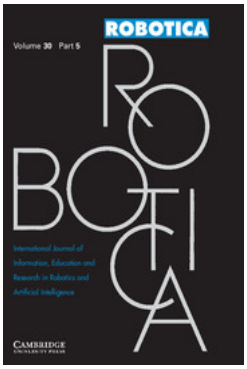

Comparing two algorithms for automatic planning by robots in stochastic environments

Alan D. Christiansen and Kenneth Y. Goldberg

Robotica / Volume 13 / Issue 06 / November 1995, pp 565 - 573
DOI: 10.1017/S0263574700018646, Published online: 09 March 2009

Robotica (1995) volume 13, pp 565–573. © 1995 Cambridge University Press

Comparing two algorithms for automatic planning by robots in stochastic environments*

Alan D. Christiansen† and Kenneth Y. Goldberg‡

(Received in final form: December 12, 1994)

SUMMARY

Planning a sequence of robot actions is especially difficult when the outcome of actions is uncertain, as is inevitable when interacting with the physical environment. In this paper we consider the case of finite state and action spaces where actions can be modeled as Markov transitions. Finding a plan that achieves a desired state with maximum probability is known to be an *NP-Complete* problem. We consider two algorithms: an exponential-time algorithm that maximizes probability, and a polynomial-time algorithm that maximizes a lower bound on the probability. As these algorithms trade off plan time for plan quality, we compare their performance on a mechanical system for orienting parts. Our results lead us to identify two properties of stochastic actions that can be used to choose between these planning algorithms for other applications.

KEYWORDS: Automatic planning; Robots; Stochastic environments; Algorithms.

1. INTRODUCTION

To plan, a robot must anticipate the outcome of its actions. If we choose to model the robot's environment as a deterministic mapping on a finite set of possible states, a robot operating in the physical environment often encounters *apparent non-determinism*: a given action applied to a given state does not always yield the same result. As the robot's model of the environment is an approximation to the real environment, such uncertainty can arise from: errors in sensing (either the initial or next state is not measured accurately), errors in control (the commanded action is not repeated accurately), or a combination of the two caused by physical effects too fine to be modeled, such as friction or collision dynamics.

* A portion of this work was performed by the authors at Carnegie Mellon University.

† Computer Science Department, Tulane University, New Orleans, LA 70118-5674 (USA).

Supported at CMU by an AT&T Bell Laboratories Ph.D. Scholarship and by the National Science Foundation under grant DMC-8520475. A portion of this work was completed during a visit to the Laboratoire d'Informatique Fondamentale et d'Intelligence Artificielle (LIFIA) in Grenoble, France, supported by INRIA.

‡ Institute for Robotics and Intelligent Systems, University of Southern California, Los Angeles, CA 90089-0273 (USA).

Supported by the National Science Foundation under Awards No. IRI-9123747, and DDM-9215362 (Strategic Manufacturing Initiative).

To generate plans that will be reliable under such conditions, we can model events *stochastically*. When an action is applied to a given state, the probability of reaching a particular next state is predicted using a discrete probability distribution. We model the state of the environment as a random variable with an associated probability distribution.

This distribution can be thought of as a *hyperstate* with "probability mass" assigned to each state of the environment.[1] Each action is a mapping from one hyperstate to another. If the current probability distribution depends only on the action and previous hyperstate, then the sequence of hyperstates is known as a *Markov chain*.

The conditional probabilities that define each action can be represented using a matrix. Depending on the application, the field of statistics offers a variety of approaches to estimating the value of such matrices.[2,3] In this paper we use a Bayesian estimator to estimate the values in these matrices based on observations of a physical robot system. We also test the resulting plans on a particular part orienting application. Though we use a specific robot application, we are motivated to identify task characteristics that are common to a wide variety of robot manipulation tasks.

Our problem can be defined as follows: Given a known initial state, desired final state and a finite set of actions represented with stochastic transition matrices, find a sequence of actions that will maximize the probability of reaching the desired final state. We consider the "unobserved" planning problem (where we do not observe the state between actions during plan execution).[4,5] The exponential-time algorithm to solve the problem (by checking all possibilities) is straightforward. It has been shown[6] that finding an optimal plan is *NP-Complete*; thus it is unlikely that a polynomial-time algorithm exists. This motivates us to consider a faster algorithm that yields suboptimal plans. Since the general approach is to trade off planning time against plan quality, this paper compares two planners on our application task.

We have found that planning is affected by two contrasting properties of stochastic actions. Some actions have the property that a single initial state tends to map onto more than one result state. We say that the action exhibits *state divergence* when, in executing the action, there is a tendency to distribute probability mass. Alternatively, some actions have the property that

multiple initial states map into a single result state. In these cases, we say that an action exhibits *state convergence*—it tends to focus distributed probability mass into a single state. An action can exhibit both divergence and convergence if it spreads out probability mass from one state while combining mass from other states. Of course, some actions may exhibit neither state divergence nor state convergence.

A stochastic planner evaluates plans by tracking how probability mass is distributed by sequences of actions. The objective is to find a plan that transfers a maximal amount of probability mass into the desired final state. This can be accomplished in several ways. One way is to use actions that cause state divergence followed by actions that cause convergence toward the desired state. Another way to plan is to *avoid* actions that cause state divergence. These two ways of transferring probability mass suggest two planning methods: Method I propagates hyperstates, using a forward exhaustive search (to a fixed plan length) to find a plan most likely to produce the desired goal. Method II propagates only the most likely state, using a backward best-first search to find a plan that maximizes a lower bound on the probability of reaching the goal.

