Randomization for Robot Tasks:
Using Dynamic Programming in the Space of Knowledge States

Michael Erdmann

Abstract. This paper explores the use of randomization as a primitive action in the solution of robot tasks. An example of randomization is the strategy of shaking a bin containing a part in order to orient the part in a desired stable state with some high probability. Further examples include tapping, vibrating, twirling, and random search. For instance, it is sometimes beneficial for a system to execute random motions purposefully when the precise motions required to perform an operation are unknown, as when they lie below the available sensor resolution.

The purpose of this paper is to provide a theoretical framework for the planning and execution of randomized strategies for robot tasks. This framework is based on the backward-chaining approach of dynamic programming. Specifically, a randomized planner backtracks from the goal in a state space whose states describe the knowledge available to the system at run-time. By choosing random actions in a principled manner at run-time, a system can sometimes obtain a probabilistic strategy for accomplishing a task even when no guaranteed strategy exists for accomplishing that task. In other cases, the system may be able to obtain a speedup over an existing guaranteed strategy.

The main result of this paper consists of two examples. One example shows that randomization can sometimes speed up task completion from exponential time to polynomial time. The other example shows that such a speedup is not always possible.

Key Words. Uncertainty, Robotics, Randomization, Probabilistic algorithms, Automatic programming, Planning with uncertainty

1. Introduction. Over the past several years a planning methodology [31] has evolved for synthesizing strategies that are guaranteed to solve robot tasks in the presence of uncertainty. A guaranteed strategy is a set of actions and sensory operations that accomplishes a task in a bounded predetermined amount of time. Traditionally, such guaranteed strategies make judicious use of sensing and task mechanics, in conjunction with the maintenance of past sensory information and the prediction of future behavior, in order to overcome uncertainty. There are two restrictions on the generality of this approach. First, not all tasks admit guaranteed solutions. Uncertainty simply may be too great to guarantee task success in a specific number of steps. Second, a strategy is only as good as the validity of its assumptions. In an uncertain world all assumptions are subject to uncertainty. For instance, there may be unmodeled parameters that govern the behavior of a system. This fundamental uncertainty limits the guarantees that we can expect from any strategy.

The randomization approach proposed in this paper attempts to bridge these difficulties. First, the underlying philosophy of a randomized strategy assumes that several attempts may need to be made at solving a task. A task is only assumed to be solvable with some probability on any given attempt. This view of a solution to a task broadens the class of solvable tasks. Second, by actively randomizing its actions a system can blur the significance of unmodeled or uncertain parameters. Effectively, the system is perturbing its task solutions slightly through randomization. The intent is to obtain probabilistically a solution that is applicable for particular instantiations of these unknown parameters.

The primary purpose of this paper is to provide a formal framework for randomization. The paper develops randomized strategies on discrete spaces. The main results of the paper are contained in Section 9. That section considers the convergence properties of some simple examples that use randomized strategies. We exhibit one example in which randomization changes the convergence time from exponential to polynomial time. Less one think that randomization can always speed up convergence times, we exhibit another example in which goal convergence requires exponential time both in the randomized and in the non-randomized case.

2. Motivation and Issues

2.1. Accepting Uncertainty. The assumption that the world is perfect, that it may be modeled accurately and controlled precisely, is much too strong an assumption to be realistic. Instead much effort has been devoted over the last few decades to accounting for uncertainty explicitly. The aim has been to develop methods for reducing uncertainty or entropy by judicious use of sensing and action. The difficulty with these approaches is that they continue to make strong assumptions about the world. For instance, generally those frameworks that produce guaranteed plans have trouble dealing with tiny variations in geometry. A strategy that slides one object on top of another may fail if the component surfaces contain small nicks and protrusions. Similarly, if a sensing error is larger than expected, or if a sensor contains an unknown bias, a strategy that relies crucially on the validity of its assumptions will fail. This defeats the philosophy motivating the construction of planners that explicitly account for uncertainty. That philosophy states that one should from the outset be aware of uncertainty, rather than ignore it in the hope that the plans developed for a perfect world will be good enough in the face of uncertainty. The philosophy is defeated because the strategies developed in the quest for guaranteed plans are only as good as the assumptions preceding them.
A key approach in dealing with uncertainty is to tolerate failure. This is a fairly recent idea within the formal planning methods of robotics [15]. No task possesses an absolutely guaranteed solution. Instead of searching for guaranteed solutions, we should try to answer the following three questions, for any task of interest.

- What is the information needed to solve a task?
- What tasks can be solved by a given repertoire of operations?
- How sensitive are solutions of tasks to particular assumptions about the world?

2.2. Randomization. Randomization enters into the investigation of the previously cited questions at the simplest level. Randomization is omnipresent. For instance, uncertainty that is due to noise, either in sensing or control, may be thought of as randomization on the part of nature. The basic issue that this paper and a companion paper [22] address is how active randomization on a robot's part can aid in the solution of tasks.

Let us consider the difference between active randomization and probabilistic or nondeterministic actions. A strategy or action is said to be nondeterministic if its outcome is modeled as a set of possible configurations. The pure nondeterministic model is intended as a worst-case model. A strategy or action is probabilistic if it is nondeterministic and if the set of possible outcomes is probabilistically distributed.

While an action may be probabilistic or nondeterministic, the decision to execute that action is often deterministic. In other words, given certain sensor values, the system selects a certain action in a completely deterministic fashion. It is simply the outcome of the action that is probabilistic or nondeterministic. An alternative approach is for a system to actively make random choices in selecting actions. This process is what we call randomization.

This paper focuses on a basic theoretical framework and presents some simple combinatorial examples. The companion paper [22] discusses convergence times in detail. The results of that paper show that a combination of sensing and randomization can be a useful technique for solving manipulation tasks. The paper considers a peg-in-hole problem, and shows that randomization can extend the class of such tasks that are solvable. In particular, randomization can solve problems that are not guaranteed to be solvable by bounded-step strategies that seek guarantees on each step of iteration. Furthermore, for Gaussian errors in sensing and control, the convergence times of the randomized strategy can be estimated and shown to be at worst linear in the distance from the goal.

2.3. Why Randomization? The need for randomization arises in the context of nondeterministic actions. Randomization is a means of decoupling the world’s behavior from the robot’s strategy, thereby reducing the worst-case effects of this potentially adversarial interaction. When actions are probabilistic, we can, at least in principle, compare different decisions based on their probability of success, then select that decision which maximizes the probability of success. No randomization is required. However, in the setting of nondeterministic actions, we must be prepared to handle worst-case scenarios. This means that we should view uncertainty as an adversary who is trying to foil the system's strategy for attaining the goal, and who will therefore always choose that outcome of a nondeterministic action that prevents the system from attaining its goal.

Imagine a discrete three-state system, as shown in Figure 1. There is one goal state, and two other states, labeled and . There are two nondeterministic actions, , and . If the system is in state then action is guaranteed to move the system to the goal. However, action will nondeterministically move the system from either back to or to the other state . Similarly, if the system is in state , then action is guaranteed to attain the goal, while action will nondeterministically either remain in or move to state . Suppose that the only sensing available is goal recognition. In other words, the system can detect goal attainment, but cannot decide whether it is in state or in state . We observe that there is no guaranteed strategy for attaining the goal. For any fixed deterministic sequence of actions there is some interpretation of the diagram for which the sequence fails to achieve the goal. Said differently, from a worst-case point of view, no finite or infinite deterministic strategy is guaranteed to attain the goal, given that the initial state of the system is unknown.

In contrast, we see that there exists a randomized strategy whose expected convergence time is very low, namely, two steps. This strategy chooses randomly between actions and on each step, choosing each action with probability . Since the system is in some state , the strategy will choose the correct action for that state with probability . This is true independent of the behavior of the system. Thus, by a waiting time argument, the expected time until the system guesses the correct action is two. We should note that there is a tacit assumption here that nature, that is, the adversary, cannot control or pre-observe the dice used to make the randomizing decisions.

The argument of the above example is essentially a worst-case versus expected-case analysis. It may seem strange to compare worst and expected cases. However, there are two important observations to take from this example. First, there is a
major advantage to be gained by considering the expected case rather than the worst case. This is because the task of attaining the goal is solvable only in the expected case, not in the worst case. Second, the expected case convergence time is computed over randomizing decisions actually made by the run-time system, not over externally defined probability distributions. Thus the system has some control over the expected convergence time; the upper bound on this expectation applies for all interpretations of the underlying nondeterministic model.

3. Previous Work. Work on planning in the presence of uncertainty goes back in time as far as can be imagined. Credit for the modern approach probably goes to Bellman [3], who formulated the dynamic programming approach that underlies much of optimal control and decision theory. His ideas were, themselves, based to some extent on the calculus of variations and game theory. See [4] for an introduction to dynamic programming in the discrete domain, and see [49] for an overview of techniques in optimal control.

Within the domain of robotics, uncertainty has always been a central problem. Much of the work on compliant motion planning and geometric modeling in the last decade was motivated by a desire to compensate for uncertainty in control and inaccuracies in the modeling of parts. The aim was to take advantage of surface constraints to guide assembly operations. Inoue [28] used force feedback to perform peg-in-hole assembly operations at tolerances below the inherent positional accuracy of his manipulator. Sinumovic [48] considered both Kalman filtering techniques in position-sensing, and the use of force information to guide assembly operations in the presence of uncertainty. In conjunction with this work there grew an interest in friction and the modeling of contact to describe the possible conditions under which an assembly could be accomplished successfully. See [40], [17], [41], and [55]. More recent work, with an emphasis on understanding three-dimensional peg-in-hole assemblies in the presence of friction and uncertainty, includes [10] and [50].

The formalization and understanding of compliant motion techniques received several major boosts. Whitney [54] introduced the notion of a generalized damper as a way of simplifying the apparent behavior of a system at the task level. Salisbury's [47] work on generalized springs provided a means of stiffness control for six degrees of freedom. Several researchers considered a form of control known as hybrid control (see [34] for an overview of this work). The work of Mason [33] contributed to the understanding of compliant motions by modeling and analyzing compliance in configuration space. In particular, he introduced and formalized the ideas of hybrid control, showing how these could be modeled naturally on surfaces in configuration space. The basic approach is to maintain contact with an irregular and possibly unknown surface, by establishing a force of contact normal to the surface, while position-controlling directions tangential to the surface of contact. In short, uncertainty is overcome in some dimensions. Rairbert and Craig [46] describe a combination of position and force control in their implementation of a hybrid control system. See also [28] and [44] for earlier work on hybrid control.

Randomization for Robot Tasks

The generalized spring and generalized damper approaches provided a new set of primitives with which one could reduce uncertainty in specific local settings. In parallel with this work, there arose a desire to synthesize entire planning systems that could account for uncertainty. Early work considered parameterizing strategies in terms of quantities that could vary with particular problem instantiations. The skeleton strategies of Lozano-Pérez [30] and Taylor [51] offered a means of relating error estimates to strategy specifications in detail. Brooks [6] extended this approach using a symbolic algebra system. His system could be used both to provide error estimates for given operations, as well as to constrain task variables or add sensing operations in order to guarantee task success. Along a slightly different line, Dufay and Latombe [18] developed a system that observed execution traces of proposed plans, then modified these using inductive learning to account for uncertainty.

In 1983, Lozano-Pérez et al. [31] proposed a planning framework for synthesizing fine-motion strategies in the presence of uncertainty. This framework has strong connections to the dynamic programming approach mentioned above. In particular, the framework generates plans by recursively backchaining from the goal. The preimage framework directly incorporates the effect of uncertainty into the planning process. See also [35], [19], [9], [15], and [12] for further work on preimages, and see the recent book by Latombe [29] for an excellent discussion of preimage work.

An important offspring of the LMT preimage planning methodology was Donald's Ph.D. thesis [14], [15], [16], which moved away from the requirement that a strategy, in order to be considered a legitimate strategy, actually be guaranteed to solve a task in a fixed predetermined number of steps. This is an important and subtle point, that forms a motivation for this paper. By permitting strategies to fail, one can vastly increase the class of tasks that one would consider solvable. Indeed, it is clear that in some completely imperfect world, no task is ever guaranteed to be solvable assuming worst-case adversaries. The real world is such a world. Yet many tasks are solvable simply because they may be completed successfully sometimes.

