite horizons, we adopt a myopic decision model with a limited horizon. In particular, we assume that, at the beginning of each n-tick interval, we are planning to follow the current policy π for n steps, follow the policy $F(\pi)$ generated by some strategy F attempting to improve on π for the next n steps, and thereafter follow the optimal policy π^*. If we assume that it is impossible to get to a goal state in the next $2n$ steps, the expected value of using strategy F is given by

$$\left[-\sum_{i=0}^{2n-1}\gamma^i + \gamma^{2n}\sum_{s'\in S}\delta_n(s,\pi,s')\left[\sum_{s''\in S}\delta_n(s',F(\pi),s'')V_{\pi^*}(s'')\right]\right] - V_\pi(s),$$

where $0 \leq \gamma < 1$ is a discounting factor, controlling the degree of influence of future results on the current decision.

Extending the above model to account for the possibility of getting to the goal state in the next $2n$ steps is straightforward; computing a good estimate of V_{π^*} is not, however. We might use the value of some policy other than π^*, but then we run the risk of choosing strategies that are optimized to support a particular suboptimal policy when in fact the agent may be able to do much better. In general, it is difficult to estimate the long term prospects for sequential decision problems of indefinite duration. In the next model, we consider an alternative decision model that avoids computing or even estimating the value of the optimal policy, but has related problems in practice.

Discrete; weakly-coupled; variable intervals One practical problem with the fixed-interval model is that it is difficult to design strategies for a fixed n-tick interval. In this case, we allow variable planning intervals and assume that we can predict reasonably accurately the time required for a given deliberation strategy to run. Also, in anticipation of combinatorial issues that arise in our experimental studies, we adopt a simpler myopic decision model. In this case, we assume that the agent will apply exactly one deliberation strategy and commit to the resulting policy thereafter. The expected value of using strategy F on π assuming that F will take k steps is just

$$\left[-\sum_{i=0}^{k-1}\gamma^i + \gamma^k \sum_{s'\in S}\delta_k(s,\pi,s')V_{F(\pi)}(s')\right] - V_\pi(s),$$

where the first term corresponds to the value of using π for the first k steps and $F(\pi)$ thereafter and the second term corresponds to the case in which we do no deliberation whatsoever and use π forever. As in the model described in the previous section, we assume that the goal cannot be reached in the next k steps; again it is simple to extend the analysis to the case in which the goal may be reached in fewer than k steps.

The above decision model does not require that we compute the value of the optimal policy. The model does, however, require that we compute the long-term performance of policies. In practice, of course, we will only compute an estimate, but this estimation will turn out to be rather difficult.

5.3 Off-line Statistical Methods

In order to estimate the quantities needed for on-line deliberation scheduling, we gather data off-line. The data are used to compute statistical estimates of the expected improvement of various deliberation strategies. Although this process can be time consuming, it is a fixed off-line cost that allows us to deliberate effectively when new problems are presented on line.

6 Experimental Results

The algorithms described in Section 4 have been implemented in a planning and execution system called Plexus. This section reports on three experiments conducted with Plexus in the simulated robot-navigation environment described in Section 4.3. The first explores how the value of the policy increases with time in the precursor model. The second investigates the use of statistics-based deliberation scheduling in the recurrent model. In both the first and second experiments we compare the performance with policy iteration. The third experiment compares our planning approach with other methods on a variety of domains, attempting to characterize the domain properties that contribute to the success of the various algorithms.

6.1 Greedy Precursor Deliberation

In general, computing the optimal deliberation schedule for the multiple-round precursor-deliberation models described above is computationally complex. We have experimented with a number of simple, greedy and myopic scheduling strategies; we report on one such strategy here.

We gathered a variety of statistics on how extending the envelope increases value. The statistics that proved most useful corresponded to the expected improvement in value for different numbers of states added to the envelope. Instead of conditioning just on the size of the envelope prior to alteration we found it necessary to condition on both the size of the envelope and the estimated value of the current policy (*i.e.,* the value of the optimal policy computed by policy iteration on the restricted automaton). At run time, we use the size of the automaton and the estimated value of the current policy to index into a table of *performance profiles* giving expected improvement as a function of number of states added to the envelope.

Using the mobile-robot domain (the single fourth floor, 664 states), we generated 1,600,000 data points to compute statistics of the sort described above. We also generated estimates of the time required for one round of envelope alteration followed by policy generation, given the size of the envelope, the number of states added, and value of the current policy. We use the following simple greedy strategy for choosing the number of states to add to the envelope on each round. For each round of envelope alteration followed by policy generation, we use the statistics to determine the number of states which, added to the envelope, maximizes the ratio of performance improvement to the time required for computation.

We compared the performance of (1) the Plexus system using the greedy deliberation

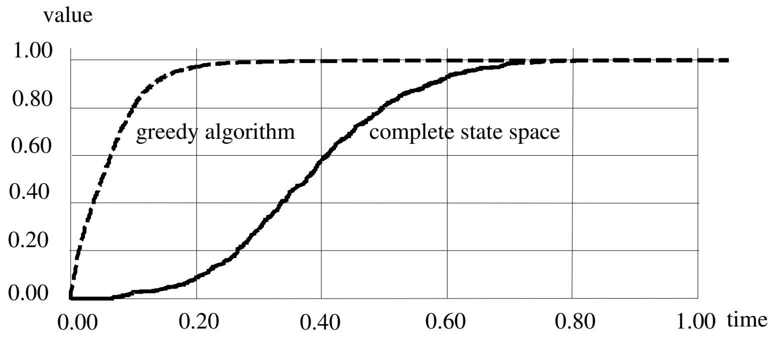

Figure 7: Comparison of Plexus using greedy deliberation strategy (dashed line) with the policy iteration optimization method (solid line): Average over 620 runs

strategy with (2) policy iteration optimizing the policy for the whole domain. Our results show that Plexus 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 7 shows average results from 620 runs, where a single run involves a particular start state and goal state. The graph shows the average estimated value of the start state under the policy available at time t, $\hat{V}_\pi(s_0)$, 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 optimal policy to the value of the current 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[1].

The greedy deliberation strategy performs significantly better than the standard optimization method. We also considered simple strategies such as adding a small fixed number of fringe states each iteration, or adding the whole fringe each iteration. These strategies 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.

[1]Because values are negative, this ratio is close to 0 for the worst policies, and 1 for optimal policies.