We describe the two planning methods in detail and analyze the computational complexity of each method. We compare the plans produced by each method and the planning time required by each planner. We conclude the paper by revisiting the concepts of state divergence and state convergence and their importance to planning with stochastic actions.

2. RELATED WORK

2.1 Stochastic reasoning

The general problem of planning with stochastic actions has been considered in the AI literature beginning with the landmark paper[7] that advocated the use of decision theory with numerical cost and probability models. Today, decision theory is a standard tool in AI;[8,9] in fact, it has been argued[10] that Bayesian decision theory subsumes many alternative approaches to coping with uncertainty. One factor that is not always considered is the computation time required to find an optimal plan: often we are willing to accept a satisfactory plan now rather than an optimal plan later. Building on pioneering work by Simon,[11] Etzioni[12] incorporated a model of deliberation cost into a single Bayesian planning framework. The present paper addresses the tradeoff between plan time and plan quality by comparing the measured performance of two algorithms on the same problem.

In the field of control theory, stochastic dynamic programming has been shown[13,14] to be a powerful tool for finding an optimal control *policy* (i.e. a mapping from states to actions). During execution, the robot looks up the current state in the table and performs the action stored therein. Unfortunately, these methods do not apply to the "unobserved" planning problem that we consider in this paper, where the robot is not allowed to sense the state of the system and must proceed open-loop.

2.2 Empirical models

One can also view the systems of the current paper as *learning systems*, because the models of the task are developed by the robot from direct observations of the effects of actions. (These observations are made *off-line*, before planning occurs, in keeping with the unobserved planning problem that we consider. There are variants of our problem where sensing is allowed during plan execution, and this sensing may be used to improve the probability of achieving the goal.) The observations allow the robot to adapt its model of action effects. If these adaptations reflect physical reality, then the robot improves its ability to generate correct plans, and also its ability to achieve its goals. Learning systems for robotics and control applications have been studied for more than a decade. The peg-in-hole problem was considered[15] using a learning automata approach.[16] Dufay and Latombe[17] considered the problem of inducing robot control programs from example program traces. Barto[18] provided a survey of neural net approaches for control problems. Buckley[19] considered the problem of human-guided robot teaching in the context of robot motion planning. Gross[20] described a method for learning backprojections of desired part orientations under pushing actions. Bennett[21] presented a method for tuning stochastic plans based upon prior models of the task actions. Christiansen's PhD Thesis[22] provides a detailed review of learning systems for robotic manipulation.

As stated earlier, a planning algorithm must be able to predict the outcome of alternative actions. This prediction can either be based on analytic (*a priori*) models of actions, or on empirical observations. In this paper we consider the latter (although we do employ an a priori discretization of the state space).

Research in robotic motion planning has tended to focus on analytic models, even though such models are often underspecified when friction is present.[23–27] An analytic model for the part orienting application considered in this paper, based on a novel representation for multiple frictional contacts in the plane, was originally described by Erdmann and Mason.[28] Erdmann and Mason considered the problem of finding plans that achieve a unique final orientation for a part regardless of its initial orientation. The authors based their analytic model on two strong assumptions: that friction obeyed Coulomb's Law with a uniform constant coefficient throughout the task space, and that inertial effects could be ignored. Since this model does not uniquely specify the outcome of every action, the planning algorithm kept track of all possible outcomes. Although this planner performed remarkably well in experiments (approximately 50% success rate), the prospect of adding more elaborate analysis of friction and dynamics was daunting. In contrast, our approach is to plan with a stochastic model based solely on physical observations.

3. TRAY-TILTING

To compare the two planning algorithms, we applied them to a method for orienting parts called "tray-tilting".[28] The objective is to manipulate planar objects in a walled tray by tilting the tray so that the objects slide to a desired position and orientation. (See Figure 1.) The physics of this task are subtle and include frictional and impact effects. Tray-tilting has been studied before, both analytically[28,29] and empirically.[30–32]

Our robot arm was programmed to perform tilts in any desired direction. As the tray is tilted by the robot, gravity acts on the object and causes it to slide. For the purposes of this paper, we define tilts by a parameter which we call the tilt *azimuth*. This angle is the direction, in the plane of the tray bottom, of the force of gravity on the object during the tilt. (See Figure 2.) Other parameters of the tilt, such as steepness of the tilt and tilt speed, had constant values during all tilts. (The tilt steepness was thirty degrees from the horizontal and the tilt speed was set at a relatively slow speed: 25% of the nominal monitor speed for the Puma 560 robot that we used.) Although there is an infinite number of possible tilt azimuths, and therefore an infinite number of tilts, we sampled this space of azimuths at 30 degree intervals. In all cases, the robot restricted its actions to one of these 12 tilts.

In addition to defining tilts by a single parameter, we simplified the description of task *state* by discretizing the object's position and orientation. As indicated in Figure 2, we divided the tray (conceptually) into nine regions, corresponding to the four corners, the four sides, and the middle of the tray. We gave each of these regions symbolic names, as shown in Figure 2. We described an object's position by one of these names – the name of the region in which the object's center is located. A camera above the tray and an industrial image processor provided this position information. We described orientations by a similar discretization – our objects were described as being either horizontal (h) or vertical (v), depending on the orientation of the object's major axis, as reported by our vision system. The rectangular tile of Figure 2 would be classified as ($sw\ h$). This discretization is coarse and somewhat arbitrary. However, because the object tends to be in contact with tray walls as a result of the applied tilting actions, the states defined by our discretization correspond to qualitatively distinct physical configurations.