Randomization has been used explicitly for solving some automation tasks in the past. In the domain of mobile robots, see, for instance, Arkin [1], who injects noise into potential fields to avoid plateaus and ridges. Barraquand and Latombe [2] have also investigated a Monte Carlo approach for escaping from local minima in potential fields. Some probabilistic work has aimed at facilitating the design process. For instance, Boothroyd et al. [5] considered the problem of determining the natural resting distributions of parts in a vibratory bowl feeder. Goldberg has investigated probabilistic strategies for grasping objects [25], [26], [27]. That work is also interested in the development of a general approach towards the analysis and synthesis of randomized strategies for manipulation tasks.

The main convergence results of this paper are in the domain of near-sensorless tasks. These are tasks in which the only sensing available is a sensor that signals goal attainment. Such tasks are closely related to fully sensorless tasks. Sensorless tasks form an important subclass of the set of robot tasks. Mason [36], [37] has studied these problems extensively. The motivation for studying sensorless
problems stems from the realization that almost all tasks involve some operations in which the mechanics of object interactions dominate any informational content provided by the sensors. For instance, in grasping or pushing objects, even if sensors are available to provide a general sense of the object's behavior, the behavior of the object at the instant of contact tends to lie below the resolution of the sensors. Thus it is important to understand the behavior of objects, and the manner by which one can control them in the absence of sensory information. Brost [8] and Peshkin [45] have further explored sensorless grasping and pushing. Mani and Wilson [32], Erdmann and Mason [23], and Natarajan [38] have looked at other tasks that are amenable to sensorless solutions, such as the problem of unambiguously orienting an object given complete uncertainty as to the object's initial configuration, and Wang [53] has studied the impact problem. Finally, see [52] for research that compares sensor-based and sensorless strategies in the domain of parts orienting.

4. Organization of the Rest of the Paper. The emphasis on this paper is on the definition, construction, and analysis of strategies that involve both deterministic and randomized decisions. At execution time, the system has access to a history of past and current sensory observations and previously executed actions. The planning process must take account of all the possible histories that the system might see at execution time. The basic approach taken in this paper is to create plans by backchaining in the space of knowledge states. A knowledge state captures all the information available to the system given its sensors, actions, predictions, and retention of history.

We will develop randomized strategies in the context of discrete tasks. Sections 5 and 6 set the stage for this development. Section 5 defines discrete spaces, sensors, actions, and tasks. Section 6 first reviews dynamic programming in the context of perfect sensing. Next, this section defines knowledge states, and shows how a system can update its knowledge state using prediction and sensing. The section then reviews dynamic programming using knowledge states. Finally, the section explores the assumption that the goal is reachable from any state in the state space. We make this assumption in order to ensure that a randomized strategy will not accidentally enter a trap state from which it cannot reach the goal.

Sections 7 and 8 present the basic definitions of randomized strategies. First, Section 7 considers a simple form of randomization, in which the executive makes a single guess prior to execution of a plan. The guess chooses between different possible plans. The executive executes the plan, then guesses again, and so on, until one of the partial plans successfully attains the goal. The section examines the conditions under which this loop may be repeated and reliably terminated with task attainment.

Next, Section 8 generalizes the notion of randomization to permit several guesses during the execution of a partial plan. Effectively, randomization at run-time consists of randomly choosing between a collection of knowledge states whose union comprises the true knowledge state of the system.

Finally, Section 9 presents some interesting results that relate the complexity of guaranteed and randomized strategies for solving near-sensorless tasks. These are

5. Basic Definitions

5.1. Discrete States, Actions, and Tasks. We will model a task as a problem on some state space. The state space should consist of all the parameters of a system that are required to predict its future behavior. In other words, knowing the current state of the system and some action applied to the system, it should be possible to predict the resulting state or states of the system without reference to past states.

In this paper, we will focus on discrete tasks, meaning that both the set of states and the set of available actions are finite. We denote the states by

\[ \mathcal{S} = \{s_1, s_2, \ldots, s_n\}, \]

and the actions by

\[ \mathcal{A} = \{A_1, A_2, \ldots, A_m\}. \]

We will specify a task by a set of goal states \( \mathcal{G} \subset \mathcal{S} \), whose recognizable attainment constitutes completion of the task. By recognizable attainment we mean that the system both is in a goal state and knows that it is in a goal state. (The system need not distinguish between different goal states.)

In addition to the goal, it is sometimes useful to include in the specification of a task the possible starting states \( \mathcal{S} \) of the system, with \( \mathcal{G} \subset \mathcal{S} \).

5.2. Discrete Tasks. We should convince ourselves that there are tasks that may be represented in discrete terms. An example of a discrete state space is given by the possible orientations of a polyhedral part resting on a horizontal table under the influence of gravity. Figure 2 depicts the planar case. The figure shows three stable orientations of a planar part resting on a horizontal table. By tilting the table for a short amount of time the part can be made to roll between different such configurations. While the analysis of the forces required to move the part may require consideration of a continuous space, once this analysis has been performed, it is sufficient to consider the resulting discrete space in planning operations to orient the part stably.

The representation of tasks is a difficult issue. In some cases, problems that appear to reside in a continuum state space may be transformed into equivalent

Fig. 2. This figure indicates three stable configurations of a planar Allen wrench lying on a horizontal table. These configurations may be used to define a discrete state space.
or similar problems that reside in finite state spaces. The details of the transformation tend to be task-specific, although often stability under some set of actions may be used as a criterion in defining the discrete states. The work of Brost [7], [8] involves such a transformation for the problem of pushing and grasping planar polygonal objects. Mani and Wilson [32] used a similar transformation in their work on pushing, and Erdmann and Mason [23] employed a stable-under-gravity transformation in their work on orienting planar parts in a tray.

5.3. Nondeterministic Actions. We will usually consider actions to be nondeterministic. Specifically, given some starting state $s$, the result of applying an action $A$ may be any one of a possible set of states $F_A(s) = \{s_1, s_2, \ldots, s_n\} \subseteq \mathcal{S}$. This set is called the forward projection of the state $s$ under action $A$. Figure 3 is a graphical representation of a nondeterministic action. In the figure, action $A_1$ may have one of three results when applied to state $s_0$, but has precisely one result when applied to states $s_1$, $s_2$, or $s_3$. Symbolically, we will write this as

$$A_1: \begin{align*}
  &s_0 \rightarrow s_1, s_2, s_3, \\
  &s_1 \rightarrow s_1, \\
  &s_2 \rightarrow s_2, \\
  &s_3 \rightarrow s_3.
\end{align*}$$

As an example of a nondeterministic action, consider an Allen wrench in contact with a tabletop, as shown in the top portion of Figure 4. Suppose a force is applied through the center of mass as shown. Depending upon the coefficient of friction, the accuracy of the applied force, the position of the center of mass, and so forth, there are two possible final stable states of the Allen wrench. These are shown in the lower portion of Figure 4. If the parameters determining the motion of the wrench cannot be modeled accurately, for instance, if the coefficient of friction is unknown, then the action should be modeled nondeterministically.

5.4. Probabilistic Actions. Occasionally, it will be useful to consider probabilistic actions. Probabilistic actions are a special case of nondeterministic actions, in which it is possible to assign a probability density function to the forward projection. Consider the forward projection $F_A(s) = \{s_1, \ldots, s_n\}$ of some state $s$. For a probabilistic action $A$, we can assign to each state $s_i$ a probability $p_i$. This means that if the system is initially in state $s$, and we execute action $A$, then state $s_i$ will be attained with probability $p_i$.

5.5. Sensors. We will model a sensor as the composition of two maps. The first map associates, with any state, a set of possible sensor values that the system might observe when in that state

$$\xi: \mathcal{S} \rightarrow 2^D,$$

$$s \mapsto \xi(s).$$

Fig. 3. Graphical representation of a nondeterministic action $A_1$. 

Fig. 4. The force applied to the Allen wrench at the top of the figure will cause the wrench either to slide without rotation or to rotate and possibly slide. The actual motion depends on the coefficient of friction. If the coefficient of friction is not known it is useful to model the force as a nondeterministic action.
In other words, \( \xi(s) = \{s_1^*, \ldots, s_n^*\} \) for some set of sensor values \( s_i^* \in D \). Here \( D \) is the space of sensor values; we often just let \( D = \mathcal{S} \).

The set-valued nature of \( \xi \) permits us to encode nondeterminism in the sensor's output. As we did for actions, we could also model this nondeterminism probabilistically, by assigning probabilities to the elements of \( \xi(s) \).

The second map interprets each sensor value \( s^* \)

\[
1: \quad D \to 2^\mathcal{S},

s^* \mapsto I(s^*).
\]

In other words, \( I(s^*) = \{s_1, \ldots, s_n\} \) for some set of states \( s_i \in \mathcal{S} \). This means that whenever the system observes sensor value \( s^* \), then it interprets this sensor value to mean that the actual state of the system can be any one of the states in the set \( I(s^*) \). The set-valued nature of \( I \) permits us to encode both the nondeterminism of the sensor and our potential inability to accurately interpret the sensor. In the probabilistic setting we can assign conditional probabilities to the elements of \( I(s^*) \).

We often collapse these two maps into one map

\[
\Xi: \quad \mathcal{S} \to 2^\mathcal{S},

s \mapsto \Xi(s),
\]

with

\[
\Xi(s) = \bigcup_{s^* \in I(s)} \{I(s^*)\}.
\]

In other words, for any state \( s \), \( \Xi(s) \) is a collection of sets, say \( \Xi(s) = \{ I_1, \ldots, I_h \} \). We refer to each \( I_i \) as a sensory interpretation set. \( \Xi(s) \) describes all possible sensory interpretation sets that might arise at run-time whenever the system is in state \( s \). This means that at run-time the physical sensor can return some value whose interpretation is one of the sets \( I_i \).

5.6 Constraints on Sensors. In order to avoid some technical difficulties we need to impose constraints on the form of \( \Xi \). We summarize these constraints with the following requirement. For further details see [20].

**FULL SENSING CONSISTENCY REQUIREMENT.** Let \( \Xi \) be a sensing function on a state space \( \mathcal{S} \). Denote by \( \Xi(\mathcal{S}) \) the set of all possible sensory interpretation sets, i.e., \( \Xi(\mathcal{S}) = \bigcup_{s \in \mathcal{S}} \Xi(s) \). We say that a sensing function satisfies the full sensing consistency requirement if the following condition holds for all states \( s \in \mathcal{S} \):

\[
I \in \Xi(s) \quad \text{if, and only if,} \quad s \in I \quad \text{and} \quad I \in \Xi(\mathcal{S}).
\]

In other words, if a state \( s \) can give rise to a sensory interpretation set \( I \), then that interpretation set must include the state itself. Conversely, if a state \( s \) appears in some interpretation set \( I \), then it must be possible to get \( I \) as an interpretation when the system is actually in state \( s \).

6. Planning with Uncertainty. In this section we outline the form of a standard planner for generating guaranteed strategies in the presence of sensing and control uncertainty. We start with the perfect sensing case, then build to the general sensing case via the notion of knowledge states. Throughout we permit control to be imperfect. Finally, we consider a connectivity assumption that ensures that the goal is reachable from all states despite control uncertainty.

6.1 Dynamic Programming. The approach that we will use for generating plans in the presence of control uncertainty is to backchain from the goal. Given a goal, the backchaining planner constructs all states from which it is possible to attain the goal by executing a single action. This collection of states defines a new subgoal from which the planner can backchain further. The process is repeated until a subgoal is constructed that contains the initial state of the system, if that is possible. The resulting backchaining graph describes a plan that will reliably attain the goal given perfect sensing despite uncertainty in the actions.

This process is known as Dynamic Programming [3]. It is often stated in a more general setting in which the actions are probabilistic and the objective is to attain the goal while minimizing some cost function.

6.1.1 A Probabilistic Example. The following example consists of a series of states connected by actions that have probabilistic transitions (see Figure 5). After any transition, sensors report the resulting state with complete accuracy. The starting state can also be sensed with perfect accuracy. The task is to determine a mapping from states to actions that maximizes the probability of attaining the goal in a specific number of steps. This mapping constitutes a plan or a strategy for attaining the goal.

The basic idea of dynamic programming is to maximize (or minimize) some value function in terms of the actions available and the number of steps remaining to be executed. At each stage, an action is selected for each state that would maximize the value function given that there remain a certain number of steps to be executed. This maximization is performed by first recursively determining the maximum values obtainable for each state given one fewer step, then selecting an
Table 1. Probabilities of success; optimal actions.