6.2 Recurrent Deliberation

In this section, we present results for recurrent-deliberation problems of indefinite duration using statistical estimates of the value of a variety of deliberation strategies. We do this for the discrete, weakly-coupled decision model which allows variable-length intervals for deliberation. Although fixed-length intervals facilitate exposition, it is much easier to collect useful statistical estimates of the utility of deliberation strategies if the deliberation interval is allowed to vary. For the remainder of this section, a deliberation strategy is just a particular sequence of invocations of envelope alteration and policy generation routines.

6.2.1 Gathering Statistics

The utility of a deliberation strategy is characterized as a function of attributes of the policy to which it will be applied, such as the estimated value of the policy and the size of the envelope. For example, the function $EIV(F, \hat{V}_\pi, |\mathcal{E}_\pi|)$ provides an estimate of the expected improvement in the estimated value from using the strategy F assuming that the estimated value of the current policy and the size of the corresponding envelope fall within specified ranges. This function is implemented as a table in which each entry is indexed by a strategy F and a set of ranges over the attributes. We determine the EIV function off-line by gathering statistics for F running on a wide variety of policies. At run time, the deliberation scheduler computes an estimate of the value of the current policy $\hat{V}_\pi$, determines the relevant attributes of current policy, for example the size $|\mathcal{E}_\pi|$ of the corresponding envelope, and chooses the strategy F maximizing EIV for those attributes. Note that actual results given in Section 6.2.2 use EIV contingent on other attributes also.

To build a table of estimates of function EIV off-line, we begin by gathering data on the performance of strategies ranging over possible initial states, goals, and policies. For a particular strategy F, initial state x, and policy π, we run F on π, determine the elapsed number of steps k, and compute the estimated improvement in value as defined in the section describing the discrete, weakly-coupled, variable interval deliberation model. Given data of the sort described above, we build the table for $EIV(F, \hat{V}_\pi, |\mathcal{E}_\pi|)$ by appropriately dividing the data into buckets with equal numbers of elements.

One unresolved problem with this approach is exactly how to compute $\hat{V}_\pi(x)$. Recall that π is only a partial policy defined on a subset of $\mathcal{S}$ augmented with a set of reflexes to handle states outside the current envelope. In estimating the value of a policy, we are really interested in estimating the value of the augmented partial policy. If the reflexes kept the agent in the same place indefinitely, then as long as there was some nonzero probability of falling out of the envelope with a given policy starting in a given state, the actual value of the policy in that state would be $-1/(1-\gamma)$. Of course, this is an extremely pessimistic estimate for the long term value of a particular policy since in the recurrent model the agent will periodically compute a new policy based on where it is in the state space. The problem is that we cannot directly account for these subsequent policies without extending the horizon of the myopic decision model and absorbing the associated computational costs in off-line data gathering and on-line deliberation scheduling.

To avoid complicating the on-line decision making, we have adopted the following expedient, which allows us to keep our one-step-lookahead model. We modify the transition probabilities for the restricted automaton so that there is always a nonzero probability of getting back into the envelope after having fallen out of it. Exactly what this probability should be is difficult to determine. The particular value chosen will determine just how concerned the agent will be with the prospect of falling out of the envelope. In fact, the value is dependent on the actual strategies chosen by deliberation scheduling which, in our particular case, depends on EIV and this value of falling back in. We might possibly resolve the circularity by solving a large set of simultaneous equations; in practice, we have found that it is not difficult to find a value that works reasonably well.

6.2.2 Experimental Results

Domain The experimental results for the recurrent model were obtained on the mobile-robot domain in a range of sizes. Table 1 shows the numbers of locations and states for the different sized domains in the columns `nLocs` and `nStates` respectively. The larger domains are obtained by combining multiples of the floor plan shown in Figure 4 into two-dimensional grids, with one connecting corridor on each side.

The actions available to the agent were the same as those described in Section 4.3 and used to obtain the precursor-model results. The transition probabilities were also the same, except that the domain was modified slightly to no longer contain any complete sinks; this means that even if the agent falls into a *semi-sink* state (i.e. one with low but nonzero probabilities of making transitions into another state), it can eventually reach the goal. In

World	nLocs	nStates	ExpCost	CostOut
1x1	158	632	-500	-1
2x2	632	2528	-1000	-4
3x3	1422	5688	-1500	-9
4x4	2528	10112	-2000	-16
5x5	3950	15800	-2500	-25

Table 1: Information about the domains used to obtain experimental results for the recurrent deliberation model

each case, the discount factor, γ, was 0.9999.

In addition to the numbers of locations and states, Table 1 also shows the values used for `CostOut` and `ExpCost` when generating the statistics; `CostOut` is the estimate of how long the agent must wait once it has fallen out of the envelope using only its reactive policy and `ExpCost` is the estimate of the average value of the states in the world. These values are conservative estimates given the relatively benign nature of the domain — about 3% of the states are semi-sinks.[2]

Deliberation strategies Our implementation provided a number of operations on the envelope, including envelope `optimization` (`O`) and the following types of envelope alteration:

1. `findfirstpath (F)`: find 10 paths from the agent's current state, x_{cur}, to a goal state, using a randomized depth-first search, and chose the shortest path to be the initial envelope
2. `depth-first search (D)`: if the agent's current state x_{cur} is not in the envelope, using a depth-first search as above, find a path from x_{cur} back to the envelope, and add this path to the envelope
3. `strengthen (S[N])`: we used the following heuristic to extend the envelope: find the N most likely fringe states and add them to the envelope

[2] Given $\gamma = 0.9999$, the value of a complete sink is $-1/(1-\gamma) = -10,000$.

World	# Stats Runs	# Data Points
1x1	5858	155696
2x2	6423	168215
3x3	5758	128611
4x4	5363	110830
5x5	4963	94021
total	28365	657373

Table 2: Statistics for different sized domains

4. `prune (P[N])`: of the states that have a worse value than the current state, remove the N least likely to be reached using the current policy.

`findfirstPath` is executed only once, to obtain the initial envelope. The deliberation module then chooses between a set of 24 hand-crafted strategies. Each of these 24 strategies begins with a `depth-first search` and ends with an `optimization` operation. Between these first and last operations, strengthening, pruning and optimization are used in different combinations with different numbers of states to be added or deleted. In order to be able to compare the same strategy for the different sized domains, we formulated the number of states to be added and deleted in terms of the dimensions of the world; for world size n x n, where $n = 1, ..., 5$, the number of states to be added or deleted was one of $\{5n^2, 10n^2, 20n^2\}$. Examples strategies are:

{D S[10n^2] O}
{D P[20n^2] O}
{D P[5n^2] S[10n^2] O}
{D S[20n^2] P[10n^2] O}
{D S[10n^2] O P[10n^2] O}

Statistics We collected statistics over a large number of runs (where a run is an execution of the system with a particular start/goal pair) for each size domain, generating data points for strategy execution as shown in Table 2.