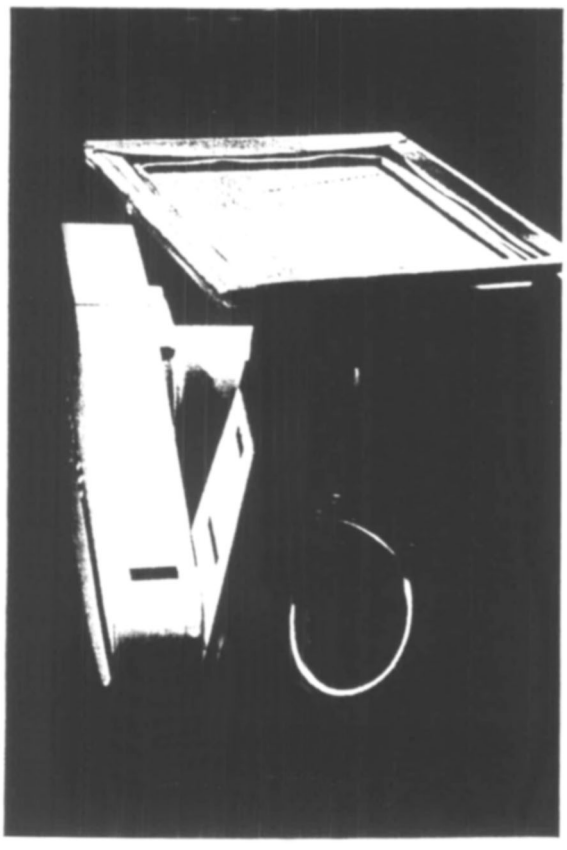

Fig. 1. The robot performing a tilting action.

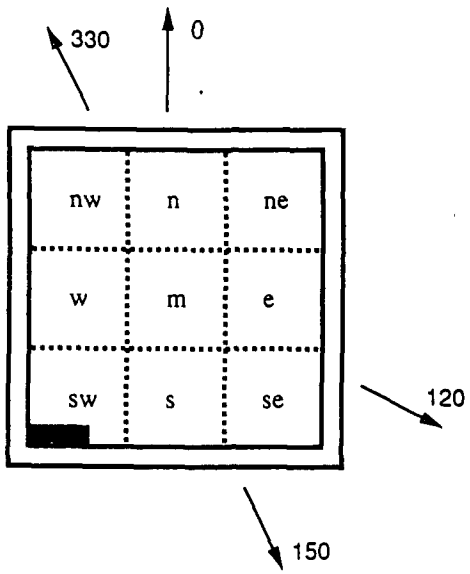

Fig. 2. The tray (viewed from above), its states, and examples of tilt azimuths.

Because of our discretization of state, errors in our vision system, inaccuracies of the robot, and unmodeled physics (such as impact), this task exhibits *apparent non-determinism*. Our robot system, given only coarse sensory capabilities and no knowledge of physics, observes that its actions are not perfectly repeatable. This lack of repeatability makes the task seem non-deterministic. However, it is possible for the robot to build a stochastic model of its environment by observing the relative frequencies with which events occur in response to its actions. The observed non-determinism may or may not be due to a non-deterministic world. Noisy sensors and effectors are sufficient to make the world seem non-deterministic to the robot.

4. DEVELOPING A STOCHASTIC MODEL FROM OBSERVATIONS

Given a set of observations from physical experiments, what is an appropriate stochastic model of actions? If the probabilities of future states depend only on the current state, then we can use a *Markov chain* to represent the

actions. We represented each tilting action u with a stochastic transition matrix, $\mathbf{P}_u$, where p_{ij} is the (conditional) probability that the system will be in state j after action u is applied to state i. We assume that the set of states and the set of actions are both finite.

In the physical experiments, each observation consists of an initial state, a tilting action, and a final state. For each tilting action u, consider the matrix $\mathbf{X}_u$, where x_{ij} is the number of observations with initial state i and final state j. Given an observation matrix $\mathbf{X}_u$, how do we generate a stochastic transition matrix $\mathbf{P}_u$?

One possible approach is to use the observed frequencies. The difficulty is that some observed transition frequencies may be zero. For such a transition, it isn't clear whether the transition is truly impossible – maybe the transition just has a low probability and hasn't yet been observed. A standard approach from statistical estimation is to use the following estimator for each action u,

$$\hat{p}_{ij}(\mathbf{x}) = \frac{\alpha_{ij} + x_{ij}}{\sum_j \alpha_{ij} + x_{ij}}, \tag{1}$$

where the numbers α_{ij} for $i = 1, 2, \ldots, k$ are Dirichlet parameters based on a priori assumptions. This is equivalent to a Bayes' estimator using a squared error loss criterion.[2]

We could set $\alpha_{ij} = 1.0$ to represent the prior assumption that the conditional probability distribution is uniform: after an action is applied, the system is as likely to be in any one state as in any other. For the tray tilting problem, we set $\alpha_{ij} = 0.01$ to represent our prior assumption that the conditional probability distribution for each action will not be uniformly distributed, but will in fact be skewed toward some subset of states.

We generated 2000 tilt azimuths by random selection (with replacement) from the set of twelve azimuths described previously. Our robot performed the corresponding tilts of the tray, and observed the outcome of each tilt. The $\mathbf{X}_u$ matrices were defined by this data and we used equation (1) to generate the corresponding stochastic transition matrices.