<table>
<thead>
<tr>
<th>Steps remaining</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>States</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A₂</td>
<td>1/20</td>
<td></td>
<td></td>
<td></td>
<td>A₄</td>
</tr>
<tr>
<td>A₃</td>
<td>1/4</td>
<td>A₄</td>
<td></td>
<td></td>
<td>s₁</td>
</tr>
<tr>
<td>A₄</td>
<td>1/10</td>
<td>A₃</td>
<td></td>
<td></td>
<td>s₂</td>
</tr>
<tr>
<td>A₅</td>
<td></td>
<td>A₃</td>
<td></td>
<td></td>
<td>s₃</td>
</tr>
<tr>
<td>stop</td>
<td></td>
<td>stop</td>
<td></td>
<td>stop</td>
<td>s₄</td>
</tr>
</tbody>
</table>

action for the current state that maximizes the expected value. One starts the whole process off by assigning values to each state that reflect the value of the value function if no actions whatsoever remain to be executed. When looking for strategies with maximal probability of success, the value function represents the probability of achieving the goal in the remaining steps. Goal states are initially assigned a value of 1; nongoal states a value of 0. Further, the value in the kth stage of the computation for a particular state is the probability of attaining the goal from that state in at most k steps, assuming that the system can sense perfectly and that it always executes the maximizing action at each state.

The backchaining maximization of dynamic programming may be depicted by a table (see Table 1). The columns of the table correspond to the stages in the backchaining process; the rows correspond to the states of the execution system. Counting from right to left, an entry in the kth column of the table for state sᵢ specifies the action to be taken at run-time along with the optimal probability of eventual goal attainment, given that there remain k steps in which to execute actions and that the system’s current state is sᵢ.

Table 1 reflects the backchaining computations for the graph of Figure 5. The table shows that the goal can be achieved with certainty from any state using no more than three steps.

6.1.2. A Nondeterministic Example. Suppose now that the transition graph of Figure 5 is nondeterministic rather than probabilistic. In this case the value function to be maximized by the dynamic programming approach is a Boolean function. The dynamic programming table (Table 2) for the nondeterministic case is similar to the table for the probabilistic case. A blank entry in the table indicates that success cannot be guaranteed in the number of steps remaining from that state. Conversely, an entry with an action Aᵢ indicates that eventual goal attainment is guaranteed in the number of steps remaining if the system executes action Aᵢ.

6.2. Knowledge States. When sensors are perfect, we may plan and execute strategies in the state space of the system. However, once the sensors are imperfect, the system may not be able to ascertain its precise state at run-time. Instead, it is appropriate to consider the system’s knowledge states. A knowledge state describes the system’s possible configurations at run-time, as best the system can determine using a history of its past and current sensory information as well as its past actions. In the nondeterministic setting a knowledge state is a set of possible system states. In the probabilistic setting it is a probability distribution over the system’s state space. We consider here the nondeterministic case.

6.2.1. Forward Projection. In the nondeterministic case, the space of knowledge states is the set of all subsets of the state space, namely 2ˢ. Given a set K₁ of possible states that the system could be in, and an action Aᵢ, the result of executing action Aᵢ is a new knowledge state K₂, given by

\[ K₂ = \bigcup_{s \in K₁} F₄(s). \]

In other words, K₂ is the union of all the possible nondeterministic transitions resulting from possible states in K₁. Notice that this knowledge is equivalent both at execution time and at planning time. The process of forming K₂ is called forward projecting set K₁ under action Aᵢ, and is written K₂ = F₄(K₁).

Forward projections possess a nice property. The forward projection of a collection of sets is just the union of the forward projections of the individual sets. This is summarized in the following lemma.

**Lemma 6.1.** Let \( \{Kᵢ\} \) be a collection of knowledge states, and let Aᵢ be a nondeterministic action. Then

\[ F₄(\bigcup_{i} Kᵢ) = \bigcup_{i} F₄(Kᵢ). \]

**Proof.** Clear from the definition. \( \square \)

6.2.2. Sensing. Given a prior knowledge state K₁ and a current sensory interpretation set I, the resulting knowledge state is K₂ = K₁ ∩ I. Intersection is the correct operation since the full sensing consistency requirement holds. Without this requirement, a more complicated procedure is required. We refer the interested reader to [20]. Observe that if K₂ = \( \varnothing \), then the observed sensor value is inconsistent with the previously computed possible states of the system.
Randomization for Robot Tasks

space \(2^\mathcal{S}\) as

\[ A: K_1 \rightarrow K_1^1, K_1^2, \ldots, K_1^n. \]

This means that at execution time the action \(A\) (which corresponds to performing action \(A\) followed by some sensory operation), will transit nondeterministically from knowledge state \(K_1\) to one of the knowledge states \(K_1^n\). By construction, our sensing guarantees that the execution system will know precisely which knowledge state has been attained. Thus the problem is a perfect-sensing problem, but in the space of knowledge states.

Since the problem has a perfect-sensing function, we can apply the techniques previously discussed for such problems. In particular, we can plan strategies for achieving a goal state (and knowing that it has been achieved), by backchaining from the goal in the space \(2^\mathcal{S}\). This amounts to applying the dynamic programming discussed in Section 6.1, but in the space of knowledge states \(2^\mathcal{S}\).

6.4. Connectivity Assumption. A characteristic of randomized strategies, as we shall see, is the need to execute partial plans repeatedly until one such plan succeeds. In order to be able to guarantee that the goal is eventually reachable, it must be the case that both the goal is, in fact, reachable from the start state, and that the system will never enter a trap state from which the goal becomes unreachable. We would thus like to make a connectivity assumption that ensures that the goal is reachable from every possible state of the system.

6.4.1. Probabilistic Setting. In the case that actions are specified as probabilistic transitions, the connectivity assumption amounts to the condition that the transitive closure of each state in the induced transition graph contains a goal state. The transitive closure of a state in a directed graph is the set of all states reachable from that state by some directed path. By the induced transition graph we mean the directed graph whose vertex set is the set \(\mathcal{S}\) of underlying states, and whose directed arcs are given by the set of all transition arcs whose associated probabilities are nonzero. This set is computed by considering the set \(\mathcal{S}\) of all possible actions.

6.4.2. Nondeterministic Setting. In the case that actions are specified as nondeterministic transitions, we need a stronger condition than for the probabilistic case. In order to understand the difference between the nondeterministic and the probabilistic cases, consider Figure 7. In both Part A and Part B, if we interpret all the arcs as probabilistic arcs with positive transition probabilities, then the task satisfies the connectivity assumption. In other words, from any state there is a sequence of transitions that attains the goal with nonzero probability. However, if we interpret the arcs as worst-case transitions, then only the task of Part B satisfies the connectivity assumption. In Part A, from a worst-case point of view, there is a possibility that the system will forever loop between states \(s_i\) and \(s_j\).

Let us formalize the connectivity assumption. As we stated, even in the worst case there should for each state exist a sequence of actions that leads to the goal. Recall that a nondeterministic action \(A\) can cause a given state \(s\) to transit
nondeterministically to any one of a set of states \([s_1, \ldots, s_4]\). There is no further information in the model, and we must thus be prepared that any one of the transitions can occur. We will refer to an instantiation of such a nondeterministic transition as a particular choice \(s_i\). In other words, on a particular execution of action \(A\) while the system is in state \(s\), the result is instantiated as state \(s_i\). By an instantiation of all possible actions we mean a choice \(s_i\) for all actions \(A \in \mathcal{A}\) at all possible states \(s \in \mathcal{S}\). An instantiation of all possible actions yields a directed graph whose vertex set is \(\mathcal{S}\) and whose arcs are the directed arcs defined by the instantiation. We will refer to a particular such graph as an instantiated transition graph. Figure 8 shows the four instantiated transition graphs that are possible by instantiating in all possible ways the nondeterministic actions of the graph in Part A of Figure 7. Notice that for one of the graphs, two states are disconnected from the goal. This says that in a worst-case scenario it might not be possible to reach the goal.

**Definition.** We will say that it is certain possible to reach a set of goal states \(\mathcal{G}\) from a given state \(s\) if for any instantiated transition graph there is some path that leads from the state \(s\) to some goal state in \(\mathcal{G}\).

This definition captures the notion that no matter how the world behaves within the nondeterminism allowed by the specified actions, there is some path for attaining the goal. This is precisely our connectivity assumption.

### 6.4.3. Goal Reachability and Perfect Sensing

It turns out that the connectivity assumption in the nondeterministic setting is equivalent to the existence of a guaranteed perfect-sensing strategy. This is proved below. Furthermore, the perfect-sensing strategy need not have more steps than there are states.

**Claim 6.2.** Let \((\mathcal{S}, \mathcal{A}, \Xi, \mathcal{G})\) be a discrete planning problem, where \(\mathcal{S}\) is the set of states, \(\mathcal{A}\) is the set of nondeterministic actions, \(\Xi\) is the sensing function, and \(\mathcal{G}\) is the set of goal states.

It is certain possible to reach \(\mathcal{G}\) from any state \(s \in \mathcal{S}\) if, and only if, there exists a guaranteed perfect-sensing strategy for attaining \(\mathcal{G}\) from any state \(s \in \mathcal{S}\).

**Proof.** First, suppose that there exists a perfect-sensing strategy that is guaranteed to move the system from any state \(s\) to some goal state. Then for any instantiated transition graph there must be a path from \(s\) to \(\mathcal{G}\). This path may be determined by executing the perfect-sensing strategy while selecting action transitions as prescribed by the instantiated transition graph.

Conversely, suppose that for any instantiated transition graph and any state \(s \in \mathcal{S}\) there is a path from \(s\) to \(\mathcal{G}\). We would like to exhibit a perfect-sensing strategy for attaining the goal \(\mathcal{G}\) from any state \(s \in \mathcal{S}\).

We will construct a collection of sets of states \(\mathcal{S}_0, \ldots, \mathcal{S}_q\), for some \(q \leq |\mathcal{S}|\).
The intuition behind these sets is that a state is in $S_i$ if there exists a perfect-sensing strategy for attaining the goal in at most $i$ steps, and if there is some possible instantiated transition graph for which $i$ steps are actually required.

Define $S_k$ to be the goal set $S$. Clearly, there is a perfect-sensing strategy that attains a goal state from any state in $S_0$, requiring zero steps. Suppose that $S_i$ has been defined, and that there exists a perfect-sensing strategy defined on the union $\bigcup_{i=0}^{i} S_i$. The perfect-sensing strategy is assumed to attain a goal state from any state in the union without ever passing through any state in the complement $S - \bigcup_{i=0}^{i} S_i$. Define $S_{k+1}$ to be the set of all states in this complement for which there exists some action that attains a state in $\bigcup_{i=0}^{k} S_i$ in a single step. In other words,

$$S_{k+1} = \left\{ s \in S \setminus \bigcup_{i=0}^{k} S_i \mid F_A(s) \subseteq \bigcup_{i=0}^{k} S_i \text{ for some action } A = A(s) \right\}.$$  

We need to show that $S_{k+1}$ is not empty, unless $S = \bigcup_{i=0}^{k} S_i$. Once we establish this, then the existence of a perfect-sensing strategy on the union $\bigcup_{i=0}^{k+1} S_i$ that does not touch the complement of this union will be clear. Furthermore, since each set $S_i$ is nonempty, there can be at most $|S|$ of them.

Let us write $S_k = S \setminus \bigcup_{i=0}^{k} S_i$. Suppose that $S_k \neq \emptyset$, but that $S_{k+1} = \emptyset$. This says that for every state $s \in S_k$ and every action $A$, the intersection of the forward projection $F_A(s)$ with $S_k$ is nonempty. Said differently, for each state $s \in S_k$, and each action $A$, there is an instantiation that causes $s$ to traverse to a state in $S_k$. This means that there is an instantiated transition graph for which the set $S_k$ is completely disconnected from the goal. That violates the assumption of the claim, and thus we see that $S_{k+1} \neq \emptyset$.

The next claim implies that a perfect-sensing strategy for attaining the goal need not be very long. The proof of the claim is contained in the proof of the previous claim.

**Claim 6.3.** Let $(S, A, \Xi, \mathcal{G})$ be a discrete planning problem. Assume that $\Xi$ is the perfect-sensing function.

Suppose that there exists a guaranteed strategy for moving from any state to the goal set $\mathcal{G}$. Then there exists a sequence of nonempty disjoint sets $S_0, S_1, \ldots, S_k$ that cover $S$, such that states in the set $S_{k+1}$ can traverse to states in the union $\bigcup_{i=0}^{k} S_i$ in a single step. Furthermore, $S_0 = \emptyset$, and $k \leq |S| - |\mathcal{G}|$.

