An Algorithm for Probabilistic Least-Commitment Planning*

Nicholas Kushmerick Steve Hanks Daniel Weld
Department of Computer Science and Engineering, FR-35
University of Washington Seattle, WA 98195
{*nick, hanks, weld*}*@cs.washington.edu*

Abstract

We define the probabilistic planning problem in terms of a probability distribution over initial world states, a boolean combination of goal propositions, a probability threshold, and actions whose effects depend on the execution-time state of the world and on random chance. Adopting a probabilistic model complicates the definition of plan success: instead of demanding a plan that *provably* achieves the goal, we seek plans whose probability of success exceeds the threshold.

This paper describes a probabilistic semantics for planning under uncertainty, and presents a fully implemented algorithm that generates plans that succeed with probability no less than a user-supplied probability threshold. The algorithm is sound (if it terminates then the generated plan is sufficiently likely to achieve the goal) and complete (the algorithm will generate a solution if one exists).

Introduction

Classical planning algorithms have traditionally adopted stringent certainty assumptions about their domains: the planning agent is assumed to have complete and correct information about the initial state of the world and about the effects of its actions. These assumptions allow an algorithm to build a plan that is *provably* correct: given an initial world state, a successful plan is a sequence of actions that logically entails the goal. Our research effort is directed toward relaxing the assumptions of complete information and a deterministic action model, while exploiting existing techniques for symbolic least-commitment plan generation.

This paper presents the BURIDAN[1] planning algorithm. BURIDAN is a sound, complete, and fully implemented least-commitment planner whose underlying semantics is probabilistic: a probability distribution over states captures the agent's uncertainty about the world and a mixed symbolic and probabilistic action representation allows the effects of action to vary according to the (modeled) state of the world at execution time as well as unmodeled (random) factors.

Our planner takes as input a probability distribution over (initial) states, a goal expression, a set of action descriptions, and a probability threshold representing the minimum acceptable success probability. The algorithm produces a plan such that, given a probability distribution over initial states, the probability that the goal holds after executing the plan is no less than the threshold. We have proved that the algorithm is sound (any plan it returns is a solution) and complete (if there is an acceptable plan the algorithm will find it).

The work reported here makes several contributions. First, we define a symbolic action representation and its probabilistic semantics. Second, we describe an implemented algorithm for probabilistic planning. Third, we briefly describe our investigation of alternative plan assessment strategies. The paper concludes with a discussion of related work.

This research is described in detail in the long version of this paper (Kushmerick, Hanks, & Weld 1993).

Example. The following example will be developed throughout the paper. Suppose a robot is given the goal of holding a block (HB), making sure it is painted (BP), and simultaneously keeping its gripper clean (GC). Initially the gripper is clean, but the block is not being held and is not painted, and the gripper is dry (GD) with probability 0.7. Suppose further that we will accept any plan that achieves the goal with probability at least 0.8. Finally, the robot has the following actions available:

- pickup: try to pick up the block. If the gripper is dry when the pickup is attempted, executing this action will make HB true with probability 0.95. If the gripper is not dry, however, HB will become true only with probability 0.5.

*We gratefully acknowledge the comments and suggestions of Tony Barrett, Tom Dean, Denise Draper, Mike Erdmann, Keith Golden, Rex Jacobovits, Oren Etzioni, Neal Lesh, Judea Pearl, and Mike Williamson. This research was funded in part by National Science Foundation Grants IRI-9206733 and IRI-8957302, Office of Naval Research Grant 90-J-1904, and the Xerox Corporation.

[1] Jean Buridan (bo͞o rē dän′), 1300-58, a French philosopher and logician, has been credited with originating probability theory. He seems to have toyed with the idea of using his theory to decide among alternative courses of action: the parable of "Buridan's Ass" is attributed to him, in which an ass that lacked the ability to choose starved to death when placed between two equidistant piles of hay.