5. PLANNING WITH METHOD I

We now turn to the planning problem: given a stochastic model of actions, a known initial state and desired final state, find a sequence of open-loop tilting actions – a plan – to reach the desired final state. We will present two planning methods and their resulting plans. In this section we describe a method that maximizes the probability of reaching the desired final state. This method has been applied to programming parts feeders.[33]

Method I is based on the control of Markov chains. Consider a system with finite state space. Let us refer to a probability distribution on this state space as a *hyperstate*. Each action is a mapping between hyperstates. A plan is a sequence of actions, which also corresponds to a mapping between hyperstates. For a given initial hyperstate, each plan generates a final hyperstate. To compare plans, we compare their final hyperstates. To rank hyperstates, we introduce a function that maps each hyperstate into a real number called its *cost*. The best plan is the plan with the lowest cost.

Let the vector λ refer to a hyperstate. In a Markov chain, the hyperstate that results from applying action u to λ is given by post-multiplying λ by $\mathbf{P}_u$,

$$\lambda' = \lambda \mathbf{P}_u.$$

A *plan* is a sequence of actions. The outcome of a plan is the composite effect of its inputs; the transition matrix for a plan is the product of the transition matrices of its actions. That is, for a plan consisting of the sequence of actions $\langle u_1, u_4, u_2 \rangle$, the final hyperstate is

$$\lambda' = \lambda \mathbf{P}_{u_1} \mathbf{P}_{u_4} \mathbf{P}_{u_2}.$$

For the tray tilting task we are given a known initial state and desired final state. In this case the initial hyperstate is a vector with a 1 corresponding to the initial state and zeros elsewhere. Each action (and hence plan) defines a final hyperstate using the stochastic transition matrices described in Section 4. The cost function depends on the desired final state. If we want to reach state i, let

$$C(\lambda) = -p_i,$$

so that the minimum cost hyperstate corresponds to the highest probability that the system is in state i. Note that there may be more than one minimum-cost hyperstate.

To compare plans, we compare their final hyperstates. We express our desire for a particular outcome with a cost function on the set of hyperstates. Typically, a hyperstate's cost is determined by a weighted average of its constituent states.

To find the best plan, we consider all plans and find one with minimum cost. The difficulty is that there is an infinite number of plans to consider, requiring an infinite amount of computation (unless we have prior information that the optimal plan is not infinite in length). We compromise and ignore plans longer than some cutoff threshold. This threshold depends on how much time we have and how fast we can evaluate potential plans. In our case we considered all plans with length ≤ 3 to find a plan with minimal cost.

6. PLANNING WITH METHOD II

An alternative planning method is based on heuristic graph search. The transition probability matrices described in Section 4 define a graph, or more correctly, a multigraph. The vertices of the graph are the *states* that were defined previously (the object configurations) and the graph's edges are the *actions* that cause one state to be changed to another. Associated with each edge is an estimate of the probability that the action will provide the indicated state change. The graph is a multi-graph because there may be multiple edges possible between a single pair of vertices, corresponding to multiple actions causing the same state transition.

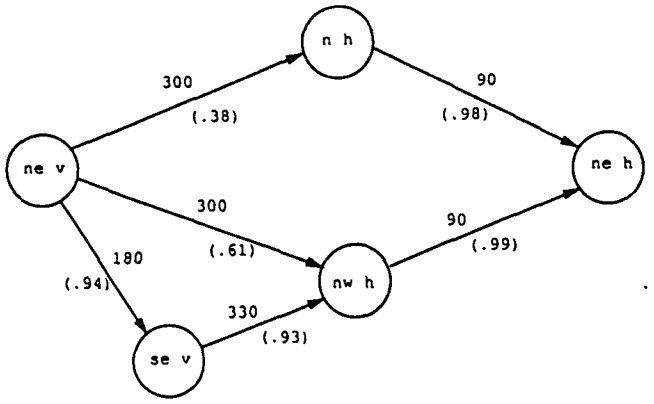

Fig. 3. A portion of the robot's task model.

Figure 3 shows a small portion of the graph defined by our data. Above each edge is a tilt azimuth. Below each edge is the associated transition probability. Note that when the object is in state (*ne v*), and a tilt of azimuth 300 is applied, the object's new state is uncertain. It is estimated that it has a 61% chance of achieving the (*nw h*) configuration, and a 38% chance of moving to the (*n h*) configuration. Of course, not all transitions are shown in Figure 3. The sum of the probabilities for a particular action executed from a particular state must be equal to one.

The planning problem of finding a sequence of actions that will transform a given current state to a desired goal state can now be viewed as finding a path in the graph that links the two states. Since our model of state transitions is stochastic, we naturally wish to find a plan (path) with high probability of achieving the goal. When a plan is executed, several actions are performed in succession. The action probabilities for the plan steps must be combined to give an estimate of the probability of success for the whole plan.

A *lower bound* on the probability that a plan will reach a desired state can be computed by multiplying the probabilities along one path between initial and desired states. Such a computation produces only a lower bound because there can exist multiple paths between the initial and desired states that share the same sequence of action labels. We can find a plan that maximizes this lower bound by using shortest-path graph search. This leads to an efficient algorithm for finding plans that we will call Method II.

Method II is an example of *uniform-cost search.*[34] While uniform-cost search is usually cast as finding a minimum cost path in a graph (where the cost of a path is the sum of the costs of the edges in the path), it is easy to adapt the algorithm to our problem by changing the evaluation function. In finding a minimum cost path in a graph, one desires to find a sequence of edges linking the start and goal vertices such that the sum of the costs of those edges is as small as that of any other such sequence. Our problem is to find a sequence of edges linking the start and goal vertices such that the *product of the probabilities is as large* as any other such sequence. It is possible to map our problem exactly onto a shortest path problem by transforming the values associated with edges. For each edge probability p, we consider a new edge value $(-\log p)$. In this way, we guarantee that finding a shortest path in the transformed graph will correspond exactly to a maximum product probability path in the original graph.