7. Randomization with Nondeterministic Actions. As we have formulated the problem of planning in the presence of uncertainty thus far, a planner constructs a circuit of knowledge states by backtracking from the goal. The problem is considered solved if one of these knowledge states contains the initial state of the system. This is what is meant by a guaranteed solution throughout this paper. For some tasks, however, there is no such guaranteed solution. See again the example of Figure 1 in Section 2.3.

In this section we consider randomization that involves guessing the initial state of the system, executing a deterministic plan, then repeating the whole process if necessary. We examine several applicability conditions for such a randomized plan to succeed. These conditions lead us to assume that the goal is always recognizable and that the range of initial states is the entire state space. In the next section we consider general randomization in which a system can make random guesses at any step of execution.

7.1. Guessing the Starting State. We will view the planning process in terms of dynamic programming in the space of knowledge states. Consider the column in the dynamic programming table that corresponds to $k$ steps remaining in the strategy. Consider all the knowledge states whose entries in this column are not blank. Call them $\{K_{k,1}, K_{k,2}, \ldots, K_{k,J}\}$. These are all the knowledge states for which there exists a tree of depth at most $k$ of conditional actions guaranteed to attain the goal. Suppose that $S_0$ is the initial knowledge state of the system. If $S_0 = K_{k,j}$ for some $j$, then there is a guaranteed strategy consisting of no more than $k$ steps that will attain the goal.

More generally, however, we may have that

$$S_0 \subseteq \bigcup_{j=0}^{K_{k,J}} K_{k,j}.$$  

In that case there exists a randomized strategy for attaining the goal. The randomized strategy randomly chooses one of the knowledge states $K_{k,j}$, then executes the proper sequence of actions designed to attain the goal from that knowledge state. Since the states $\{K_{k,1}, K_{k,2}, \ldots, K_{k,J}\}$ cover $S_0$, we can assume that at most $q$ of them cover $S_0$, with $q \leq |S|$. Thus the system only needs to guess between $q$ different knowledge states. If the system guesses uniformly, then the guess will be correct with probability no less than $1/q$. Thus with probability at least $1/q$ the strategy will attain the goal in at most $k$ steps.

7.2. Execution Traces and Goal Failure. In order to gain some intuition as to the types of execution traces that might occur, let us introduce some additional notation. Given a starting knowledge state $K$, and a nondeterministic action $A$ in knowledge space (see Section 6.3), let us write the effect at execution time of this action on $K$ as $K; A; I$, where $A$ is the generating nondeterministic action in the underlying state space, and $I$ is a sensory interpretation set that is returned by the sensor at execution time. Specifically, $K; A; I$ is the knowledge state $F_A(K) \cap I$. More generally, given a sequence of actions $\{A_1, A_2, \ldots, A_i\}$ and an associated sequence of run-time sensory interpretation sets $\{I_1, I_2, \ldots, I_i\}$, the effect on $K$ will be denoted by $K; A_1; I_1; A_2; I_2; \ldots; A_i; I_i$.

Suppose now that the system guesses that the initial knowledge state is the set $K_0$. The strategy for attaining the goal $G$ from $K_0$ is encoded in the dynamic programming table. Suppose that the first action, $A_1$, is taken from the entry for $K_0$ in the $k$th column of the dynamic programming table. Execution of $A_1$ involves execution of some action $A_1$ on the underlying state space, followed by some
sensory observation that yields a sensory interpretation set \( I_1 \). Once \( \bar{A}_1 \) has been executed, the resulting knowledge state determines the next action to perform. This action \( \bar{A}_2 \) is again encoded in the dynamic programming table. Action \( A_2 \) in turn results in some new run-time sensory interpretation set \( I_2 \), and so forth. If the system ever encounters the empty set as a knowledge state then it stops execution.

If the initial state of the system is indeed contained in the guessed starting knowledge state \( K_0 \), then after \( k \) actions the resulting knowledge state will be nonempty and inside the goal, i.e., \( \emptyset \neq K_0 \cap A_1; I_1; A_2; I_2; \ldots; A_k; I_k \subseteq \mathcal{G} \). The precise sequence is of course not determined until execution time. On the other hand, if the initial state of the system is not contained in \( K_0 \), then the final knowledge state may or may not be empty, and may or may not accurately depict whether the goal has been attained.

Now consider the effect of the sequence \( A_1; I_1; A_2; I_2; \ldots; A_k; I_k \) on knowledge states other than the guessed starting knowledge state \( K_0 \). In particular, suppose that for each possible starting state \( s_0 \), the final knowledge state \( \{ \emptyset s_0 \cap I_1; A_1; I_2; \ldots; A_k; I_k \} \) is either the empty set \( \emptyset \) or lies inside the goal \( \mathcal{G} \). Then clearly the goal must have been attained even if the initial guess \( K_0 \) was wrong. Conversely, suppose that for some state \( s_0 \), the final knowledge state includes states outside of the goal. If \( s_0 \) could have been a starting state of the system, then we cannot be sure that the system has entered the goal. This establishes the following claim.

**Claim 7.1.** Consider a discrete planning problem \((\mathcal{G}, \mathcal{A}, \mathcal{I}, \mathcal{G})\) for which the full sensing consistency requirement holds. Suppose that the initial state of the system is known to lie in some subset \( \mathcal{G}_0 \subseteq \mathcal{G} \). Suppose further that there exists a guaranteed strategy for attaining the goal in \( k \) steps if the initial state were actually known to be in the set \( \mathcal{G}_0 \), with \( K_0 \cap \mathcal{G}_0 \subseteq \mathcal{G}_0 \). Imagine that the system executes this strategy as if the initial knowledge state were indeed \( K_0 \). Let the execution trace be given by \( A_1; I_1; A_2; I_2; \ldots; A_k; I_k \). Then the system is guaranteed to have attained the goal if, and only if, \( \mathcal{G}_0; A_1; I_1; A_2; I_2; \ldots; A_k; I_k \subseteq \mathcal{G} \).

(Notice that \( K_0; A_1; I_1; A_2; I_2; \ldots; A_k; I_k \) may be the empty set, if the initial state of the system is not in \( K_0 \). However, the knowledge state \( \mathcal{G}_0; A_1; I_1; A_2; I_2; \ldots; A_k; I_k \) must be nonempty, since the system is known to have started in the set \( \mathcal{G}_0 \), and since sensing is consistent.)

The claim suggests that a randomizing system should retain two knowledge states, one derived from the actual starting knowledge state \( \mathcal{G}_0 \), the other from the guessed knowledge state \( K_0 \). The knowledge state derived from \( \mathcal{G}_0 \) permits the system to assess goal attainment faithfully. The knowledge state derived from \( K_0 \) tells the system the next action to execute by indexing into the dynamic programming table.

**Definition.** Let us define the phrase the strategy is assured of reliable goal recognition from \( K_0 \) to mean that any execution trace of the strategy, which transforms \( K_0 \) into a nonempty knowledge state within the goal, actually implies goal attainment.

With the same hypotheses as the claim above, we obtain the following corollary. The corollary is essentially a restatement of the definition of reliable goal recognition.

**Corollary 7.2.** Suppose that a randomized strategy guesses that the system is in \( K_0 \), and plans to execute the guaranteed strategy for \( K_0 \), even though the actual state of the system may be in \( \mathcal{G}_0 \cap \mathcal{G} \). The strategy is assured of reliable goal recognition from \( K_0 \) if, and only if, \( \mathcal{G}_0; A_1; I_1; A_2; I_2; \ldots; A_k; I_k \subseteq \mathcal{G} \) for all possible execution traces that might occur for which \( \emptyset \neq \mathcal{G}_0; A_1; I_1; A_2; I_2; \ldots; A_k; I_k \subseteq \mathcal{G} \).

7.3. Repeated Goal Reachability. The second issue that needs to be addressed concerns the behavior of the randomized strategy upon failure. Thus far, we have merely asked that the strategy guess a starting knowledge state and execute a strategy guaranteed to achieve the goal if the guess is correct. If the guess is incorrect and the strategy fails to achieve the goal, then we need to worry about how to proceed. This leads to a second applicability condition.

**Definition.** A randomized strategy repeatedly guesses its initial starting region \( K_0 \), then executes some guaranteed strategy for attaining the goal from \( K_0 \). The execution terminates either with recognizable goal attainment or failure. We will refer to each such guess and strategy execution as a single guessing loop of the randomized strategy.

**Definition.** We will say that a randomized strategy may be reliably restarted from \( K_0 \) if, whenever the strategy fails to attain the goal recognizably from \( K_0 \) on a single guessing loop, the system recognizably lies within its initial starting region \( \mathcal{G}_0 \).

The following claim establishes a complement to Corollary 7.2. The claim is essentially a restatement of the definition of reliable restart, but with a slightly more explicit condition.

**Claim 7.3.** Assume the hypotheses of Claim 7.1, and suppose that the guaranteed strategy for \( K_0 \) is assured of reliable goal recognition from \( K_0 \). The randomized strategy may be reliably restarted from \( K_0 \) if, and only if, \( \mathcal{G}_0; A_1; I_1; A_2; I_2; \ldots; A_k; I_k \subseteq \mathcal{G} \) for all possible execution traces that might occur which fail to attain the goal recognizably and for which \( K_0; A_1; I_1; A_2; I_2; \ldots; A_k; I_k \)

is either empty or contains nongoal points.

7.4. Observations and Assumptions. Notice that if a strategy both is assured of reliable goal recognition and may be reliably restarted from all relevant knowledge states \( K_0 \) that cover \( \mathcal{G}_0 \), then whenever a single guessing loop of the randomized strategy is executed from the region \( \mathcal{G}_0 \), it is guaranteed to attain recognizability either the goal or again the region \( \mathcal{G}_0 \). This condition is in appearance very similar to Donald's EDR condition (see page 100 of [16]), which insists that a strategy be guaranteed to attain recognizably either the goal or a region called the failure region from which success is not possible.
Randomization for Robot Tasks

Unfortunately, the condition does not work in reverse. In other words, the converse statement that recognizable attainment of the goal \( g \) or the start region \( K_0 \) implies reliable goal recognition and reliable restart is simply not true. After all, if the start region is the entire state space, then any strategy is guaranteed to attain recognizable either the goal or the start region, but the strategy need not satisfy the condition of reliable goal recognition.

The failure of the converse statement suggests that verifying reliable goal recognition and reliable restart are in general quite difficult. However, they are easily satisfiable conditions if we make two special assumptions.

**Assumption of Goal Recognizability.** First, we will assume that the goal is recognizable independent of any particular execution. This means that if the sensor signals goal attainment then the goal has indeed been attained, and conversely, if the goal is entered then the sensor will signal goal attainment.

**Assumption of Covering Start Region.** Second, we will assume that the start region for any guessing strategy is the entire state space. In general, we can relax this assumption by considering only that portion of the state space that might ever be traversed.

8. Multiguess Randomization. Thus far we have only dealt with randomization that guesses the starting state of the system. In general, it is equally possible to consider sequences of several guesses. Suppose that at some point during execution the system encounters a knowledge state that is the union of several smaller knowledge states. Instead of executing a strategy applicable to the larger knowledge state, the system could simply guess between the smaller states, then use strategies appropriate for each of these. In terms of planning such strategies, the standard preimage or dynamic programming approaches continue to apply, but with an additional operator. Call this operator SELECT. SELECT operates as follows.

8.1. An Augmented Dynamic Programming Table. First, let us augment the dynamic programming table. Each column in the dynamic programming table will contain three types of entries, namely, BLANK, GUARANTEED, and RANDOMIZED. The intuition is that BLANK and GUARANTEED are as before. Specifically, if the entry for knowledge state \( K \) is GUARANTEED then there exists a tree of conditional actions that is guaranteed to attain the goal recognizable, assuming that the initial state of the system is indeed inside \( K \). A BLANK entry implies that there is no such strategy, and also that there is no strategy involving random choices. Finally, a RANDOMIZED entry for a knowledge state \( K \) means that there is a tree of operations that has some probability of attaining the goal from \( K \). The operations involve both standard nondeterministic actions and the guessing operator SELECT.

It is sometimes also useful to distinguish between different RANDOMIZED entries based on an estimate of the worst-case probability of attaining the goal. For a given knowledge state one easily-computed estimate is the minimum product of guessing probabilities along possible execution paths from that knowledge state to the goal. These estimates do not take into account serendipitous goal attainment resulting from wrong guesses, and thus may considerably underestimate the actual probabilities of success. Nonetheless, in some situations, these estimates provide useful lower bounds for comparing different strategies.