The start/goal pairs were chosen uniformly at random from all states excluding the semi-sinks and we ran the simulated robot in parallel with the planner until the goal was reached. The planner executed `findfirstpath (F)` to obtain the initial envelope, then executed the following loop: choose one of the 24 strategies uniformly at random, execute that strategy, and then pass the new policy to the simulated robot.

Conditioning attributes We found the following conditioning variables to be significant: the envelope size, $|\mathcal{E}|$, the estimated value of the current state $\hat{V}_\pi$, the "fatness" of the envelope (the ratio of envelope size to fringe size), and the Manhattan distance, M, between the start and goal locations. We then built the lookup tables of the expected improvement in value as a function of $|\mathcal{E}|$, $\hat{V}_\pi$, the fatness, M and the strategy s. The lookup table granularity used was 3 buckets per attribute dimension.

Simulation results To compare the planners, we took 50 pairs of start and goal states from each world, chosen uniformly at random from all states excluding the semi-sink states. For each pair we ran the simulated robot in parallel with the following deliberation mechanisms:

- recurrent-deliberation with strategies chosen using statistical estimates of EIV (LOOKUP)
- policy iteration over the entire domain, the agent initially acts according to its reflexes, with a new policy given to the agent
  - after each iteration (ITER)
  - only after the policy has been completely optimized (WHOLE).

We found that the statistics were not very sensitive to the size of the domain; statistics gathered for the smaller-sized worlds transferred fairly well to the larger-sized worlds. The LOOKUP results use the statistics lookup table compiled from the approximately 660,000 data points.

Figure 8 shows the average number of steps taken by the agent to reach the goal for the various algorithms. For the smaller domains, the greedy deliberation algorithm does not perform better than either of the policy iteration algorithms. However, as we move to larger domains, the improvement is marked. As we might expect, WHOLE becomes

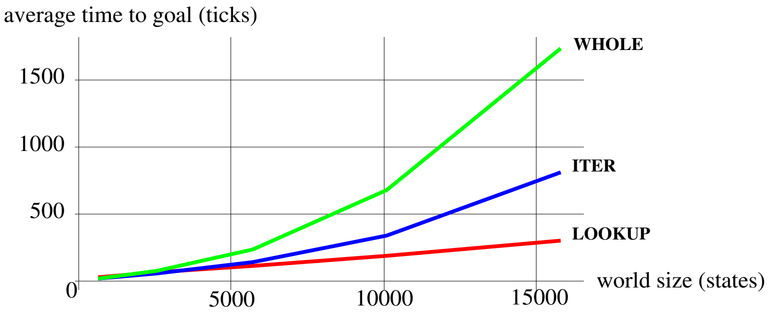

Figure 8: Recurrent model comparison of greedy deliberation and policy iteration on various sizes of domain

computationally infeasible as the size of the domain increases. ITER also shows a nonlinear degradation in the time to goal. LOOKUP shows linear behavior, clearly suffering less than the other planners as the domain size increases.

The implementation used to obtain these experimental results did not include a separate trajectory planning operation that looks for new paths to goal states. This lack of exploration meant that the planner did not look for shortcuts either to states in the envelope but significantly closer to the goal, or to the goal itself. Therefore, the performance depended more on the quality of the first path used as the initial envelope than it might have if trajectory planning had been implemented. Without the exploratory trajectory planning, all the significant path planning is done when the initial envelope is found; in this case, if that initial envelope contains a path of length close to the Manhattan distance, the greedy deliberation strategy performs quite well. However if the first path is not a good one, the strategy ends up exploring most of the state space and loses its performance gains over the policy iteration algorithm. The addition of trajectory planning would probably improve the performance of the system as a whole.

6.3 Comparative Studies

In this section, we describe the results of extensive experimental comparison between the recurrent-deliberation planning algorithm and a variety other algorithms. The comparisons were made on variations of the robot-navigation domain that illustrate different types of problem difficulty.

6.3.1 Algorithms

In our experiments, we compare the performance of the following six algorithms on a variety of domains.

Recurrent-deliberation with fixed strategy We run a version of the recurrent-deliberation algorithm described in Section 4. It has the following fixed deliberation strategy: best-first search, prune, strengthen, optimize (BPSO). The best-first search uses as its measure of goodness of a state s the probability of a path from the current state to s divided by the length of this path. If several paths to s have been found, the one with the highest measure is used. We have found this search technique to be much more effective than the depth-first search described in Section 4.

We used a fixed strategy rather than the greedy deliberation strategy because the latter requires a large number of executions of the system in order to gather the statistics it uses. If we are interested in designing a planner to work very well in one domain, this overhead is reasonable, but the computational cost of gathering these statistics for all 500 domains would have been prohibitive: several CPU years on a Sun Sparcstation 10.

The prune and strengthen algorithms used in this study were slightly different from those presented in Section 4; the parameter for strengthen is the proportion of the fringe to add, rather than the number of states to add, and the parameter for prune is the proportion of the envelope to delete, rather than the number of states to delete.

RTDP The *real-time dynamic programming* (RTDP) algorithm was developed by Barto, Bradtke, and Singh [Barto *et al.*, 1993]. It is a generalization of Korf's Learning-Real-Time-A* algorithm to stochastic problems. In our application of RTDP, the goal state has reward 0 and all other states have reward -1.

The algorithm begins with value estimates $V(s) = 0$ for all $s \in \mathcal{S}$. It then conducts a series of simulated "runs" in the environment, using the greedy policy with respect to the value estimates and updating the value estimates as it goes. More formally, with start state s_0, goal state g, and time-limit l, the algorithm is specified as follows.

```
loop
```

```
s := s_0;  c := 0
while c < l and s ≠ g do begin
    a := argmax_{a∈A} Σ_{s'∈S} Pr(s, a, s')V(s')
    s' := element drawn from S according to distribution Pr(s, a, s')
    V(s) := r(s) + max_{a∈A} Σ_{s'∈S} Pr(s, a, s')V(s')
    c := c + 1;  s:=s'
end while
end loop
```

Barto, Bradtke, and Singh have shown, under some additional conditions, that the RTDP algorithm converges with probability 1 to the optimal value function on the set of relevant states, where a state is relevant if it is reachable from the start state under the optimal policy. In our experiments, the policy is being executed while it is being improved. On every cycle of the main loop, the start state s_0 is set to be the current actual state of the agent and the newly computed policy (greedy with respect to the value function) is given to the agent for execution.