Since Method II only considers single paths, it sometimes misses good plans. Consider when Method II is applied to the problem of getting from state (*ne v*) to (*ne h*), and its stochastic action model is defined by Figure 3. Method II returns the plan (180 330 90). This path's product of action probability estimates is larger than any other single path in the graph. Method II has found a good plan, but note that the plan (300 90) would be better. Under this plan, the configuration achieved after the first tilt is very likely to be either (*n h*) or (*nw h*), and the second tilt is highly likely to achieve the goal no matter which intermediate configuration was actually achieved. The combination of paths yields a high probability even though neither single path has higher probability than the path (180 330 90). Note that Method I would return the better plan in this case but we shall show that Method I requires significantly more computational effort to achieve this rigor.

7. COMPUTATIONAL COMPLEXITY

Our planning problem can be related to finding an optimal control policy for an unobserved Markov chain where the control and observation sets are finite.[13] When system state is perfectly observed at each stage and either the transition probabilities are non-stationary and the time horizon is finite or the transition probabilities are stationary and the cost function decays with time, the problem can be solved in polynomial time by dynamic programming techniques. For cases where the system state is only partially observed (in a known element of a partition) at each stage, the problem is *PSPACE-Hard,* even for stationary, finite-horizon problems.

Finding optimal open-loop plans with stochastic actions has been shown to be *NP-Complete.*[6] This suggests that an algorithm with good worst-case running time may not exist.

Recall that Method I considers all plans up to some length limit, k. Let n be the number of states and m be the number of actions. There are m^k k-step plans. We can visualize the search for an optimal strategy as proceeding through a tree, where the root node contains the initial hyperstate and has a branch for each action in the action space. Each branch leads to a new hyperstate which in turn has branches for each action. We expand the tree to some fixed depth (horizon) and select the optimal path. To generate each node in the tree we must perform $O(n^2)$ multiplications. The total time for finding the best k-step plan is $O(n^2m^k)$.

Recall that Method II is based on uniform-cost graph search. Because the edge values $(-\log p)$ are nonnegative, uniform cost search on this problem has a monotone heuristic, which implies that whenever a node is

expanded, a best path to that node has been found. This means that nodes will never have to be re-expanded, and in a finite graph of n vertices, there can be no more than n node expansions. If there are m actions available at each state, then m is the maximum number of edges that can be between any pair of vertices. Therefore, the maximum amount of work that will have to be done at each node expansion is $O(nm)$, and the complexity of the algorithm is $O(n^2m)$. The implemented algorithm deals with probabilities and products directly instead of transforming the problem to a shortest path problem, but it performs analogous steps to those of uniform cost search on the transformed graph. So, the complexity of Method II is also $O(n^2m)$.

Note that if Method I and Method II were applied in a situation where only 1-step plans were desired, then the complexity for planning with the two methods would differ by no more than a constant factor.

Note the exponential factor m^k in Method I's complexity. For Method II, if the number of actions is increased by a constant factor, the planning time increases by a constant factor. However, if Method I is applied to a task with a constant factor c more actions, then the total planning time would grow by the factor c^k, where k is the length of the longest plan to be considered. In this paper, we consider 12 actions available at each step of a plan. The average planning time was 62 seconds for Method I and 0.46 seconds for Method II. If we considered the case where the tray could be tilted at one degree intervals (360 possible actions), and kept the three step plan length bound, then the number of actions would increase by a factor of 30. Method II's planning time would increase to $30 \cdot 0.46$, or 14 seconds. However, Method I's planning time would swell to $30^3 \cdot 62$ seconds, which is more than 19 days. Thus, Method I becomes less practical as the number of actions to be considered increases.

8. EMPIRICAL COMPARISONS OF THE PLANNERS

The two planning methods were implemented in *Common Lisp*. To explore performance tradeoffs, we performed several experiments with the tray-tilting task. In all experiments reported in this paper, we used an 11 inch square tray and a 1 by 3 inch rectangular tile (Figure 2).

In Section 3, we described the state space of possible tile configurations for the tray-tilting tasks. There are nine possible discrete positions and two discrete orientations, making a total of eighteen possible configurations. It turns out that only twelve of these configurations occur in practice. When the tile is in one of the four corners, both orientations are possible, for a total of eight configurations. The two configurations in the middle of the tray are impossible. For the remaining positions – where the rectangular tile is against a side of the tray – only the orientation where the tile's major axis parallels the tray wall occurs in practice. This adds four more possible configurations, for the total of twelve.

For twelve possible tile configurations, there are 144 pairs of configurations defining start state and goal state. Let us refer to each of these pairs as a *problem*. There are 132 non-trivial problems for our tray domain. (The twelve problems with start state and goal state equal to each other are trivial, since a null plan always solves the problem.) We ran the two planners on each of the 132 non-trivial problems. Table I lists some of the resulting plans.

For 51 (39%) of the problems, the two planners produced identical plans. In many other cases, the two planners produced similar or symmetrically equivalent plans. In nearly every case, the estimated success ratios of the two methods were within a few percentage points of each other. Note that for plans of length ≤ 3, the estimated success ratio P for Method II is less than or equal to that for Method I, since in those cases, Method II's estimate is a lower bound on the Method I estimate.