8.2. Planning. The planner initializes the dynamic programming table by filling in column zero. All nonempty subsets of the goals are labeled GUARANTEED; all other sets are labeled BLANK. Suppose that the planner has backchained to the \( k \)th column of the dynamic programming table, and is currently considering the \((k+1)\)st column. First the planner fills in all entries using only the standard nondeterministic actions. In other words, for each knowledge state \( K \), if there is an action \( A \) of the form \( A \rightarrow K_1, \ldots, K_k \), and each of the \( K_i \) has a non-BLANK entry in the \( k \)th column, then the entry for \( K \) in the \((k+1)\)st column may be taken to be \( A \). If there are several such actions \( A \), then the planner may wish to distinguish between different actions by considering the labels of the entries for the knowledge states \( K_i \) to which the actions can transit. In particular, suppose RANDOMIZED entries actually include worst-case probabilities of success. Then we assign the probability 0 to every BLANK entry, and the probability 1 to every GUARANTEED entry. The planner may then associate with each action \( A \) a worst-case probability of success. Specifically, if \( p_e \) is the probability of success associated with the entry for \( K_i \) in the \( k \)th column, then the probability of success \( p_{\text{w}} \) for \( A \) may be taken as \( \min_{K} |p_e| \). If several actions \( A \) are applicable at the current knowledge state, then the planner can then select that action which maximizes \( p_{\text{w}} \). In particular, if there is an action that only transits to GUARANTEED states, then the planner should select it. Similarly, if all actions have worst-case probability zero of success, then the planner should simply leave the entry for \( K \) blank. Once an action has been selected, it provides a label and an estimated probability of success for the current knowledge state \( K \). The label is GUARANTEED if all the \( K_i \) are GUARANTEED; otherwise, the label is RANDOMIZED.

Once the entries in the \((k+1)\)st column have been filled in in this way, the planner next considers all remaining BLANK entries in that column. In particular, suppose \( K \) is a nonempty knowledge state whose entry is BLANK. If the knowledge state can be written as a finite union of non-BLANK states \( \{K_1, \ldots, K_n\} \), then the SELECT operator comes into play. It provides a transition from \( K \) to one of the \( K_i \) via randomization. Specifically, at run-time the system will select knowledge state \( K_i \) with probability \( q_i \). During the planning phase, the system may pick \( \{q_i\} \) as it wishes, so long as the probabilities add up to 1. The entry for \( K \) in the \((k+1)\)st column of the dynamic programming table becomes a RANDOMIZED entry. It consists of a SELECT operation and a worst-case probability of success. The SELECT operation specifies the set \( \{K_i\} \) and the probabilities \( \{q_i\} \). Suppose that \( p_{\text{w}} \) is the worst-case probability of success associated with knowledge state \( K_i \) in the \((k+1)\)st column. Then the worst-case probability of success for \( K \) has the value

\[
\min_{K} \sum_{i} q_i p_{\text{w}}.
\]
The examples in this section are abstract examples on graphs. We do not investigate in detail whether these graphs can be realized by physical devices. However, in Section 9.5, we describe a physical device that has some of the same properties as the first example. The first example demonstrates a task for which there exists a guaranteed strategy for performing the task, but which requires exponential time to plan and execute. The key observation is that there also exists a randomized strategy that only requires quadratic expected time to attain the goal. This example indicates that some tasks may be performed more quickly with a randomized strategy than with a guaranteed strategy. The second example in this section exhibits a task for which both guaranteed strategies and randomized strategies require exponential execution time. Thus this example indicates that it is not always possible to reduce execution time from exponential to polynomial time with the aid of randomization.

We will first briefly outline how the general backchaining planners discussed earlier specialize to the sensorless case, then indicate the relationship between sensorless and near-sensorless tasks, and finally proceed to the examples.

8.3. Execution. At run-time, suppose nominally there are k steps remaining and the current knowledge state is K. If the entry for K is BLANK, then execution of this particular guessing loop stops, and a new loop is restarted, if possible. If the entry for K is not BLANK, but contains an action A, then the system executes that action, thereby proceeding to the new column of the dynamic programming table. If the entry for K contains a SELECT operation, then the system randomly chooses one of the possible actions specified by this SELECT operation, using the guessing probabilities q_i. Having chosen a set K of actions, the system then executes the action stored in the entry for K in the kth column of the dynamic programming table. If ever the goal is attained, execution stops.

Starting or restarting the guessing loop entails determining an initial knowledge state by performing a sensory operation and intersecting the resulting sensory interpretation set with the set A_n, in which all motions are assumed to occur. An alternative is to restart the guessing loop by considering the set A_n+1 = A_n \cap A_1 \cap A_2 \cap \ldots \cap A_i \cap \ldots \cap A_k \cap \ldots \cap A_n \cap A_k \cap \ldots \cap A_2 \cap A_1 \cap A_n, where A_n is the initial knowledge state at the start of the i-th iteration of the guessing loop. This procedure preserves full history independent of any guesses, and thereby may reduce the number of states between which the strategy must guess on each new iteration.

9. Some Complexity Results for Randomized Near-Sensorless Tasks. In this section we consider a special form of discrete tasks, in which the sensors provide no information other than that signal goal attainment. We refer to such tasks as near-sensorless tasks. The main thrust of this section is given by two examples that investigate the execution-time complexity of strategies for solving near-sensorless tasks both with and without randomization.

We know, of course, from [43] that planning finite-horizon strategies for discrete tasks in the absence of sensing is NP-complete. More generally, we know that many other robot planning problems in the presence of uncertainty are at least PSPACE-hard [42, 43, 39, 13, 11]. The proofs of some of these hardness results involve construction of environments in which the robot's uncertainty encodes the configuration of a Turing machine. Any guaranteed strategy for attaining the goal effectively simulates the computation of that Turing machine. Thus the hope for speedup from exponential time to polynomial time, either of planning or execution, by moving to randomized algorithms, seems futile in general [24]. Nonetheless, there is some hope that some problems experience speedups as a result of randomization.

9.1. Planning and Execution. Suppose that a system is initially in knowledge state K and suppose that at execution time a sequence of actions \{A_1, \ldots, A_i\} is executed yielding a sequence of sensory interpretation sets \{I_1, \ldots, I_i\}. The final knowledge state resulting from this particular execution trace is given by K; A_1; I_1; \ldots; A_i; I_i. In general, a plan might specify a decision tree, so that the actions executed are themselves functions of the observed sensory information. In the sensorless case, the sensing at each stage provides no additional information, we can write the execution trace as K; A_1; \ldots; A_i; I_i.

In the near-sensorless case each of the sensory interpretation sets I_i is either the whole nongoal space \mathcal{G} = \mathcal{G} \setminus I_i or the goal set I_i. If we assume that an execution trace stops once the goal is attained, then each successful execution trace is of the form K; A_1; \ldots; A_i; I_i. \mathcal{G} \setminus I_i. In this case, the actions are indeed functions of the sensory information. In particular, the number of actions executed depends on when the goal is entered, an event that is only determined in a nondeterministic fashion at execution time. However, as in the sensorless case, for a guaranteed strategy there is a unique sequence of actions that will be executed while the system is not in the goal. Said differently, the decision tree is not really a general tree, but rather a linear sequence with one-step branches at each step corresponding to early goal attainment. See Figure 9.

It follows that in both the sensorless and near-sensorless cases planning entails determining a linear path from K to \mathcal{G} in a directed graph. The states of this graph are the system's knowledge states. The edges correspond to forward projections, with one modification. In the near-sensorless case, whenever a forward projection contains both goal and nongoal states, then the corresponding arc points only to the set of nongoal states.

9.2. Relationship of Sensorless and Near-Sensorless Tasks. This section establishes a relationship between sensorless and near-sensorless tasks, that permits us
The additional action $A_G$ is designed to move any goal state in the old system into $s_0$, and nondeterministically move any nongoal state to any one of the states in $\mathcal{S}$. In other words, if the states of the original system are given by $\mathcal{S} = \{s_1, \ldots, s_n\}$, with goal states $\mathcal{G} = \{s_i\}$, then $A_G$ is specified by

$$A_G: \begin{align*}
s_1 &\rightarrow s_0, \\
\vdots \\
s_i &\rightarrow s_0, \\
s_{i+1} &\rightarrow s_1, \ldots, s_n, \\
\vdots \\
s_n &\rightarrow s_0, \\
s_0 &\rightarrow s_0.
\end{align*}$$

Finally, we define a new sensing function $\Xi$ that gives goal sensing in the new system. In other words, $\Xi(s) = \{\mathcal{S}\}$ for every $s \in \mathcal{S}$, and $\Xi(s_0) = \{\{s_0\}\}$.

It is easy to verify that there is a strategy for recognizably achieving the goal in the sensorless system if, and only if, there is a strategy for recognizably achieving the goal in the near-sensorless system. Thus the existence and structure of a guaranteed strategy for accomplishing a sensorless task is not fundamentally affected by the addition of a goal-sensor; we can always modify the problem slightly so that the goal-sensor does not provide any information useful to the guaranteed strategy.

We can also establish a correspondence in the other direction, that is, we can convert any near-sensorless problem into a sensorless one with minor modifications, while preserving the existence and essentially the structure of guaranteed strategies. The basic idea is to replace the goal-sensor with a mechanical trap that precludes ever leaving the goal once it has been attained. So, suppose we are given a discrete planning problem $(\mathcal{S}, \mathcal{A}, \Xi, \mathcal{G})$ in which the state space is $\mathcal{S} = \{s_1, \ldots, s_n\}$ and the goal states are $\mathcal{G} = \{s_i\}$. The sensor can recognize goal attainment, but otherwise provides no information. Thus $\Xi(s) = \{\mathcal{S} - \mathcal{G}\}$ for all nongoal states $s$ and $\Xi(s_0) = \mathcal{G}$ for all goal states $s_0$. Now, consider a modified problem $(\mathcal{S}, \mathcal{A}', \Xi, \mathcal{G})$, which has the same state space and goal set as the previous problem, but modified actions and a modified sensing function. The new sensing function $\Xi'$ provides no information whatsoever, i.e., $\Xi(s) = \{s\}$ for all states. The new actions are identical to the old, except that transitions out of goal states have been changed to self-transitions.

In terms of finding guaranteed strategies, we see that sensorless and near-sensorless problems are very similar. Adding a goal sensor to a sensorless problem does not change the structure of the problem much, if the applicability of the sensor depends on first executing a proper action. Conversely, for a near-sensorless problem, removing the sensor does not change the problem substantially, if the sensor can be replaced by a physical trap.
9.3. Probabilistic Speedup Example. Let us turn to the first example. See Section 9.5 below, for a physical device that has important commonalities with the following example. The purpose of this example is to prove the following claim.

**Claim 9.1.** There exists a near-sensorless discrete planning problem for which the shortest guaranteed strategy has exponential length, but for which there exists a randomized strategy that only requires quadratic expected time to attain the goal. (These complexities are measured in terms of the number of states and actions.)

We will construct a nondeterministic discrete task, consisting of $n$ states and $n$ actions. There will be one goal state, and no sensing. We will exhibit a guaranteed solution for attaining the goal from an initial knowledge state of complete uncertainty. The solution requires $2^n - n - 1$ steps, and is the shortest possible solution guaranteed to attain the goal. However, if the starting state is known exactly, there will be solutions of linear length. This suggests a guessing strategy that repeatedly guesses the current state of the system. After each guess, the system executes the appropriate linear-length plan for getting from that state to the goal. If the guess is correct, the system will wind up in the goal. Otherwise, it can guess again. This strategy attains the goal in quadratic expected time. Of course, one must add a goal-sensor in order to recognize goal attainment. Doing so does not change the fundamental character of the problem, by the relationship established in Section 9.2.

Let the states be $S = \{s_1, \ldots, s_n\}$, with the goal being state $s_1$. The task is to attain the goal from a knowledge state of complete uncertainty. For convenience we will sometimes refer to states by their indices, and specify knowledge states as subsets of the integers. Thus $K = \{1, 2, 7\}$ means that the system is in one of the states $s_1, s_2, s_7$, or $s_1, s_2, s_7$ in the usual notation.