Trajectory Planning (Replan) This algorithm operates under the standard assumptions of AI planning. An initial path to the goal is found, then the agent follows the path. If it falls off the path, the reflexes are executed until a new path to the goal is found.

Trajectory Planning with Repair (Recover) This algorithm is like trajectory planning except that when the agent falls off the nominal path, a path is planned back to the original path, rather than to the goal. This is likely to be somewhat more efficient in planning time, though perhaps slightly slower in number of real-world steps to the goal.

6.3.2 Domains

It is unlikely that any one algorithm will perform best on every possible domain. In order to explore the question of which algorithms are best suited to which kinds of domains, we examined 500 variants of the robot navigation domain presented in Section 4.3. The domains were chosen by varying four parameters called size, volatility, noise and danger.

Size We used four domain sizes: the 664-state world described in Section 4.3 and 2x2, 3x3 and 4x4 copies of this world, with a corridor joining each adjacent pair of copies.

Volatility The volatility of a domain is the average time taken to execute one action; this characterizes the amount of time the planner has to generate new policies as the agent moves through the state space. In the experiments shown below, the volatilities were 1, 10, 30, 100 and 300 actions per second of planner CPU time (on a Sun Sparcstation 10) [3].

Noise We used five levels of noise in the outcome of the robot's actions; the probability of a GO action succeeding was 1, 0.85, 0.7, 0.4 and 0.25 respectively. The turn actions were modeled similarly, with probabilities 1, 0.9, 0.8, 0.6 and 0.4 of success.

Danger We introduced states from which the robot had a low probability of escape; we call such states p-sinks, where p indicates the maximum, over all actions, of the probability of escape from the state. These p-sinks are created by modifying the probability of success of the GO action. The five danger levels correspond to 0, 10, 20, 30 and 40% of the states being 0.95-sinks, and 0, 5, 10 and 15% of the states being 0.999-sinks. If the robot enters a 0.999-sink, it will stay there on average about 700 steps. In dangerous worlds it is clearly an advantage to give such states a wide berth; the more noisy the domain, the more advantageous it is.

6.3.3 Results

We implemented the 500 domains corresponding to all combinations of the parameters size, volatility, noise and danger, and ran 5000 trials of each planner.

Figure 9 shows the performance of the four planners as each of the parameters varies. For example, for Figure 9a we divided the 500 domains into four sets corresponding to the four world sizes. For each planner, we computed the mean number of steps to goal over all trials from each set, and plotted the means against the world sizes.

[3] The volatilities used seem high for this type of domain; this is because the model is highly simplified (*e.g.,* coarse discretization into states), so the computation time allotted to the planner must be correspondingly shortened.

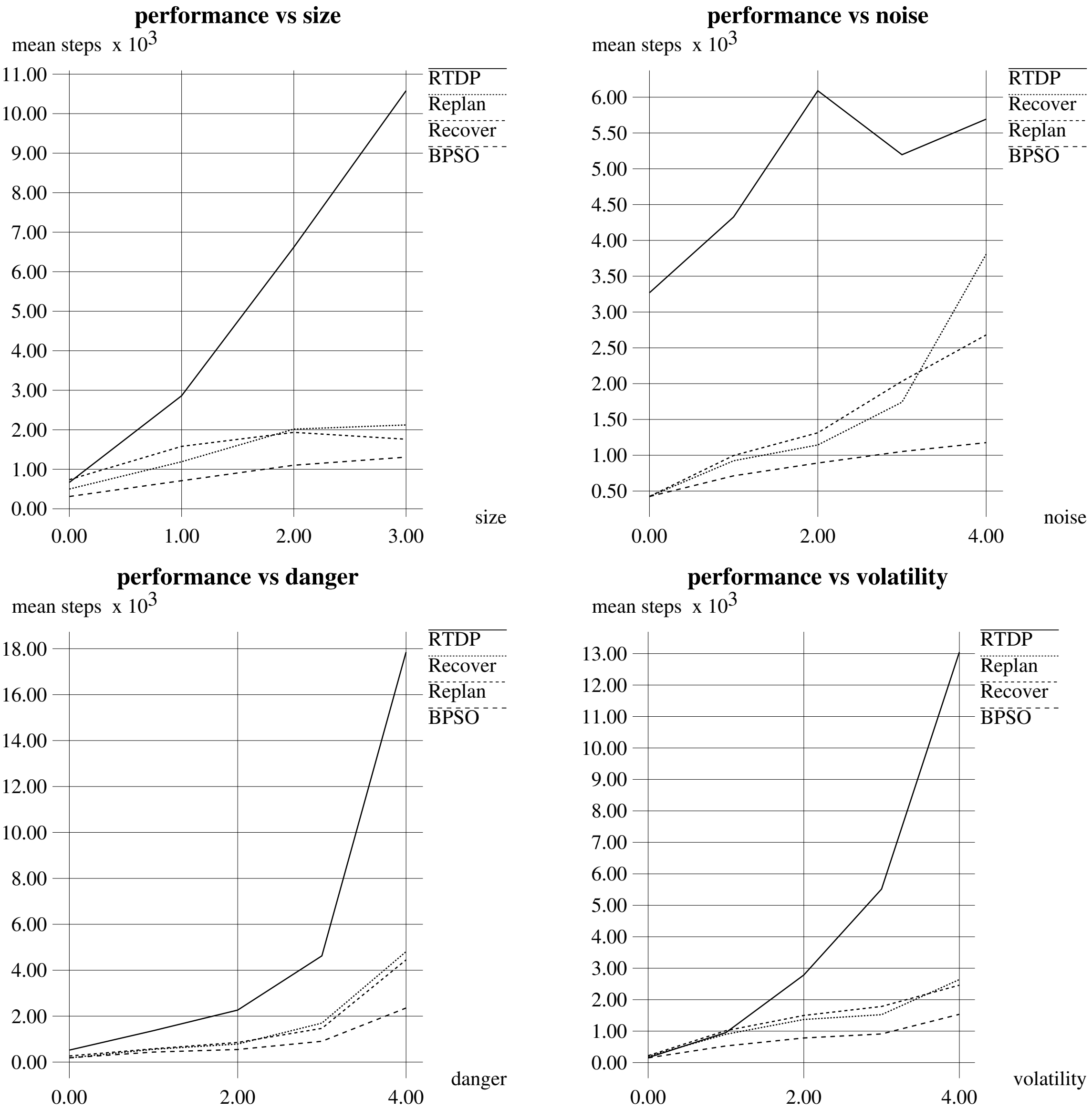

Figure 9: Comparison of recurrent-deliberation with fixed strategy (BPSO) with RTDP and classical planning (Replan and Recover) for domain characteristics: (a) Size (b) Volatility (c) Entropy (d) Regularity. Each graph shows the average performance in terms of steps to goal, versus a single attribute