- paint: paint the block. This action always makes BP true, and if the robot is holding the block when paint is executed, there is a 10% chance that the gripper will become dirty ($\overline{\mathsf{GC}}$).
- dry: try to dry the gripper. This action succeeds in making GD true 80% of the time.

A Semantics for Probabilistic Planning

We begin by defining a planning problem, and what it means to solve one. First we defining states and expressions, then actions and sequences of actions, then the planning problem and its solution.

States & expressions. A *state* is a complete description of the world at a single point in time, and uncertainty about the world is represented using a random variable over states. A state is described using a set of *propositions* in which every proposition appears exactly once, possibly negated.[2] An *expression* is a set (implicit conjunction) of literals. We define the probability of an expression $\mathcal{E}$ with respect to a state s as:

$$\mathrm{P}[\mathcal{E} \mid s] = \begin{cases} 1 & \text{if } \mathcal{E} \subseteq s \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

For our example, the world is initially in one of two possible states: $s_1 = \{\mathsf{GD}, \overline{\mathsf{HB}}, \mathsf{GC}, \overline{\mathsf{BP}}\}$ and $s_2 = \{\overline{\mathsf{GD}}, \overline{\mathsf{HB}}, \mathsf{GC}, \overline{\mathsf{BP}}\}$, and the probability distribution over these states is characterized by a random variable $\tilde{\mathsf{s}}_I$ as follows: $\mathrm{P}[\tilde{\mathsf{s}}_I = s_1] = 0.7$, $\mathrm{P}[\tilde{\mathsf{s}}_I = s_2] = 0.3$.

Actions & action sequences. Our model of action, taken from (Hanks 1990; 1993), combines a symbolic model of the changes the action makes to propositions with probabilistic parameters that represent chance (unmodeled) influences. Fig. 1 is our representation of the pickup action: if the gripper is dry (GD holds) at execution time, it makes HB true with probability 0.95, and with probability 0.05 makes no change to the world state. But if GD is false at execution time, pickup makes HB true only with probability 0.5. Note that the propositions in the boxes refer to *changes* the action makes, not to world states. For example, it is not correct to say that the HB holds with probability 0.95 after executing pickup in a state where the gripper is dry, since the probability of HB *after* pickup is executed also depends on the probability of HB *before* execution (as well as the probability of GD before execution).

Formally, an action is a set of *consequences* $\{\langle \mathsf{t}_\alpha, \rho_\alpha, \mathsf{e}_\alpha \rangle, \ldots, \langle \mathsf{t}_\eta, \rho_\eta, \mathsf{e}_\eta \rangle\}$. Each t_ι is an expression called the consequence's *trigger*, ρ_ι is a probability, and e_ι is a set of literals called the *effects*. The representation for the pickup action is thus $\{(\{\mathsf{GD}\}, 0.95, \{\mathsf{HB}\}), (\{\mathsf{GD}\}, 0.05, \{\}), (\{\overline{\mathsf{GD}}\}, 0.5, \{\mathsf{HB}\}), (\{\overline{\mathsf{GD}}\}, 0.5, \{\})\}$.

[2] We use this representation for expository purposes only; an implementation need not manipulate states explicitly. In fact our plan refinement algorithm has no explicit representation of state: it reasons directly about the state's component propositions.

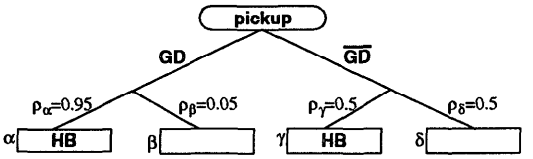

Figure 1: The pickup action.

We require that an action's triggers be mutually exclusive and exhaustive:

$$\forall \iota \qquad \sum_s \rho_\iota \mathrm{P}[\mathsf{t}_\iota \mid s] = 1 \quad (2)$$

$$\forall s, \iota, \kappa \qquad \mathsf{t}_\iota \neq \mathsf{t}_\kappa \Rightarrow \mathrm{P}[\mathsf{t}_\iota \cup \mathsf{t}_\kappa \mid s] = 0 \quad (3)$$

The notation $\mathsf{A}_{i,\iota}$ refers to consequence ι of action A_i, and superscripts refer to parts of a particular action: $\mathsf{A}_i = \{\ldots, \langle \mathsf{t}^i_\iota, \rho^i_\iota, \mathsf{e}^i_\iota \rangle, \ldots\}$.