The actions will have the following effect. We want to force the system to traverse almost all knowledge states, beginning from $\{1, 2, \ldots, n\}$, before arriving at the goal $\{1\}$. Specifically, the system will be forced to traverse all knowledge states of size $n - 1$, then all knowledge states of size $n - 2$, and so forth, through all knowledge states of size 2, until finally arriving at the goal $\{1\}$. Furthermore, within a collection of knowledge states of a given size, the system will be forced to traverse the knowledge states in lexicographic order. The lexicographic order of a knowledge state $K = \{s_{i_1}, s_{i_2}, \ldots, s_{i_k}\}$ (also written as $K = \{i_1, i_2, \ldots, i_k\}$) containing $k$ elements is determined by the string $s_{i_1} s_{i_2} \cdots s_{i_k}$ of length $k$, where the $s_{i_j}$ are assumed to be ordered in such a way that $i_1 < i_2 < \cdots < i_k$. As an example, the knowledge state $\{2, 1, 7, 12\}$ precedes the knowledge state $\{3, 6, 1, 7\}$ since $s_2 s_1 s_7 < s_3 s_6 s_1$ lexicographically. Observe that the first state of length $k$ in this ordering is the knowledge state $K_{\text{min}}^k = \{1, 2, \ldots, k\}$, whereas the last state is $K_{\text{max}}^k = \{n - k + 1, n - k + 2, \ldots, n\}$. We will refer to the collection of knowledge states of size $k$ as the $k$th level.

For the sake of example, consider the case $n = 4$. The relevant knowledge states and the order in which the system will be forced to traverse them is given by the following sequence, arranged by level. Within each level the knowledge states are listed in lexicographic order from left to right.

Level 4: \[\{1, 2, 3, 4\}\]
Level 3: \[\{1, 2, 3\} \rightarrow \{1, 2, 4\} \rightarrow \{1, 3, 4\} \rightarrow \{2, 3, 4\}\]
Level 2: \[\{1, 2\} \rightarrow \{1, 3\} \rightarrow \{1, 4\} \rightarrow \{2, 3\} \rightarrow \{2, 4\} \rightarrow \{3, 4\}\]
Level 1: \[\{1\}\]

The first action $A_0$ that we will specify is designed to permit motions between levels, specifically from the last state in each level to the first state in the next lower level, i.e., from $K_{\text{max}}^k$ to $K_{\text{max}}^{k-1}$ for all $k = n, \ldots, 2$. In addition, $A_0$ should not be useful for any other motions, that is, the action should not be capable of moving the system ahead more than one knowledge state in the order that we just specified. This means that the only other motions possible should move either to a higher level or to a previous state in the same level. The action is given as

\[A_0: \quad 1 \rightarrow 1, 2, \ldots, n - 2, n - 1,\]
\[2 \rightarrow 1, 2, \ldots, n - 2,\]
\[\vdots\]
\[k \rightarrow 1, 2, \ldots, n - k,\]
\[\vdots\]
\[n - 2 \rightarrow 1, 2,\]
\[n - 1 \rightarrow 1,\]
\[n \rightarrow 1.\]

Since there is no sensing, we will write $A(K)$ to mean $F_A(K)$ for any action $A$ and any knowledge state $K$. Observe then that indeed $A_0(K_{\text{max}}^k) = K_{\text{max}}^{k-1}$. Now consider an arbitrary knowledge state with $k$ elements, say $K = \{i_1, i_2, \ldots, i_k\}$, with $i_1 < i_2 < \cdots < i_k$. Then $A_0(K) = A_0(\{i_1\}) = \{1, 2, \ldots, n - i_1\}$, with $A_0(\{i_k\}) = \{1\}$. If we suppose that $K$ is not $K_{\text{max}}^k$, then it must be the case that $i_k < n - k + 1$. This in turn implies that $A_0(K) \supseteq A_0(\{n - k\}) = \{1, 2, \ldots, k\}$, $K_{\text{max}}^k$. In other words, either $A_0(K)$ contains $k$ elements and is equal to the least such set, or $A_0(K)$ contains more than $k$ elements. In either event, $A_0(K)$ appears before $K$ in the sequence of knowledge states that we are forcing the system to traverse. Thus $A_0$ cannot be used to any advantage in jumping ahead in that sequence.

For the case $n = 4$, $A_0$ is given by

\[A_0: \quad 1 \rightarrow 1, 2, 3,\]
\[2 \rightarrow 1, 2,\]
\[3 \rightarrow 1,\]
\[4 \rightarrow 1.\]
which maps between levels as follows:

Level 4:  
\[ \{1, 2, 3, 4\} \]

\[ A_4 \downarrow \]

Level 3:  
\[ \{1, 2, 3\} \xrightarrow{(A)} \cdots \xrightarrow{(A)} \{2, 3, 4\} \]

\[ A_3 \downarrow \]

Level 2:  
\[ \{1, 2\} \xrightarrow{(A)} \cdots \xrightarrow{(A)} \{3, 4\} \]

\[ A_2 \downarrow \]

Level 1:  
\[ \{1\} \]

Here \((A)\) refers to any action other than \(A_0\).

Now we must define the remaining \(n - 1\) actions. The purpose of each of these will be to permit the system to advance between consecutive knowledge states in the lexicographic ordering, while preventing the system from using the actions to advance more than one step in the ordering. In order to understand the definition of these actions, we will look at how to form the successor of a given knowledge state within a specific level, relative to the lexicographic ordering. Again, let us introduce some auxiliary notation. First, whenever we write a knowledge state as a set, we will write its elements in order, so that the representation of the state corresponds to its lexicographic label. In other words, a knowledge state \(K = \{s_1, s_2, \ldots, s_n\}\) will be depicted in the form \(K = \{i_1, i_2, \ldots, i_n\}\), with \(i_1 < i_2 < \cdots < i_n\). Second, if we are only interested in the last elements of the knowledge state relative to this ordering, then we will write it as \(\{s_n, i_{n-1} + 1, i_{n-2} + 2, \ldots, i_1\}\). In other words, the prefix "\(\cdot\)" will mean zero or more elements whose lexicographic value is less than that of the elements that follow. If this symbol appears more than once in an equation, then it is assumed to be bound to the same value throughout the equation. And third, we will let \(\text{Stucc}\) denote the successor function relative to the lexicographic ordering and the level in which a knowledge state is located.

Now consider the successor to a knowledge state \(K\). \(K\) is necessarily of the form \(\{s_n, i_n\}\) for some \(i_n\). If \(i_n \neq n\), then \(\text{Stucc}(K) = \{s_n, i_n + 1\}\). On the other hand, if \(i_n = n\), then we must consider the next to last entry, i.e., we must look at \(i_{n-1}\) in the representation \(K = \{s_n, i_{n-1}, \ldots, i_1\}\). Again, if \(i_{n-1} \neq n - 1\), then \(\text{Stucc}(K) = \{s_n, i_{n-1} + 1, i_{n-2} + 2\}\). Notice that in this case the successor function changes not only the next to last entry, but perhaps also the last entry. In particular, the last entry is set to be exactly one more than the next to last entry. This follows from the definition of a lexicographic order (without duplicates). Once again, if \(i_{n-1} = n - 1\), then we must look at the second to last entry \(i_{n-2}\), and so forth. In general, if we are required to look at the last \(l\) entries, then \(K\) must be of the form \(\{s_n, i, n - l + 2, n - l + 3, \ldots, n\}\), for some \(i\) with \(1 \leq i \leq n - l\). Thus \(\text{Stucc}(K)\) is of the form \(\{s_n, i + l, i + 2, i + 3, \ldots, i + l\}\). The only exception to these rules is if \(K = K_{\text{max}}\) for some \(K\). However, in that case, we are not interested in \(\text{Stucc}(K)\) anyway, as action \(A_0\) applies.

We will now define actions \(A_1, \ldots, A_{n-1}\), where the purpose of action \(A_i\) is to change \(K\) to \(\text{Stucc}(K)\) for all knowledge states of the form \(K = \{s_n, i, n - l + 2, n - l + 3, \ldots, n\}\), for some \(i\). In other words, if the relevant entry in determining the successor of \(K\) has value \(i\), then \(A_i\) will be the action that permits the system to make progress towards the goal. Furthermore, none of the other actions will permit progress at \(K\).

From the previous discussion we see that \(A_i\) must be of the form

\[
A_i: \\
1 \rightarrow 1, \\
2 \rightarrow 2, \\
\vdots \\
i - 1 \rightarrow i - 1, \\
i \rightarrow i + 1, \\
i + 1 \rightarrow i + 1, 2, \ldots, n, \\
i + 2 \rightarrow i + 2, i + 3, \ldots, n, \\
\vdots \\
i + j \rightarrow i + 2, i + 3, \ldots, n + 2 - j, \\
\vdots \\
n - 1 \rightarrow i + 2, i + 3, \\
n \rightarrow i + 2.
\]

Notice that \(A_i\) leaves all states in the range \([1, i - 1]\) unchanged. This corresponds to the "\(\cdot\)" entries in the representation \(K = \{s_n, i, n - l + 2, n - l + 3, \ldots, n\}\). Also, \(A_i\) advances \(i\) to \(i + 1\), which is the first entry changed by the successor function. State \(i + 1\) is nondeterministically sent to all possible states. This is done to preclude use of \(A_i\) when the relevant entry determining the successor of \(K\) actually has value \(i + 1\). The remaining states \(i + 2, \ldots, n\) are each sent nondeterministically to a subset of themselves. These sets form a tower collapsing to \(i + 2\), that ensures proper computation of the successor function.

We will now prove that these actions do indeed define a task for which there exists a guaranteed solution whose length necessarily is of exponential size. Then we will instantiate the actions and the strategy for the case \(n = 4\).

**Claim 9.2.** For the actions and task defined above, there exists a guaranteed strategy that traverses essentially all knowledge states, in the order described above. Furthermore, there is no shorter guaranteed strategy.

**Proof.** First, let us show that for every knowledge state containing two or more states, there is some action that makes progress towards the goal. Once we establish this, the existence of a guaranteed solution of the type described is established. Recall that progress means either moving to a successor state, or
moving down to the next level, where each level consists of knowledge states of a
given size.

Let \( K = \{ i_1, \ldots, i_k \} \), with \( i_1 < \cdots < i_k \), be given. As we already indicated, if \( K = K_{\text{max}} = \{ n - k + 1, n - k + 2, \ldots, n \} \), then \( A_2 \) will make progress at \( K \). Otherwise, let \( l \) be the index for which \( K \) is of the form \( K = \{ i_1, \ldots, i_{l-1}, i, n - l + 2, n - l + 3, \ldots, n \} \), with \( i_{l-1} = i + 2 \). Then, if \( i_{l-1} < n - l + 2 \), action \( A_l \) will make progress at \( K \), by construction. This follows from the following calculation (which makes use of the fact that \( A_l(i_j) = i_j \) for \( 1 \leq j \leq l \) and the fact that \( A_l(n - l + j) = i + 2, i + 3, \ldots, i + l - j + 2 \) for \( 2 \leq j \leq l \)).

\[
A_l(K) = \bigcup_{j=1}^{l} A_l(i_j)
= \left( \bigcup_{j=1}^{l} A_l(i_j) \right) \cup A_l(i_l) \cup \left( \bigcup_{j=2}^{l} A_l(n - l + j) \right)
= \{ i_1, \ldots, i_{l-1} \} \cup \{ i + l \} \cup \{ i + 2, i + 3, \ldots, i + l \}
= \text{succ}(K).
\]

Now let us proceed in the other direction, and show that applying the wrong
action \( A_i \), to a knowledge state cannot cause the system to advance in the ordering
outlined earlier. This will establish uniqueness of the solution, in the sense that
there is no shorter guaranteed strategy.