While RTDP performs substantially better than the policy iteration algorithms (WHOLE and ITER), it appears to perform asymptotically worse than BPSO, Recover and Replan. RTDP suffers greatly from increased danger since it bases its search on a simulation of the stochastic process; with many sinks in the world, RTDP spends much of its time simulating the effects of taking actions from within a sink. We conjecture that the improvement in RTDP's performance when the noise is high is due to the fact that high noise forces RTDP to examine the entire state space early; this is highly advantageous to it when the world is also dangerous, and runs in this type of domain were the principal contributors to this improvement.

Replan and Recover both perform better than we might have expected from such simple strategies. In the robot-navigation domain, in general there is two-way connectivity; the robot can reverse its recent movements. Replan is effective because the best-first search algorithm works well in this class of domains. Both Recover and Replan deteriorate compared to BPSO as the uncertainty in the result of the action increases, and when the domain becomes more irregular and the consequences of a non-desired outcome become more severe.

In most cases, BPSO performs better than all the other planners; as the domains become more difficult, it performs comparatively better. It is more sensitive to danger and volatility than to size and noise.

7 Representing Goals with Reward Functions

In early AI work on planning, it was traditional to have a goal of achievement specified by a logical expression over properties of world states. This translates into having a set of desirable world states and the implicit goal to reach one of these states in the least possible amount of time. At the same time, work in temporal and dynamic logics gave us the notions of a proposition being *always* true or *eventually* true and of one predicate being true *until* another became true. These ideas have been attractive to AI researchers because they give us a more complex, compositional language in which to express goals. Unfortunately, these expressions are not suitable for direct use as goals in AI applications. An agent with the goal of $eventually(p)$ has the option of postponing p indefinitely; there is no requirement to achieve p sooner rather than later. Similarly, to have the goal $always(p)$ is to require that p be maintained true into the infinite future, which is impossible in any sort of real world.

7.1 Basic Goal Types

One way to retain these ideas but make them more useful is to replace *eventually* by *asap*, meaning, intuitively, *as soon as possible* and to replace *always* by *alap*, meaning *as long as possible*. In stochastic domains, we can convert goals of this kind into reward functions and apply the same algorithms for finding good policies with respect to the reward functions.

Given a goal of $asap(p)$, we can generate a policy that takes actions in such a way as to minimize the expected number of steps taken before a state in which p holds is entered. First, we generate the reward function

$$r_p(s) = \begin{cases} 0 & \text{if } p(s) \\ -1 & \text{otherwise} \end{cases}$$

We must also modify the environment so that all states s such that $p(s)$ are absorbing; this ensures that we go to the "nearest" state in which p holds, independent of the states that will follow. This is a common kind of reward function used in reinforcement-learning problems, as well.

Similarly, given a goal of $alap(p)$, we can generate a policy that takes actions in such a way as to maximize the number of steps taken before a state in which $\neg p$ holds is entered. We can use the reward function given above, but this time, we modify the environment so that all states s such that $\neg p(s)$ are absorbing; this ensures that no good results can ensue after encountering a state in which p does not hold.

If we have a longer-term goal of staying in states in which p holds *as much as possible*, that is $amap(p)$, then we need only adopt the reward function above and make no changes to the environment.

7.2 Goal Combination

It is often useful to think of an agent as having multiple goals simultaneously. Goals based on reward functions can be combined to achieve this effect, although the kinds of combination that are appropriate are different than for logical goals.

Goals of achievement can be disjoined in two ways: either $asap(p) \vee asap(q)$ or $asap(p \vee q)$. The first method requires that either p be achieved as soon as possible or that q be achieved as soon as possible, but is indifferent between them. This kind of combination will rarely be useful, because it would allow p to be pursued, even though it takes much longer than achieving q. We therefore prefer the second form, which can be performed by taking the maximum of the reward functions at each state and making any state with zero reward absorbing. Prioritized disjunction (in which, for instance, states in which p holds are to be preferred to states in which q holds, but only if the length of time to achieve p is not too much greater) can be achieved by taking the maximum of scaled versions of the reward functions:

$$r(s) = \max(\alpha r_p(s), \beta r_q(s))$$

where α and β encode the relative desirability of achieving p and achieving q. The same technique can be applied to create reward functions for the goals $alap(p \vee q)$ and $amap(p \vee q)$.

External conjunction of *asap* goals is problematic: what does it mean to achieve p as soon as possible *and* to achieve q as soon as possible? In general, these goals will be conflicting and the conjunction is meaningless. However, we can accommodate internal conjunction by taking the minimum of the reward function at each state. Again, we can apply the same technique to create reward functions for the goals $alap(p \wedge q)$ and $amap(p \wedge q)$.

Another useful goal combination is *asap-maint*(p, q), in which the goal is to achieve p as soon as possible, and to maintain q until p has been achieved. This can be accomplished using the same reward function as before, but making all states in which $\neg q$ holds absorbing as well. Even though there is no difference in instantaneous value between states in which q does and does not hold, the $\neg q$ states are both bad and absorbing, which will give them a very high negative value. This is essentially what was done in the experimental domain described earlier, with the stairwells being absorbing states.

Many languages for the combination of goals allow sequencing, in which it is specified that p is to be achieved, then q is to be achieved. If it is not possible for the agent to perceive or remember that p has been achieved, then sequenced goals cannot be specified using reward functions. If the agent can perceive that p has been achieved (notated $prev(p)$), then the goal $asap(prev(p) \wedge q)$ will have the desired effect.

7.3 Modifying the Planning Algorithm

The planning algorithm described earlier can be applied directly to the whole range of possible reward functions, but some aspects can be tuned to improve the early behavior of the algorithm.