On some problems, the planners did not agree. In Table I we have listed three such problems (the last three entries). Since Method I was limited to searching for plans of three steps or shorter, there were occasions when Method II was able to search deeper and find a superior plan. The four and five step plans listed in the table are two such examples. On some problems Method I was able to find a superior plan within its length bound by taking advantage of state convergence. The plan (300 90) is an example. In this problem, the initial state is the northeast corner of the tray in a vertical configuration, and the desired goal is the same corner, but in a horizontal configuration. Figure 4 shows a trace of the anticipated object locations as each of the plans proceeds. Method II (below) finds an adequate plan: It tilts the tray at 180, moving the object to the $(se\ v)$ configuration. Then it tilts at 330, moving the object nominally to $(n\ h)$. Finally, it tilts at 90, moving the object to $(ne\ h)$. This plan is good, but it can fail by (for example) the tilt 330 not aligning the object horizontally. Method I's plan (above) is better since its intermediate hyperstate aligns the object horizontally but causes divergence of its position. The second tilt of the plan causes all such intermediate states to converge to $(ne\ h)$.

Both planners were implemented as compiled Common Lisp programs and were tested on the same computer, a Sun 3/280. Method I's search was truncated at depth three, and so all of its plans were between one and three steps in length. Method II's plans were all between one and six steps in length. Over the 132 problems, Method I took an average of 62 seconds real time per problem, with a standard deviation of 2.8 seconds. The average time for Method II on the same problems was 0.46 seconds, with a standard deviation of 0.62 seconds. The minimum planning time for Method I was 59 seconds and the maximum was 87 seconds. The minimum planning time for Method II was 0.040 seconds and the maximum was 6.5 seconds. The average length of plans found by Method I was 2.4 tilts with a standard deviation of 0.75 tilts. The average plan length for the Method II planner was also 2.4 tilts, but with a standard deviation of 1.2 tilts.

Table I. Comparison of plans generated by the two methods.

Problem		Method I		Method II	
Start	Goal	Plan	P	Plan	P
(*n h*)	(*ne h*)	(90)	0.98	(90)	0.98
(*n h*)	(*ne v*)	(240 180 60)	0.97	(240 180 60)	0.96
(*n h*)	(*e v*)	(120)	0.97	(120)	0.97
(*n h*)	(*se h*)	(90 180 90)	0.98	(270 150)	0.96
(*n h*)	(*se v*)	(150 30 180)	0.98	(240 0 120)	0.97
(*n h*)	(*s h*)	(240 150)	0.97	(240 150)	0.97
(*n h*)	(*sw h*)	(270 270 180)	0.97	(270 180)	0.96
(*n h*)	(*sw v*)	(240 180)	0.98	(240 180)	0.97
(*n h*)	(*w v*)	(240)	0.99	(240)	0.99
(*n h*)	(*nw h*)	(270 30 270)	0.97	(270)	0.97
(*n h*)	(*nw v*)	(240 0)	0.98	(240 0)	0.98
		...			
(*sw v*)	(*n h*)	(120 300 30)	0.62	(60 180 240 300 30)	0.75
		...			
(*ne v*)	(*n h*)	(270 30 60)	0.67	(180 240 300 30)	0.75
(*ne v*)	(*ne h*)	(300 90)	0.98	(180 330 91)	0.87

8.1 Physical test of resulting plans

We further tested the last two problems of Table I with the robot. These two problems represented cases where the methods produced significantly different plans. In terms of predicted success ratios (probability of reaching the goal), Method II found a better plan for the first problem because it was able to search deeper. In the second problem, Method I found a better plan because it considered state divergence and state convergence.

Table II summarizes the head-to-head competition. Each of the four plans was executed 200 times. In the first problem, the estimated success ratio was lower than the observed success ratio. There are at least two explanations for this. First, the estimates for these plans may be low because insufficient data had been collected to predict the correct transition probabilities for the actions comprising the plans. This could be corrected by taking more experimental trials prior to planning. A second explanation is that the circumstances under which the data were collected did not truly reflect the circumstances under which the plans were executed. The data were collected by selecting tilt azimuths at random from the set of twelve nominal aximuths, and this may have led to a greater variety of tilt outcomes than is the case when a sequence of high-quality plans is executed. Since we don't know a priori which plans are the best, it is unclear how to correct this possible source of error.

In the second problem, the estimated and observed success rates were nearly equal. In both cases selected for detailed testing, the plan with higher estimated success rate performed better.

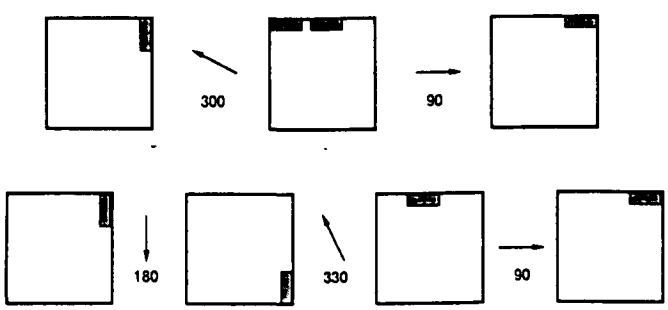

Fig. 4. Two tray-tilting plans. The plans are read from left to right, with intermediate states shown as views from above the tray. Tray tilt directions are shown between the states that they link. The problem is to re-orient the object from vertical to horizontal, leaving the object in the upper right corner of the tray. Above: Method I's plan. Below: Method II's plan.

9. DISCUSSION

We began this paper by acknowledging that it is not always possible to predict the exact outcome of actions in robotic manipulation due to factors such as control error, friction, and dynamics. In response, we considered *stochastic* models of action. When actions are viewed as stochastic, the conventional planning paradigm must be modified to search for plans that achieve a goal with high probability.