Let a knowledge state \( K \) be given, and consider applying action \( A_i \). We have
already shown that \( A_i \) cannot make progress unless \( K = K_{\text{max}} \), for some \( k \), so
assume that \( i > 0 \). Observe that if \( i + 1 \notin K \), then \( A_i(K) = \{ 1, 2, \ldots, n \} \), i.e., \( A_i \)
maps \( K \) to complete uncertainty. This is definitely not progress, so we may as well
assume that \( i + 1 \notin K \). Now suppose that, in fact, \( K \subseteq \{ 1, 2, \ldots, i - 1 \} \). Then
\( A_i(K) = K \), which again means there is no progress. Similarly, if \( K \subseteq \{ 1, 2, \ldots, i \} \),
then \( A_i(K) = \{ 1, 2, \ldots, i - 1, i + 1 \} \), which is progress, but now \( K \) is of the form
for which \( A_i \) was designed in the first place. So, we may assume that \( K \) intersects
the set \( \{ i + 2, \ldots, n \} \). Let \( i \) be the minimal element in \( K \cap \{ i + 2, \ldots, n \} \). Then
\( A_i(K) \supsetneq A_i(l) \). Now write \( K \) as

\[
K = (K \cap \{ i + 1, \ldots, i - 1 \}) \cup (K \cap \{ i + 2, \ldots, n \}) \cup (K \cap \{ i \}).
\]

Given the minimality of \( i \), this says that \( |K| = |K \cap \{ i + 1, \ldots, i - 1 \}| +
|K \cap \{ i + 2, \ldots, n \}| + x_K(0) \), where \( x_K \) is the characteristic function of \( K \). Applying
action \( A_{i+1} \), we see that

\[
A_{i+1}(K) = (K \cap \{ i + 1, \ldots, i - 1 \}) \cup \{ i + 2, i + 3, \ldots, i + n - l - 2 \} \cup A_l(K \cap \{ i \}).
\]

where \( A_l(i) = \{ i + 1 \} \). Thus \( |A_{i+1}(K)| = |K \cap \{ i + 1, \ldots, i - 1 \}| + (n - l + 1) + x_K(0) \). If
\( |A_{i+1}(K)| > |K| \), then \( A_i \) is moving \( K \) back up one or more levels, hence not making
progress, so consider the possibility that \( |A_{i+1}(K)| \leq |K| \). This is possible if, and only
if, \( n - l + 1 \leq |K \cap \{ i, \ldots, n \}| \). Clearly, this inequality can at best be an equality,
in which case we must have that \( K = \{ i, \ldots, n \} \). Now there are two possibilities:
either \( i \notin K \) or not. In the first case, we have that \( K \) is of the form \( \{ \alpha, i, l + 1, \ldots, n \} \),
with \( i \leq l + 1 \). In this case \( A_i \) is designed to make progress at \( K \).
Thus, finally, assume that \( i \notin K \). So, \( K = \{ \alpha, i, l + 1, \ldots, n \} \) and \( A_2(K) = \{ \alpha, i, l + 2, \ldots, i + n - l + 2 \} \), with \( i + 2 \leq l \). But this says that either \( A_2(K) \)
is equal to \( K \) or \( A_2(K) \) precedes \( K \) lexicographicaly. In short, \( A_i \) does not make
progress at \( K \).

Let us instantiate these actions for the case \( n = 4 \). We have

\[
A_1: \quad 1 \rightarrow 2, \quad A_2: \quad 1 \rightarrow 1, \quad A_3: \quad 1 \rightarrow 1.
\]

\[
2 \rightarrow 1, 2, 3, 4, \quad 2 \rightarrow 3,
3 \rightarrow 3, 4, \quad 3 \rightarrow 1, 2, 3, 4, \quad 3 \rightarrow 4,
4 \rightarrow 3, \quad 4 \rightarrow 4, \quad 4 \rightarrow 1, 2, 3, 4.
\]

The guaranteed solution is given by

\[
\{ 1, 2, 3, 4 \}
\]

We see then that there are tasks for which the planning and execution times are
exponential in the size of the input. Observe, however, for this particular example,
that if the initial state of the system were known precisely, then there would be a
fast solution for attaining the goal. In particular, if the initial state is \( s_1 \), then
the system is already in the goal. If the initial state is \( s_2 \), then action \( A_0 \)
will attain the goal in a single motion. Finally, if the initial state is \( s_3 \), then action
\( A_2 \) will cause a transition to state \( s_4 \), from which \( A_0 \) will attain the goal. In short,
if we write out the dynamic programming table to two columns for this task, then
we have a collection \( \{ K_i \} \) of knowledge states that cover the entire state space.
Thus we can employ a randomized strategy that guesses the initial state of the system, then executes a short sequence of actions designed to attain the goal. We must, of course, add a goal sensor, in order to ensure reliable goal recognition.

For the sake of completeness, note that the relevant portion of the backtracking diagram corresponding to the dynamic programming table out to two columns is given by the following diagram (depicting vertical levels rather than horizontal columns):

```
(2)                      (3, 4)                      (4)
|                        |                           |
A_3                       A_4                          A_5
|                        |                           |
(3)                      (4)                          (1)
|                        |                           |
A_2                       A_1                          A_0
```

In the general case, we must backchain out to the \((n - 2)\)nd column of the dynamic programming table. A guessing strategy consists of guessing between the \(n - 1\) nongoal states, then executing a strategy of no more than \(n - 2\) steps, that is guaranteed to attain the goal if the guess is correct. Thus the expected number of actions executed until the goal is attained is on the order of \(n^2\).

Notice that adding a goal sensor does not fundamentally change the exponential character of the guaranteed solution, by the relationship of sensorless and near-sensorless tasks, as established in Section 9.2. It is important to keep this relationship in mind, since a goal sensor clearly permits a speedup of the guaranteed solution if we do not make the modifications suggested by Section 9.2. We now have a proof of Claim 9.1.

**Proof of Claim 9.1.** Most of this claim has been proved. We only need to verify that there does indeed exist a linear time strategy for attaining the goal if the initial state of the system is known. We return to the construction above.

First notice that action \(A_3\) is guaranteed to move states \(s_5\) and \(s_{n-3}\) into the goal in a single motion. Observe also that action \(A_4\) is guaranteed to move state \(s_i\) to state \(s_{i+1}\) for all positive \(i\). This establishes the claim.

In retrospect, the randomizing part of the claim is not very surprising. The actions \(A_i\) are actually fairly deterministic. However, the solutions for different initial states are not at all commensurate. Said differently, the solution for a given initial state is not guaranteed to serendipitously make progress at other states. This is quite unlike the fortunate situation that we encounter with one-dimensional random walks, where the same solution pretty much applies to all possible states. Thus the surprising aspect of the claim is the exponential character of the guaranteed solution for what may seem to be fairly deterministic actions.

9.4. An Exponential-Time Randomizing Example. The following example exhibits a (near-) sensorless task for which the shortest guaranteed solution requires an exponential number of steps, and for which a randomized solution that guesses the starting state also requires exponential time. The basic idea is to generate a problem in which the knowledge states play the role of bit vectors, that may be modified only by counting. The main purpose of this example is to establish the following claim.

**Claim 9.3.** There exists a near-sensorless discrete planning problem for which the shortest guaranteed strategy has exponential length. Furthermore, the worst-case expected running time of any randomized strategy for this problem is also exponential in the number of states and actions.

The example will consist of \(n\) states, and \(2n - 3\) actions. We will present the example as if there is no sensing, bearing in mind the relationship between sensorless and near-sensorless problems described in Section 9.2. We retain some of the notation from the previous example (Section 9.3). In particular, we will interchangeably refer to a state either as \(s_i\) or as \(i\), for \(i = 1, \ldots, n\).

The state space will be of the form \(\mathcal{S} = \{s_1, \ldots, s_n\} = \{1, \ldots, n\}\), with the goal being state \(s_1\). Again, the task is to attain the goal from complete uncertainty. We will denote the actions by the symbols \(A_1, \ldots, A_{n-1}\) and \(B_1, \ldots, B_{n-2}\). We will write knowledge states as ordered tuples, as we did in the previous section. In other words, a knowledge state \(K\) of size \(k\) will be written in the form \(K = \{s_{i_1}, \ldots, s_{i_k}\} = \{i_1, \ldots, i_k\}\), with \(i_1 < \cdots < i_k\). Thinking of a knowledge state as a bit vector, \(K\) will correspond to the number \(x(K)\), with

\[
x(K) = \sum_{i \in K} 2^{x(i)}.
\]

Conversely, given an integer \(x\) in the range \([0, 2^n - 1]\), there is a unique knowledge state \(K\) for which \(x(K) = x\). We will denote this knowledge state by \(K(x)\), with

\[
K(x) = \{i | \text{bit } i \neq (n - i) \text{ is a } 1 \text{ in the binary representation of } x\}.
\]

As an example, if \(n = 10\) and \(K = \{1, 3, 7\}\), then \(x(K) = 648\). Similarly, if \(n = 4\) and \(x = 9\), then \(K(x) = \{1, 4\}\).

As before, we will let the prefix symbol "\(\alpha\)" in the representation \(K = \{s_{i_1}, i_1, \ldots, i_k\}\) denote zero or more elements whose lexicographic order precedes that of \(i_k\). This notation carries over to the binary representation of the number \(x(K)\). Comparing the binary representation of \(x(K)\) with \(K\), we have the following schematic:

```
\[\begin{array}{cccccccc}
\text{Bit }* & \cdots & n-i_1 & \cdots & n-i_2 & \cdots & n-i_k & 0 \\
\text{x(K):} & 0 & \cdots & 0 & 1 & 0 & \cdots & 1 & 0 & \cdots & 0 \\
\text{K:} & \{\alpha\} & s_{i_1} & s_{i_2} & \cdots & s_{i_k}\end{array}\]
```

9.4. An Exponential-Time Randomizing Example. The following example exhibits a (near-) sensorless task for which the shortest guaranteed solution requires an exponential number of steps, and for which a randomized solution that guesses the starting state also requires exponential time. The basic idea is to generate a problem in which the knowledge states play the role of bit vectors, that may be modified only by counting. The main purpose of this example is to establish the following claim.

**Claim 9.3.** There exists a near-sensorless discrete planning problem for which the shortest guaranteed strategy has exponential length. Furthermore, the worst-case expected running time of any randomized strategy for this problem is also exponential in the number of states and actions.

The example will consist of \(n\) states, and \(2n - 3\) actions. We will present the example as if there is no sensing, bearing in mind the relationship between sensorless and near-sensorless problems described in Section 9.2. We retain some of the notation from the previous example (Section 9.3). In particular, we will interchangeably refer to a state either as \(s_i\) or as \(i\), for \(i = 1, \ldots, n\).

The state space will be of the form \(\mathcal{S} = \{s_1, \ldots, s_n\} = \{1, \ldots, n\}\), with the goal being state \(s_1\). Again, the task is to attain the goal from complete uncertainty. We will denote the actions by the symbols \(A_1, \ldots, A_{n-1}\) and \(B_1, \ldots, B_{n-2}\). We will write knowledge states as ordered tuples, as we did in the previous section. In other words, a knowledge state \(K\) of size \(k\) will be written in the form \(K = \{s_{i_1}, \ldots, s_{i_k}\} = \{i_1, \ldots, i_k\}\), with \(i_1 < \cdots < i_k\). Thinking of a knowledge state as a bit vector, \(K\) will correspond to the number \(x(K)\), with

\[
x(K) = \sum_{i \in K} 2^{x(i)}.
\]

Conversely, given an integer \(x\) in the range \([0, 2^n - 1]\), there is a unique knowledge state \(K\) for which \(x(K) = x\). We will denote this knowledge state by \(K(x)\), with

\[
K(x) = \{i | \text{bit } i \neq (n - i) \text{ is a } 1 \text{ in the binary representation of } x\}.
\]

As an example, if \(n = 10\) and \(K = \{1, 3, 7\}\), then \(x(K) = 648\). Similarly, if \(n = 4\) and \(x = 9\), then \(K(x) = \{1, 4\}\).

As before, we will let the prefix symbol "\(\alpha\)" in the representation \(K = \{s_{i_1}, i_1, \ldots, i_k\}\) denote zero or more elements whose lexicographic order precedes that of \(i_k\). This notation carries over to the binary representation of the number \(x(K)\). Comparing the binary representation of \(x(K)\) with \(K\), we have the following schematic:

```
\[\begin{array}{cccccccc}
\text{Bit }* & \cdots & n-i_1 & \cdots & n-i_2 & \cdots & n-i_k & 0 \\
\text{x(K):} & 0 & \cdots & 0 & 1 & 0 & \cdots & 1 & 0 & \cdots & 0 \\
\text{K:} & \{\alpha\} & s_{i_1} & s_{i_2} & \cdots & s_{i_k}\end{array}\]
```
The actions that we will construct will force the system to traverse an exponential number of knowledge states, beginning with the state of complete uncertainty \( \{1, \ldots, n\} \), and ending with the goal state \( \{1\} \), in an order that corresponds to counting downward from \( 2^n - 1 \) to \( 2^{n-1} \). For the special case \( n = 4 \), this corresponds to the following transitions (for later reference the transitions are also labeled with the associated actions):

<table>
<thead>
<tr>
<th>( K )</th>
<th>( x(K) )</th>
<th>(actions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1, 2, 3, 4}</td>
<td>15</td>
<td>( A_3 )</td>
</tr>
<tr>
<td>{1, 2, 3}</td>
<td>14</td>
<td>( B_2 )</td>
</tr>
<tr>
<td>{1, 2, 4}</td>
<td>13</td>
<td>( A_3 )</td>
</tr>
<tr>
<td>{1, 2}</td>
<td>12</td>
<td>( B_1 )</td>
</tr>
<tr>
<td>{1, 3, 4}</td>
<td>11</td>
<td>( A_3 )</td>
</tr>
<tr>
<td>{1, 3}</td>
<td>10</td>
<td>( A_3 )</td>
</tr>
<tr>
<td>{1, 4}</td>
<td>9</td>
<td>( B_1 )</td>
</tr>
<tr>
<td>{1}</td>
<td>8</td>
<td>( A_3 )</td>
</tr>
</tbody>
</table>

Let us first define the actions \( \{A_k\} \). These are designed to count down from knowledge states \( K \) whose associated numbers \( x(K) \) are odd. Since an odd number contains a one in the least significant bit, the knowledge state must contain the state \( s_k \). The actions \( \{A_k\} \) are designed to remove this state. We have, for \( k = 1, \ldots, n - 1 \),

\[
A_k: \quad 1 \rightarrow 1, \\
2 \rightarrow 2, \\
\vdots \\
k \rightarrow k, \\
k + 1 \rightarrow 1, 2, \ldots, n, \\
\vdots \\
n - 1 \rightarrow 1, 2, \ldots, n, \\
n \rightarrow k.
\]

[Note, of course, that if \( k = n - 1 \), then both \( A_k(n - 1) \) and \( A_k(n) \) are just \( n - 1 \).]