For *asap* goals, it makes sense for the initial envelope to include some path, however tenuous, from the current state to some goal state. If this is not the case, there is no basis for assigning value to the states in the envelope and the policy will essentially be random. For *alap* goals, it is sufficient for the initial envelope to simply be the current state. A good envelope for an *alap* goal need only contain a cycle or set of states that the agent can stay in with high probability.

In addition, the appropriateness of various deliberation strategies will also depend on the type of the goal. This dependence can be handled directly by the statistics-based deliberation-scheduling mechanisms described above.

8 Extensions to this Approach

We are currently exploring a number of extensions to the basic Plexus planning approach. They include relaxing the assumptions of complete observabiltiy, exploring compositional representations of the state space and state transition model, and dealing with domains in which the number of actions is very large.

8.1 Partial Observability

We are exploring an extension of the MDP model called *partially observable Markov decision processes* (POMDPs) [Lovejoy, 1991, Monahan, 1982], which, like the MDP model, was developed within the context of operations research. The POMDP model provides an elegant solution to the problem of acting in partially observable domains, treating in a uniform way actions that affect the environment and actions that only affect the agent's state of information.

When the state is not completely observable, we must add a model of observation. This includes a finite set $\mathcal{O}$ of possible observations and an observation function O, mapping $\mathcal{A} \times \mathcal{S}$ into discrete probability distributions over $\mathcal{O}$. One might simply take the set of observations to be the set of states and treat a POMDP as if it were an MDP. The problem is that the process would not necessarily be Markov: there could be multiple states in the environment that require different actions but appear identical. The result is that even an

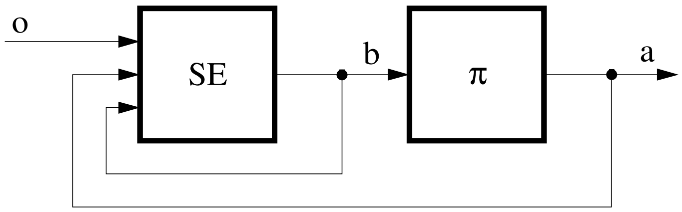

Figure 10: Controller for a POMDP

optimal policy of this form can have arbitrarily poor performance. Further, determining whether a policy with a given level of performance exists for this type of non-Markov decision problem is NP-hard [Littman, 1994].

Instead, we introduce a kind of internal state for the agent. A *belief state* is a discrete probability distribution over the set of world states, representing for each state the agent's belief that it is currently occupying that state. Now, we can decompose the problem of acting in a partially observable environment as shown in Figure 10. The component labeled "SE" is the *state estimator*. It takes as input the last belief state, the most recent action and the most recent observation, and returns an updated belief state. The second component is the *policy*, which now maps belief states into actions.

The key to finding optimal policies in the partially observable case is that the problem can be cast as a *completely observable* continuous-space MDP in which the states are the belief states. The belief MDP is Markov [Smallwood and Sondik, 1973] and having information about previous belief states cannot improve the choice of action. Most importantly, if an agent adopts the optimal policy for the belief MDP, the resulting behavior will be optimal for the partially observable process. The remaining difficulty is that the belief process is continuous and therefore not susceptible to standard methods for finding policies. Cassandra, Kaelbling, and Littman [Cassandra *et al.*, 1994] review existing algorithms from the operations research literature and sketch a new one that is more computationally tractable. In addition, they show that in many cases approximately-optimal finite-state control automata can be extracted from the resulting policies.

We are investigating ways of integrating these results into Plexus; the simplest approach would be to use the techniques of Plexus to find restricted automata, then simply apply POMDP methods to develop policies. It is likely that a closer and more efficient coupling is possible.

8.2 Abstraction

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.

Markov chains and the Bayesian network formalism [Pearl, 1988] have been shown to be equivalent [Dean and Kanazawa, 1989, Nunez, 1989]. The following simplified version of the Bayesian network formalism is well suited to specifying stochastic state transition and reward models for our purposes. 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([Shachter, 1986]). 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 and cache the values of the state transition and reward functions when they are required by the planning algorithm.

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. Nicholson and Kaelbling [Nicholson and Kaelbling, 1994] investigate the construction of different world views by specifying only a subset of the possible variables in the complete world model. In some cases, these abstract views 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.

Plexus can be extended to work with models at multiple levels of abstraction. It 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 variable nodes. The initial world view is constructed by including only those state variables 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 variable with the next most sensitivity.

8.3 Large Action Spaces

In most scheduling problems and many real-world planning problems, the space of possible actions is quite large. For example, there is ongoing work at NASA in collaboration with the FAA on building systems for the automated management and control of U.S. airports [Davis *et al.*, 1991]. Such systems face daunting combinatorial problems and the need to respond to unforeseen situations (*e.g.,* snowstorms and fog) in a timely manner. One component of such systems involves the assignment of metering gates for unloading and loading planes. In the case of gate assignment, the action space corresponds to the space of possible ways of assigning planes to metering gates.

Large action spaces make it impractical to apply standard policy improvement algorithms that require quantifying over the entire action space. A variant of the method described in this paper can be applied to scheduling problems such as metering gate assignment [Greenwald and Dean, 1994b]. This variant method involves approximation algorithms for both value determination (estimating the expected value of a given conditional schedule) and policy improvement (refining a given conditional schedule to improve its expected value). Our method uses Monte Carlo simulation to identify a subset of reachable states and bottleneck-centered heuristics from operations research [Adams *et al.*, 1988] to guide in incrementally refining conditional schedules. In this work, we have built on the work of Drummond *et al.* [Drummond *et al.*, 1994] who apply a variant of anytime synthetic projection to the problem of scheduling experiments on automatic telescopes and the work of Muscettola and Smith [Muscettola and Smith, 1987] who have demonstrated how to apply bottleneck-centered heuristics in handling uncertainty in arrival and departure times.

In our case, we employ an explicit model representing the dynamics governing the arrival and departure of planes. The combinatorics in such problems is quite daunting and some of our recent work involves anticipating computational demands off-line for processes that exhibit some degree of regularity in order to support online deliberation scheduling [Greenwald and Dean, 1994a, Greenwald and Dean, 1994c]. For example, in the metering gate assignment problem, evening rush hour arrival times for commuter aircraft are notoriously unreliable and so it often does not pay to expend considerable computational resources refining a detailed schedule for such periods.

9 Related Work

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 work of Drummond and Bresina [Drummond and Bresina, 1990]. In the following three subsections, we describe the connection to the work of Drummond and Bresina, the relationship to work in the area of reinforcement learning and adaptive control, and discuss other related work in time-critical decision making.