A consequence defines a (deterministic) transition from a state to a state, defined by a function RESULT(e, s), where e is a set of effects and s is a state. This function is similar to add and delete lists in STRIPS; see the long version of this paper for the full definition.

An action A induces a change from a state s to a probability distribution over states s':

$$\mathrm{P}[s' \mid s, \mathsf{A}] = \begin{cases} \rho_\iota \mathrm{P}[\mathsf{t}_\iota \mid s] & \text{if } \langle \mathsf{t}_\iota, \rho_\iota, \mathsf{e}_\iota \rangle \in \mathsf{A} \wedge s' = \textsc{result}(\mathsf{e}_\iota, s) \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

Since an action's triggers are mutually exclusive and exhaustive, we have that $\sum_{s'} \mathrm{P}[s' \mid s, \mathsf{A}] = 1$ for every action A and state s.

We now define the result of executing actions in sequence. The probability that a state s' will hold after executing a sequence of actions $\langle \mathsf{A}_i \rangle_{i=1}^N$ (given that the world was initially in state s) is defined as follows:

$$\mathrm{P}\left[s' \mid s, \langle \mathsf{A}_i \rangle_{i=1}^N\right] = \sum_{s''} \mathrm{P}[s'' \mid s, \mathsf{A}_1] \mathrm{P}\left[s' \mid s'', \langle \mathsf{A}_i \rangle_{i=2}^N\right] \quad (5)$$

where $\mathrm{P}[s' \mid s, \langle\rangle] = 1$ if $s' = s$ and 0 otherwise.

Finally, we define the probability that an expression $\mathcal{E}$ is true after an action sequence is executed beginning in some state s, and the probability of an expression after executing an action sequence given an initial probability distribution over states $\tilde{\mathsf{s}}_I$:

$$\mathrm{P}\left[\mathcal{E} \mid s, \langle \mathsf{A}_i \rangle_{i=1}^N\right] = \sum_{s'} \mathrm{P}\left[s' \mid s, \langle \mathsf{A}_i \rangle_{i=1}^N\right] \mathrm{P}[\mathcal{E} \mid s'] \quad (6)$$

$$\mathrm{P}\left[\mathcal{E} \mid \tilde{\mathsf{s}}_I, \langle \mathsf{A}_i \rangle_{i=1}^N\right] = \sum_{s} \mathrm{P}\left[\mathcal{E} \mid s, \langle \mathsf{A}_i \rangle_{i=1}^N\right] \mathrm{P}[\tilde{\mathsf{s}}_I = s] \quad (7)$$

Planning problems & solutions. A planning problem consists of (1) a probability distribution over initial states $\tilde{\mathsf{s}}_I$, (2) a set of actions $\{\mathsf{A}_i\}$, (3) a goal expression $\mathcal{G}$, and (4) a probability threshold τ. For the problem described in this paper, the initial distribution $\tilde{\mathsf{s}}_I$ was defined earlier, the plan can be constructed from the actions {pickup, paint, dry}, the goal

is $\mathcal{G} = \{\mathsf{HB}, \mathsf{BP}, \mathsf{GC}\}$, and the probability threshold is $\tau = 0.8$.

A *solution* to the problem is an action sequence $\langle \mathsf{A}_i \rangle_{i=1}^{N}$ if $\mathrm{P}\left[\mathcal{G} \mid \tilde{\mathsf{s}}_I, \langle \mathsf{A}_i \rangle_{i=1}^{N}\right] \geq \tau$. As we'll see, $\langle$dry, paint, pickup$\rangle$ is a solution to the example problem.