Similarly, the actions \( \{B_k\} \) are designed to count down by one from knowledge states whose associated numbers are even. Thus these actions must worry about borrowing properly from higher order bits. We have, for \( k = 1, \ldots, n - 2 \),

\[
B_k: \quad 1 \rightarrow 1, \\
2 \rightarrow 2, \\
\vdots \\
k \rightarrow k, \\
k + 1 \rightarrow k + 2, k + 3, \ldots, n, \\
k + 2 \rightarrow 1, 2, \ldots, n, \\
\vdots \\
n - 1 \rightarrow 1, 2, \ldots, n.
\]

For the special case \( n = 4 \), we have the following five actions:

\[
A_3: \quad 1 \rightarrow 1, \\
2 \rightarrow 1, 2, 3, 4, \\
3 \rightarrow 1, 2, 3, 4, \\
4 \rightarrow 1, \\
B_1: \quad 1 \rightarrow 1, \\
2 \rightarrow 3, 4, \\
3 \rightarrow 1, 2, 3, 4, \\
4 \rightarrow 1, 2, 3, 4
\]

\[
A_2: \quad 1 \rightarrow 1, \\
2 \rightarrow 2, \\
3 \rightarrow 1, 2, 3, 4, \\
4 \rightarrow 2, \\
B_2: \quad 1 \rightarrow 1, \\
2 \rightarrow 2, \\
3 \rightarrow 4, \\
4 \rightarrow 1, 2, 3, 4
\]

**Claim 9.4.** For the actions and task defined above, there exists a guaranteed strategy that traverses half of the possible knowledge states, in the order described above. Specifically, the strategy traverses all knowledge states that contain state \( s_1 \). There are \( 2^{n-1} \) such knowledge states. Furthermore, there is no shorter guaranteed solution.

**Proof.** First, let us show that for every knowledge state \( K \) there is some action that makes progress. In this case, progress means that the number determined by the bit-vector representation of \( K \) is decreased. In fact, we will exhibit an action that decreases \( x(K) \) by exactly one.

Suppose that \( x(K) \) is odd. Let \( k \) be the order of the least significant bit other than bit \( \# 0 \) which is set to \( 1 \). Then

\[
x(K) = \oplus 1 0 \cdots 0 1.
\]
meaning that \( K = \{ s_n, s_{n-1}, s_k \} \). Now note that \( A_{s_n}(K) = \{ s_n, s_{n-1} \} \), so \( x(A_{s_n}(K)) = x(K) - 1 \), as desired.

On the other hand, suppose that \( x(K) \) is even. Again, let \( k \) be the order of the least significant bit that is set to 1. Then \( k \geq 1 \), and

\[
x(K) = \underbrace{0 \cdots 0}_k \quad \text{if} \quad y = x(K) - 1,
\]

This says that \( K = \{ s_n, s_{n-1} \} \) and that \( K(y) = \{ s_n, s_{n-k+1}, s_{n-k+2}, \ldots, s_k \} \). Now note that \( B_{s_{n-k}}(K) = K(y) \), as desired.

We have shown that for any knowledge state \( K \) there exists a strategy for counting down from \( x(K) \). In particular, suppose \( K = \{ i_1, \ldots, i_l \} \), with \( i_1 < \cdots < i_l \). If \( i_1 = 1 \), then we can count from \( x(K) \) down to \( 2^{l-1} \), at which point the goal is attained. On the other hand, if \( i_1 > 1 \), then we can count from \( x(K) \) down to 1, which places the system in state \( s_n \). Applying action \( A_t \), then attains the goal.

Second, we must show that applying the wrong action at a knowledge state cannot make further progress. This will establish that the strategy just outlined is the strongest strategy guaranteed to attain the goal.

So, suppose that knowledge state \( K \) is given, and let \( x = x(K) \).

Consider applying action \( A_{s_n} \) for some \( k \). If \( K \cap \{ s_{n-1}, \ldots, s_{n-k} \} \neq \emptyset \), then \( A_{s_n}(K) = \{ s_1, \ldots, s_k \} \), which is certainly not progress. On the other hand, if \( K \subseteq \{ s_1, \ldots, s_k \} \), then \( A_{s_n}(K) = K \), which again is not progress. That leaves the possibility that \( K \subseteq \{ s_1, \ldots, s_k \} \cup \{ s_n \} \). Suppose that both \( s_n \) and \( s_{n-k} \) are in \( K \). Then \( A_{s_n} \) is designed to make progress at \( K \), so that's fine. On the other hand, suppose that \( s_n \notin K \) and \( s_{n-k} \notin K \). Then \( K = \{ s_n, s_{n-k} \} \), while \( A_{s_n}(K) = \{ s_n, s_{n-k} \} \). Note that \( x(A_{s_n}(K)) > x(K) \), so this motion also does not make progress.

Consider applying action \( B_{s_k} \) for some \( k \). If \( K \subseteq \{ s_1, \ldots, s_k \} \), then \( B_{s_k}(K) = K \), which means no progress. If \( K \) contains any elements from the set \( \{ s_{n-2}, \ldots, s_k \} \), then \( B_{s_k}(K) \) is the entire state space, i.e., complete uncertainty. The remaining case says that \( K = \{ s_n, s_{n-k} \} \), but then \( K \) is of the form for which \( B_{s_k} \) was designed to make progress.

Observe that the previous proof also shows that if the state of the system is known exactly, say \( K = \{ s_n \} \), then the only reasonable strategy for attaining the goal is to count down to 1 from \( 2^{l-1} \), followed by an application of action \( A_{s_n} \). This is because applying the wrong action at a knowledge state essentially has one of two effects: Either (1) the action does not change the knowledge state, or (2) the action yields complete uncertainty. The exception to this rule is given by the effect on state \( s_n \), but this state lies one action away from the goal, and misapplying an action when the system is in state \( s_n \) only moves it further away.

This establishes Claim 9.3.

9.5. The Odometer. The following physical device has important commonalities with the graph example presented in Section 9.3. In particular, the task associated with this device has a guaranteed solution that requires up to an exponential number of steps, and a randomized solution that only requires an expected linear number of steps.

Imagine a series of a horizontal plates or wheels arranged vertically above each other. The plates are connected by a gearlike mechanism that acts much like an odometer. Specifically, a primitive action consists of turning a plate one-tenth of a revolution. Call this a partial turn. Whenever a given plate turns, it also turns the plate above it, but at one-tenth the speed, so each time a plate makes one full revolution, the plate immediately above makes a partial turn. Similarly, turning a plate turns the plate directly below it at ten times the speed. There is a crank below the bottom plate which turns that plate, and consequently all other plates at reduced speeds. Under certain circumstances mentioned later, individual plates may also be turned directly. The crank and any individual plate can only be turned at a specific fixed speed, say, one partial turn per unit time. (Turning an individual plate directly also turns the other plates via the gearing mechanism, as described earlier.)

On one of the plates is a ball. The ball arrives from a distribution bin which nondeterministically places the ball on a nondeterministically chosen plate. There is a chute next to each plate. Turning the plate so that the ball passes by this chute causes the ball to roll off the plate, down the chute, and onto the plate below. The chutes are themselves arranged in unison above each other. They are hinged to a vertical pole, and may be swung away from the plates. In this case, if a plate is turned so that the ball passes by the location at which the chute would normally be, the ball simply drops vertically. If the ball is not caught by someone, it reenters the distribution bin and is once again nondeterministically placed on a plate. The plates cannot be turned individually when the chutes are in place; only the crank may be used. However, the plates may be turned individually when the chutes have been swung away from the plates.

There are thus two ways to remove a ball from a plate. The first is to swing the chutes away from the plates, move one's hand up to the plate containing the ball, then turn the plate until the ball falls out and onto one's hand. The second way is to swing the chutes into place, then turn the crank until the ball emerges from the bottom plate.

The first approach requires turning the given plate at most ten partial turns, before the ball falls out. The second approach may require turning the crank as many as \( \frac{1}{10}(10^k - 1) \) partial turns, should the ball happen to be on the top plate at the start. Clearly, assuming that we can determine on which plate the ball is resting, the first approach is preferable.

Now, suppose, however, that one cannot determine on which plate the ball is resting. For instance, the plates might be covered. Then the only guaranteed strategy for removing the ball is to turn the crank with the chutes in place, until the ball emerges. Turning any individual plate, with the chutes swung away, runs the risk of causing the ball to drop from a plate, forcing it back into the distribution bin. From a worst-case point of view, that strategy might never terminate.
Consequently, the only guaranteed strategy may require a long time to execute.

Fortunately, a randomized solution consists of guessing the plate on which the ball is resting, then acting as if that plate did indeed hold the ball. In other words, in the absence of a sensor, the randomized strategy simulates one. With probability 1/n, the strategy will pick the correct plate. If it picks the wrong plate, then the ball is repositioned, and the strategy can try again. The expected number of partial turns until the ball emerges is thus bounded by 10n. This is only a linear factor more than in the case in which a sensor is available, well below the exponential guaranteed strategy.²

10. Summary. This paper considers the problem of planning strategies for tasks specified on discrete spaces in the presence of action and sensing uncertainty. The paper augmented the Poisson distribution programming approach with an operator that purposefully executes randomizing actions at run-time. The motivation for including this operator is to extend the class of solvable tasks beyond those solvable by guaranteed strategies. A guaranteed strategy is one that is certain to accomplish its task using a fixed and bounded number of run-time operations that may be ascertained at planning time. Not all tasks can be solved with guaranteed strategies. Yet there are many tasks that we regard as solvable simply because they may be accomplished frequently even if not always. By placing a loop around a strategy that tries to accomplish such a task, a system can often ensure eventual task completion. Although, in principle, the looping strategy could require an unbounded amount of time, often the expected time until task completion is finite. In particular, by purposefully randomizing its decisions a system can enforce a minimum probability of success on any given attempt, thereby placing an upper bound on the expected time until task completion.

The basic scheme is to compute partial plans that are guaranteed to accomplish portions of the task. Generally these partial plans will only succeed if fairly stringent initial conditions are satisfied. While any particular plan's preconditions may not be satisfiable, the union of all the preconditions may be satisfiable. This means that, in fact, some partial plan's preconditions are satisfied, but due to uncertainty the system cannot ascertain which plan's preconditions are satisfied. In that case, it makes sense for the system to guess the appropriate partial plan. Effectively, the system is executing a randomizing action by guessing which partial plan is applicable. If the guess is correct, then the system will accomplish its task. Otherwise, the system will need to guess again, and it eventually accomplishes the task.

The main result of this paper included an example of a task for which a guaranteed solution requires exponential planning and execution time, but for which there exists a randomized strategy that only requires polynomial planning time and quadratic expected execution time.

² Of course, the randomized strategy may require more than the expected number of trials to succeed on any particular execution. However, the probability of requiring several factors of this expectation decreases exponentially quickly in the number of factors.

Acknowledgments. Many thanks to Tomás Lozano-Pérez and Bruce Donald for many insightful conversations and suggestions during the exploration of this topic. Also many thanks to Rod Brooks, Eric Crilison, and Matt Mason for their help throughout. Many of the key ideas in this work arose through our conversations. Thanks to Bruce Donald's Robotics Class for their careful reading of the thesis on which this paper is based. Finally, thanks to the reviewers for their insightful comments and suggestions.

References

Randomization for Robot Tasks