9.1 Anytime Synthetic Projection

Drummond and Bresina's *anytime synthetic projection* algorithm [Drummond and Bresina, 1990] incrementally constructs conditional plans for stochastic domains. Their work provided the initial motivation for our research. Drummond and Bresina's projection algorithm starts by constructing an initial plan that may be unlikely to be executed without error and then improves the plan as time permits by adding rules to handle situations that arise when actions fail to have their expected results.

Instead of building on Markov decision theory, Drummond and Bresina's work involves search in the space of situations given a set of operators that map situations to situations. The search space is a graph in which each node corresponds to a possible situation and each arc to an applicable operator. The plan that results from synthetic projection is represented as a set of *situated control rules* [Drummond, 1989] which constitute a partial policy by mapping situations to operators corresponding to actions. The resulting conditional plan is similar to the *triangle tables* of Fikes *et al.* [Fikes *et al.*, 1972].

Simplifying somewhat, the synthetic projection involves two basic subroutines. The first is called *traverse* and it searches for a sequence of operators and their most likely resulting situations that satisfies the goal starting from some initial situation.[4] The sequence of operators and situations is used to construct a set of situated control rules that map situations to operators. The second subroutine is called *robustify* and it identifies possible deviations from the most likely resulting situations and then calls traverse to find an alternative sequence of operators to recover from the deviation. Once an initial sequence of operators and situations is found, the algorithm repeatedly calls robustify as time allows.

In terms of incremental refinement of policies and selective exploration of a stochastic domain, our approach is very similar to the work of Drummond and Bresina. Apart from the decision-theoretic methods for guiding deliberation scheduling, the main difference concerns the performance functions on which each approach is based.

In the case of goals of achievement, our method converges to the policy minimizing expected time to achieve the goal in the limiting case that the envelope grows to include the entire state space. Synthetic projection makes no effort to construct policies maximizing expected cumulative reward. Instead synthetic projection seeks to maximize the probability of goal achievement rather than maximize the expected time to goal achievement. In particular, Drummond and Bresina are not able to use expectations to predict and then avoid whole portions of the state space that are dangerous in the sense of having very low expected value.

By adopting methods from Markov decision theory, we are able to identify relevant portions of the state space and then compute the optimal policy for the resulting restricted state space. We improve on the work of Drummond and Bresina by providing (i) coherent semantics for goals in stochastic domains, (ii) theoretically sound probabilistic foundations, (iii) and decision-theoretic methods for controlling inference.

9.2 Dynamic Programming and Reinforcement Learning

The Markov decision process model has been extensively investigated in the dynamic programming and reinforcement-learning communities. There has been a recent surge of inter-

[4] In addition to goals of achievement, Drummond and Bresina can handle temporally extended goals of prevention and achievement, and so the search algorithm has to account for intermediate situations resulting from execution as well as the final situation.

est in methods for mapping models into policies in real time. In addition to Barto, Bradtke, and Singh's real-time dynamic programming (RTDP) algorithm [Barto *et al.*, 1993], which is described in Section 6.3.1, there are a number of other relevant algorithms.

Learning real-time A* [Korf, 1988] is the deterministic special case of real-time dynamic programming. Value iteration can be made much more efficient by performing updates in an intelligent order; this idea is pursued in Moore and Atkeson's prioritized sweeping algorithm [Moore and Atkeson, 1993] and in Peng and Williams' Dyna-Queue algorithm [Peng and Williams, 1993]. These methods are not as directed as RTDP, however, because they do not take into account information about what the starting state is. Finally, Sutton's DYNA system [Sutton, 1990] and work by Whitehead and Ballard [Whitehead and Ballard, 1989] studied learning by interleaving actions in the real world with "simulated" actions that peform value updates on the policy.

9.3 Time-Critical Planning Methods

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. For an overview of resource-bounded decision making methods, see Chapter 8 of the text by Dean and Wellman [Dean and Wellman, 1991]. Boddy [Boddy, 1991] describes solutions to related problems involving dynamic programming. Hansson and Mayer's Bayesian Problem Solver (BPS) [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] and Kushmerick, Hanks, and Weld [Kushmerick *et al.*, 1993] also address the problem of planning in stochastic domains.

Kabanza [Kabanza, 1990] describes a method for planning in nondeterministic environments that relies on exploring a small portion of the set of all possible action sequences and representing the resulting restricted automaton using a propositional branching time logic. Kabanza's approach does not make use of any probabilistic information regarding state transitions and makes no attempt to construct even an approximately optimal plan for any measure of performance. Thiebaux *et al.* [Thiebaux *et al.*, 1994] attempt to extend our work and that of Drummond and Bresina to use probabilistic logic [Nilsson, 1986] for planning under uncertainty. Probabilistic logic provides a more expressive representation than that offered by the Markov chain theory which we employ, but with the increased expressiveness comes increased computational overhead.

10 Conclusions

We have described a method, based on the theory of Markov decision processes, for efficient planning under time constraints in stochastic domains. Unlike classical planning approaches, we consider the nondeterminism in the outcome of actions from the start. Existing methods for finding optimal policies in stochastic domains become intractable in the larger state spaces of real-world problems. We overcome this problem of intractability by using information about the world to restrict the planner's attention to states that are likely to be encountered in satisfying the goal. The planner generates more or less complete plans depending on the time available.

We have described the meta-level control problem of deliberation scheduling, together with a number of deliberation models. Our experimental results for a robot-navigation domain showed that our approach performs much better than policy iteration, both when all the decision making is done prior to execution and when planning and execution are performed in parallel. We have seen that the performance of our approach is influenced by certain characteristics of the domain, and have shown that our approach compares favourably to classical planning and real-time dynamic programming algorithms in domains ranging over values of these characteristics. A more detailed investigation and empirical analysis of the domain characteristics for robot-navigation and other domains including air-traffic scheduling and traffic-light control is currently being undertaken [Kirman, Forthcoming].

We have outlined a number of extensions to the basic planning approach which we are currently exploring. These include relaxing the assumptions of complete observability, using compositional representations and extending the approach to work with multiple levels of abstraction and larger action spaces.