The BURIDAN Algorithm

We now describe BURIDAN, an algorithm that generates solutions to planning problems. BURIDAN, like SNLP (McAllester & Rosenblitt 1991), searches a space of partial *plans*. Each plan consists of a set of *actions* $\{\mathsf{A}_i\}$ where each A_i is one of the input actions annotated with a unique index, a partial *temporal ordering* relation "<" over $\{\mathsf{A}_i\}$, a set of *causal links*, and a set of *subgoals*. The first two items are straightforward, but there are important differences between the last two items and the analogous SNLP concepts.

A link caches BURIDAN's reasoning that a particular consequence of a particular action could make a literal true for a (later) action in the plan. The link $\mathsf{A}_{i,\iota} \xrightarrow{\mathsf{p}} \mathsf{A}_j$ records the fact that literal p is a member of the trigger of one of action A_j's consequences (A_j is the link's *consumer*), and the effect set of consequence ι of action A_i (the link's *producer*) contains p. Action A_k *threatens* link $\mathsf{A}_{i,\iota} \xrightarrow{\mathsf{p}} \mathsf{A}_j$ if the effect set of some consequence of A_k contains $\overline{\mathsf{p}}$, and if A_k can be ordered between A_i and A_j.

A plan's subgoals consists of the literals in the plan that BURIDAN is committed to making true. A subgoal is a literal annotated with a particular action, written $\mathsf{p}@\mathsf{A}_i$. $\mathsf{p}@\mathsf{A}_i$ is a subgoal if the plan contains a link $\mathsf{A}_{i,\iota} \xrightarrow{\mathsf{q}} \mathsf{A}_j$ and $\mathsf{p} \in \mathsf{t}_\iota^i$. The set of subgoals is initialized to include all top-level goals.

BURIDAN begins searching from the *null plan*, which contains only two dummy actions A_0 and A_G, and the ordering constraint $\mathsf{A}_0 < \mathsf{A}_G$. These two actions allow BURIDAN to compactly encode the planning problem: A_0 and A_G encode the probability distribution over initial states and the goal expression, respectively. Fig. 2 shows the null plan for the example developed in this paper. The initial action A_0 has one consequence for each state in the initial distribution $\tilde{\mathsf{s}}_I$ that has nonzero probability. Its triggers are all empty, and the effect sets and probabilities are the states and probabilities defined by $\tilde{\mathsf{s}}_I$. The goal action A_G has one consequence triggered by the goal expression $\mathcal{G}$. The null plan adopts $\mathsf{p}@\mathsf{A}_G$ as a subgoal for each $\mathsf{p} \in \mathcal{G}$.

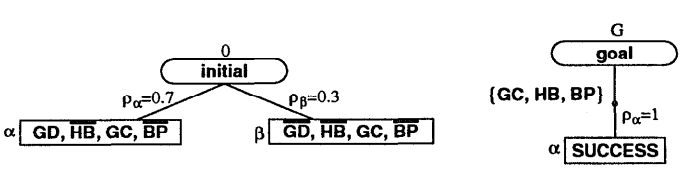

Figure 2: A_0 and A_G encode the initial probability distribution and the goal.

Starting from the null plan, BURIDAN performs two operations:

1. *Plan Assessment:* Determine if the probability that the current plan will achieve the goal exceeds τ, terminating successfully if so.
2. *Plan Refinement:* Otherwise, try to increase the probability of goal satisfaction by refining the current plan. Each refinement generates a new node in the space of partial plans. Signal failure if there are no possible refinements, otherwise nondeterministically choose a new current plan and loop.