For the class of problems where state and action spaces are finite, we can view planning as finding a sequence of actions that transfer probability mass from the initial state to the desired final state. Certainly it is true that the state and action spaces in many robot planning problems are not finite. However, they can often be discretized using a uniform grid or preferably, with a "criticality"-based partition justified by system geometry.[35]

Planning is affected by two contrasting properties of stochastic actions. We say that an action exhibits *state divergence* when it distributes probability mass. We say that an action exhibits *state convergence* when it focuses probability mass. These properties affect the choice of planing method. We considered two methods.

Table II. Summary of execution trials for four plans. The estimated and measured success columns show the number of successful plan executions (out of 200 trials) along with the corresponding decimal fractions. Method II finds a better plan in the first instance because Method I neglects plans longer than 3 steps. However in the second instance, Method I finds a better plan because Method II neglects plans which incur state divergence.

Start State	Goal State	Planning Approach	Best Plan	Estimated Successes	Measured Successes
(*ne v*)	(*n h*)	Method I	(270 30 60)	134 (0.67)	170 (0.85)
(*ne v*)	(*n h*)	Method II	(180 240 300 30)	150 (0.75)	192 (0.96)
(*ne v*)	(*ne h*)	Method I	(300 90)	196 (0.98)	198 (0.99)
(*ne v*)	(*ne h*)	Method II	(180 330 90)	174 (0.87)	171 (0.855)

Method I uses a forward exhaustive search (to a fixed plan length) to find a plan most likely to produce the desired goal. Method II uses a backward best-first search to find a plan that maximizes a lower bound on the probability of reaching the goal. Method I keeps track of *all* probability mass as it evaluates plans, monitoring both state divergence and state convergence as the plan progresses. However, keeping track of all probability mass requires substantial computation; Method I can only consider short plans. Method II keeps track of only the most probable trajectory, monitoring state divergence and ignoring state convergence. Accordingly, Method II is faster and can consider longer plans, but it sometimes misses good plans.

Which method is better? It depends on the available planning time and the degree of state divergence in the available actions. Because Method I checks all plans requiring fewer than k actions, Method II cannot find a better plan within this depth, However, depending on the number of available actions, the achievable value of k could be quite small, prompting us to prefer Method II, especially when we can avoid or minimize state divergence.

Sometimes divergence is unavoidable. For example, consider a tray-tilting problem where we want to achieve a particular object configuration but the initial configuration is unknown. This is equivalent to a highly divergent action occurring before plan execution. It is as if someone randomly shook the tray prior to the robot carrying out its plan. Consider a case where the initial hyperstate reflects a uniformly distributed state probability. Method I can find the best plan for getting the tile into the northeast corner. Method II can't solve this problem. As another example, consider programming a robot to pick up parts on an assembly line. Each time a part arrives, its initial position and orientation relative to the robot will be slightly different. This results from a divergent action earlier in the assembly line.

Sometimes state divergence is desirable. For example, it may be more efficient to allow divergence followed by effective convergence. Consider a typical plan for causing two gears to mesh: we jiggle the gears at random (divergence), until the gears fall into alignment (convergence). This plan is more efficient than trying to avoid divergence by carefully aligning the gears. The process of deliberately incurring divergence is known as *randomization*, and has been recently identified as an important component of manipulation.[36]

For most problems in the tray-tilting domain, Method II found plans that were comparable to those found by Method I. Studying the resulting plans, we discovered it was often possible to avoid significant state divergence. This is a property of the tray-tilting task, and is not guaranteed to be present in all manipulation tasks. For a few problems, it was better to incur state divergence followed by state convergence. An example of such a problem was given in Figure 3. On a problem like this, Method I is superior.

It may be possible to build a hybrid of these two methods: an efficient planner that considers some state convergence. Instead of tracking the probabilities for every state, as Method I does, or tracking the probability of only the most likely state, as Method II does, maybe the hypothetical planner could track the probabilities of the two or three most likely states. Like Method II, the hypothetical planner would, in most cases, find a plan of near-optimal quality. Even more desirable would be an approximation algorithm, where we might be able to guarantee, for all problems, that the plans produced would be suboptimal by no more than a fixed constant factor.

Uncertainty is inevitable for any robot operating in the physical environment. When the outcome of each potential action is unpredictable, planning a reliable sequence of robot actions is extremely difficult. Yet reliability is essential if robots are to be adopted in new applications. The past thirty years of research in robotics has taught us that it is extremely difficult to derive accurate analytic models for simple actions such as pushing a part across a table. More complex actions are even harder to model analytically. One alternative is to consider stochastic models obtained from physical observations. In this paper we have considered the problem of planning with such models.

Since finding optimal plans is known to be *NP-Complete*, we compared an exponential-time planner that finds optimal plans with a polynomial-time planner that finds sub-optimal plans. Intuitively, there should be a tradeoff between planning time and plan performance. We explored this tradeoff by comparing the two algorithms in the context of a physical application. These experiments led us to identify two properties of actions

that can aid in choosing between the algorithms for future applications where state and action spaces are finite.

Acknowledgements

We thank Matt Mason for suggesting this collaboration. We also thank Matt, Rob Kass, Kevin Lynch, Tom Mitchell, Steve Shreve, and Manuela Veloso for helpful comments. This paper is a revised and expanded version of a paper that appeared in the proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control, in November 1990.