Acknowledgements

Thomas Dean's work was supported in part by a National Science Foundation Presidential Young Investigator Award IRI-8957601, in part by the Air Force's Rome Laboratory and the Advanced Research Projects Agency of the Department of Defense, monitored by the Air Force under Contract No. F30602-91-C-0041, and in part 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

[Adams *et al.*, 1988] Adams, J.; Balas, E.; and Zawack, D. 1988. The shifting bottleneck procedure for job shop scheduling. *Management Science* 34(3):391–401.

[Barto *et al.*, 1993] Barto, Andrew G.; Bradtke, Steven J.; and Singh, Satinder P. 1993. Learning to act using real-time dynamic programming. Technical Report 93-02, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts. To appear in *Artificial Intelligence.*

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

[Bertsekas, 1987] Bertsekas, Dimitri P. 1987. *Dynamic Programming.* Prentice-Hall, Englewood Cliffs, N.J.

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

[Cassandra *et al.*, 1994] Cassandra, Anthony R.; Kaelbling, Leslie Pack; and Littman, Michael L. 1994. Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, WA.

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

[Davis *et al.*, 1991] Davis, Thomas Jr.; Erzberger, H.; Green, S. M.; and Nedell, W. 1991. Design and evaluation of an air traffic control final approach spacing tool. *AIAA Journal of Guidance, Control, and Dynamics* 14:848–854.

[Dean and Boddy, 1988] Dean, Thomas and Boddy, Mark 1988. An analysis of time-dependent planning. In *Proceedings AAAI-88*. AAAI. 49–54.

[Dean and Kanazawa, 1989] Dean, Thomas and Kanazawa, Keiji 1989. A model for reasoning about persistence and causation. *Computational Intelligence* 5(3):142–150.

[Dean and Wellman, 1991] Dean, Thomas and Wellman, Michael 1991. *Planning and Control*. Morgan Kaufmann, San Mateo, California.

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

[Drummond *et al.*, 1994] Drummond, M.; Swanson, K.; and Bresina, J. 1994. Robust scheduling and execution for automatic telescopes. In Zweben, M. and Fox, M., editors 1994, *Knowledge-Based Scheduling*. Morgan-Kaufmann, Los Altos, California.

[Drummond, 1989] Drummond, Mark 1989. Situated control rules. In Brachman, Ronald J.; Levesque, Hector J.; and Reiter, Raymond, editors 1989, *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*. Morgan-Kaufmann, Los Altos, California.

[Fikes and Nilsson, 1971] Fikes, Richard E. and Nilsson, Nils J. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208. Reprinted in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate, eds., Morgan Kaufmann, 1990.

[Fikes *et al.*, 1972] Fikes, Richard E.; Hart, Peter E.; and Nilsson, Nils J. 1972. Learning and executing generalized robot plans. *Artificial Intelligence* 3:251–288.

[Greenwald and Dean, 1994a] Greenwald, Lloyd and Dean, Thomas 1994a. Anticipating computational demands when solving time-critical decision-making problems. In *Workshop on the Algorithmic Foundations of Robotics*.

[Greenwald and Dean, 1994b] Greenwald, Lloyd and Dean, Thomas 1994b. Monte carlo simulation and bottleneck-centered heuristics for time-critical scheduling in stochastic domains. In *ARPI Planning Initiative Workshop*.

[Greenwald and Dean, 1994c] Greenwald, Lloyd and Dean, Thomas 1994c. Solving time-critical decision-making problems with predictable computational demands. In *Second International Conference on AI Planning Systems.*

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

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

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

[Kabanza, 1990] Kabanza, F. 1990. Synthesis of reactive plans for multi-path environments. In *Proceedings AAAI-90.* AAAI. 164–169.

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

[Kirman, Forthcoming] Kirman, Jak oming. *What Makes Planning in Stochastic Domains Hard?* Ph.D. Dissertation, Brown University.

[Korf, 1988] Korf, Richard 1988. Real-time heuristic search: New results. In *Proceedings AAAI-88.* AAAI. 139–144.

[Kushmerick *et al.*, 1993] Kushmerick, Nicholas; Hanks, Steve; and Weld, Daniel 1993. An algorithm for probabilistic planning. Unpublished Manuscript.

[Littman, 1994] Littman, Michael L. 1994. Memoryless policies: Theoretical limitations and practical results. In *From Animals to Animats 3*, Brighton, UK.

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

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

[Moore and Atkeson, 1993] Moore, Andrew W. and Atkeson, Christopher G. 1993. Memory-based reinforcement learning: Efficient computation with prioritized sweeping. In *Advances in Neural Information Processing 5*, San Mateo, California. Morgan Kaufmann.

[Muscettola and Smith, 1987] Muscettola, Nicola and Smith, Steve 1987. A probabilistic framework for resource-constrained multiagent planning. In *Proceedings IJCAI 10*. IJCAII.

[Nicholson and Kaelbling, 1994] Nicholson, Ann and Kaelbling, Leslie Pack 1994. Toward approximate planning in very large stochastic domains. In *Proceedings of the AAAI Spring Symposium on Decision Theoretic Planning*, Stanford, California.

[Nilsson, 1986] Nilsson, Nils 1986. Probabilistic logic. *Artificial Intelligence* 28:71–88.

[Nunez, 1989] Nunez, Linda Mensinger 1989. On the relationship between temporal bayes networks and Markov chains. Masters Thesis, Brown University.

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

[Peng and Williams, 1993] Peng, Jing and Williams, Ronald J. 1993. Efficient learning and planning within the dyna framework. *Adaptive Behavior* 1(4):437–454.

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

[Shachter, 1986] Shachter, Ross D. 1986. Evaluating influence diagrams. *Operations Research* 34(6):871–882.

[Smallwood and Sondik, 1973] Smallwood, Richard D. and Sondik, Edward J. 1973. The optimal control of partially observable markov processes over a finite horizon. *Operations Research* 21:1071–1088.

[Sutton, 1990] Sutton, Richard S. 1990. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, Austin, Texas. Morgan Kaufmann.

[Thiebaux *et al.*, 1994] Thiebaux, Sylvie; Hertzberg, Joachim; Shoaf, William; and Schneider, Moti 1994. A stochastic model of actions and plans for for anytime planning under uncertainty. In Sandewall, E. and Backstrom, C., editors 1994, *Current Trends in AI Planning*. IOS Press, Amesterdam.

[Whitehead and Ballard, 1989] Whitehead, Steven D. and Ballard, Dana H. 1989. A role for anticipation in reactive systems that learn. In *Proceedings of the Sixth International Workshop on Machine Learning*, Ithaca, New York. Morgan Kaufmann. 354–357.

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