Refining a plan with conditional and probabilistic actions differs in two ways from classical causal-link refinement algorithms (*e.g.* SNLP). First, SNLP establishes a single causal link between a producing *action* and a consuming action, and that link alone ensures that the link's literal will be true when the consuming action is executed. Our planner links one of an action's *consequences* to a later action. An action can have several consequences, though only one will actually occur. Furthermore, a single link $\mathsf{A}_{i,\iota} \xrightarrow{\mathsf{p}} \mathsf{A}_j$ ensures that p will be true at action A_j only if trigger t_ι^i holds with probability one. Therefore multiple links may be needed to support a literal: even if no single link makes the literal sufficiently likely, their combination might. We lose SNLP's clean distinction between an "open condition" (a trigger that is not supported by a link) and a "supported condition" that is guaranteed to be true. Causal support in a probabilistic plan is a cumulative concept: the more links supporting a literal, the more likely it is that the literal will be true.

The concept of a threatened link is different when actions have conditional effects. Recall that A_k threatens $\mathsf{A}_{i,\iota} \xrightarrow{\mathsf{p}} \mathsf{A}_j$ if some consequence of A_k asserts $\overline{\mathsf{p}}$ and if A_k can be ordered between A_i and A_j. BURIDAN resolves threats in the same way that classical planners do: by ordering the threatening action either before the producer or after the consumer. But a plan can be sufficiently likely to succeed even if there is a threat, as long as the threat is sufficiently *unlikely* to occur. We can therefore resolve a threat in an additional way, by *confrontation*: if action A_k threatens link $\mathsf{A}_{i,\iota} \xrightarrow{\mathsf{p}} \mathsf{A}_j$, plan for the occurrence of some consequence of A_k that does *not* make p false. BURIDAN does so by adopting that consequence's triggers as additional subgoals.

Plan Refinement

BURIDAN's plan refinement step generates all possible refinements of a partial plan. A plan can be refined in two ways: either resolving a threat to a causal link, or adding a link to increase the probability that a subgoal will be true. BURIDAN chooses a threat or subgoal, and then generates possible refinements as follows:

1. If the choice is to add a link to the subgoal $\mathsf{p}@\mathsf{A}_j$, BURIDAN considers all new and existing actions A_i that have an effect set e_ι^i containing p, adds a link $\mathsf{A}_{i,\iota} \xrightarrow{\mathsf{p}} \mathsf{A}_j$, and orders $\mathsf{A}_i < \mathsf{A}_j$.

2. If the choice is to resolve a threat by action A_k to link $A_{i,\iota} \xrightarrow{p} A_j$, BURIDAN resolves the threat in one of three ways: *demotion* (order $A_k < A_i$ if it is consistent to do so), *promotion* (order $A_j < A_k$ if consistent), and *confrontation* (plan for the occurrence of a consequence of action A_k that does *not* make p false).

Link creation, promotion, and demotion are analogous to refinement in SNLP-like planners, and we will not discuss them further.[3] Confrontation has no analogue in SNLP, however. The probability that link $A_{i,\iota} \xrightarrow{p} A_j$ succeeds in producing p for A_j is the probability that executing action A_i actually realizes consequence ι and that each action representing a confronted threat realizes a consequence that does not make p false. Since the consequences of action are mutually exclusive, making a non-threatening consequence more likely makes the threat less likely. BURIDAN therefore confronts a threat by choosing a non-threatening consequence of the threatening step, and adopts its triggers as subgoals. We explain confrontation in detail in the longer version of this paper.

Example. We now demonstrate how the planner constructs a plan that will achieve the goal (holding a painted block with a clean gripper) with probability at least 0.8. We simplify the presentation by presenting a sequence of refinement choices that lead directly to a solution plan.

Planning starts with the null plan (Fig. 2). The subgoals for the null plan are $\{\mathsf{HB}@A_G, \mathsf{BP}@A_G, \mathsf{GC}@A_G\}$. BURIDAN supports the first subgoal, $\mathsf{HB}@A_G$, by adding a pickup action, A_1, and creating a link from its α consequence: $A_{1,\alpha} \xrightarrow{\mathsf{HB}} A_G$. Consequence α of A_1 has GD as a trigger, so BURIDAN adopts $\mathsf{GD}@A_1$ a subgoal. Support for this subgoal is then provided by a link from the initial action: $A_{0,\alpha} \xrightarrow{\mathsf{GD}} A_1$.