References

1. K.J. Astrom, "Adaptive feedback control." *Proceedings of the IEEE* **75**(2) 185–217 (1987).
2. M.H. DeGroot, *Optimal Statistical Decisions* (McGraw-Hill, New York, 1970).
3. J.O. Berger, *Statistical Decision Theory and Bayesian Analysis* (Springer-Verlag, Berlin, 1985).
4. R.A. Howard, *Dynamic Probabilistic Systems (2 volumes)* (John Wiley, New York, 1971).
5. E.B. Dynkin and A.A. Yushkevich, *Controlled Markov Processes* (Springer-Verlag, Berlin, 1979).
6. C.H. Papadimitriou and J.N. Tsitsiklis, "The complexity of Markov decision processes" *Mathematics of Operations Research*, **12**(3) 441–450 (August 1987).
7. J.A. Feldman and R.F. Sproull, "Decision theory and artificial intelligence II: The hungry monkey". *Cognitive Science*, **1** 158–192 (1977).
8. J. Pearl, *Probabilistic Reasoning in Intelligent Systems* (Morgan-Kaufmann, Los Altos, California, 1988).
9. S. Russell and E. Wefald "Decision-theoretic control of reasoning: General thery and an application to game playing" *Technical Report UCB/CSD 88/435* (UC Berkeley, October 1988).
10. P. Cheeseman "In defense of probability" *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, IJCAI (August 1985) pp. 1002–1009.
11. H.A. Simon, "A behavioral model of rational choice" *Quart. J. Economics* **69**: 99–118 (1955). Reprinted in *Models of Thought* (Yale University Press, 1979).
12. O. Etzioni, "Tractable decicion-analytic control" *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, Los Altos, California (1989) (Morgan Kaufmann, San Mateo, 1989). An expanded version is available as technical report CMU-CS-89-119.
13. D.P. Bertsekas, *Dynamic Programming: Deterministic and Stochastic Models* (Prentice-Hall, Englewood Cliffs, New Jersey, 1987).
14. T.L. Dean and M.P. Wellman, *Planning and Control.* (Morgan Kaufmann, San Mateo, California, 1991).
15. J. Simons, H. Van Brussel, J. De Schutter, and J. Verhaert, "A self-learning automation with variable resolution for high precision assembly by industrial robots" *IEEE Transactions on Automatic Control* **AC27**(5) 1109–1113 (October 1982).
16. K.S. Narendra and M.A.L. Thathachar *Learning Automata: An Introduction* (Prentice Hall, Englewood Cliffs, New Jersey, 1988).
17. B. Dufay and J.-C. Latombe, "An approach to automatic robot programming based on inductive learning" *International Symposium on Robotics Research* (1983) pp. 97–115.
18. A.G. Barto, "Connectionist learning for control: An overview" *Technical Report COINS 89–89* (University of Massachusetts-Amherst, September 1989).
19. S.J. Buckley, "Teaching compliant motion strategies" *IEEE J. Robotics and Automation* **5**(1) 112–118 (1989).
20. K.P. Gross, "Concept Acquisition through Attribute Evolution and Experimentation" *PhD thesis* (Carnegie Mellon University, School of Computer Science, May 1991).
21. S. Bennett and G. DeJong, "Comparing stochastic planning to the acquisition of increasingly permissive plans for complex, uncertain domains" *International Workshop on Machine Learning* (June 1991) pp. 586–590.
22. A.D. Christiansen, "Automatic Acquisition of Task Theories for Robotic Manipulation" *PhD thesis* (Carnegie Mellon University, School of Computer Science, March 1992).
23. M.T. Mason, "Mechanics and planning of manipulator pushing operations" *Int. J. Robotics Research* **5**(3) 53–71 (1986).
24. R. C. Brost, "Automatic grasp planning in the present of uncertainty" *Int. J. Robotics Research* **7**(1) 3–17 (February, 1988).
25. J.C. Trinkle, J.M. Abel and R.P. Paul, "An investigation of frictionless enveloping grasping in the plane" *Int. J. Robotics Research* **7**(3) 33–51 (June, 1988).
26. M.A. Peshkin, "The motion of a pushed, sliding workpiece" *IEEE Transactions on Robotics and Automation* **4**(6) 569–598 (December, 1988).
27. R.C. Brost, "Analysis and Planning of Planar Manipulation Tasks" *PhD thesis* (Carnegie Mellon University, School of Computer Science, January 1991).
28. M.A. Erdmann and M.T. Mason, "An exploration of sensorless manipulation" *IEEE J. Robotics and Automation* **4**(4) 369–379 (August, 1988).
29. R.H. Taylor, M.T. Mason and K.Y. Goldberg, "Sensor-based manipulation planning as a game with nature" *International Symposium on Robotics Research* (August, 1987) pp. 421–429.
30. A.D. Christiansen, "Manipulation planning from empirical backprojections" *IEEE International Conference on Robotics and Automation* (April, 1991) pp. 762–768.
31. A.D. Christiansen, M.T. Mason and T.M. Mitchell, "Learning reliable manipulation strategies without initial physical models" *Robotics and Autonomous Systems* **8** 7–18 (November, 1991).
32. A.D. Christiansen, "Learning to predict in uncertain continuous tasks" *International Conference on Machine Learning* (July, 1992) pp. 72–81.
33. K.Y. Goldberg, "Stochastic Plans for Robotic Manipulation" *PhD thesis* (Carnegie Mellon University, School of Computer Science, August 1990).
34. J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* (Addison-Wesley, Reading, Massachusetts, 1984).
35. R. Wilson and J.-C. Latombe, "On the qualitative structure of a mechanical assembly" *National Conference on Artificial Intelligence* (1992) pp. 697–702.
36. M.A. Erdmann, "On Probabilistic Strategies for Robot Tasks" *PhD. thesis* (MIT, Cambridge, MA, 1989).