BURIDAN next supports the subgoal of having the block painted, $\mathsf{BP}@A_G$, by adding a new paint action, A_2, creating a link $A_{2,\beta} \xrightarrow{\mathsf{BP}} A_G$, and adopting $A_{2,\beta}$'s trigger $\overline{\mathsf{HB}}@A_2$ as a subgoal. A link $A_{0,\alpha} \xrightarrow{\overline{\mathsf{HB}}} A_2$ is added to support $\overline{\mathsf{HB}}@A_2$. Notice that pickup and paint are unordered, thus pickup threatens the new link: if pickup is executed before paint, then the block will be in the gripper when paint is executed, making false the trigger $\overline{\mathsf{HB}}$ of paint's β consequence. BURIDAN resolves this threat by promoting pickup, adding the ordering constraint $A_2 < A_1$. Finally, BURIDAN supports the subgoal $\mathsf{GC}@A_G$ with the link $A_{0,\alpha} \xrightarrow{\mathsf{GC}} A_G$, and resolves the threat posed by paint by confrontation: since consequence β of A_2 does not cause $\overline{\mathsf{GC}}$, BURIDAN adopts its trigger $\overline{\mathsf{HB}}@A_2$ as a subgoal. The resulting plan is shown in Fig. 3.

The assessed probability of Fig. 3's plan is 0.7335 (as described below), which is less than $\tau = 0.8$, so BURIDAN continues to refine the plan. Eventually the plan shown in Fig. 4 is generated; it has success probability 0.804, so BURIDAN terminates.

[3] We ignore SNLP's *separation* refinement, since it applies only to actions with variables.

Plan Assessment

The plan assessment algorithm decides whether a plan's probability of success exceeds the threshold τ. The FORWARD assessment algorithm is a straightforward implementation of the definition of action execution.[4] FORWARD computes the probability distribution over states produced by "executing" each action in sequence, pruning zero-probability states from the distribution for the sake of efficiency.

Complicating assessment is the fact that a solution is defined in terms of a totally ordered sequence of actions, whereas a plan's actions might be only partially ordered. We can still compute a lower bound on the plan's success, however, by considering the minimum over all total orders consistent with the plan's orderings. This policy is conservative in that it computes the best probability that can be expected from *every* total order. The long version of this paper discusses these issues in more detail.

Example. We now illustrate how the FORWARD plan assessment algorithm computes the success probability for the plan in Fig. 3. FORWARD starts with the input distribution over initial states, $\tilde{s}_I$, and computes the following distribution over final states (where $\mathcal{A} = \langle \mathsf{paint}, \mathsf{pickup} \rangle$ is the plan's action sequence):

$$\begin{aligned}
P[\{\mathsf{GD},\mathsf{HB},\mathsf{GC},\mathsf{BP}\} \mid \tilde{s}_I, \mathcal{A}] &= 0.5985 \\
P[\{\overline{\mathsf{GD}},\mathsf{HB},\mathsf{GC},\mathsf{BP}\} \mid \tilde{s}_I, \mathcal{A}] &= 0.135 \\
P[\{\mathsf{GD},\mathsf{HB},\overline{\mathsf{GC}},\mathsf{BP}\} \mid \tilde{s}_I, \mathcal{A}] &= 0.0665 \\
P[\{\overline{\mathsf{GD}},\mathsf{HB},\overline{\mathsf{GC}},\mathsf{BP}\} \mid \tilde{s}_I, \mathcal{A}] &= 0.015 \\
P[\{\mathsf{GD},\overline{\mathsf{HB}},\mathsf{GC},\mathsf{BP}\} \mid \tilde{s}_I, \mathcal{A}] &= 0.0315 \\
P[\{\overline{\mathsf{GD}},\overline{\mathsf{HB}},\mathsf{GC},\mathsf{BP}\} \mid \tilde{s}_I, \mathcal{A}] &= 0.135 \\
P[\{\mathsf{GD},\overline{\mathsf{HB}},\overline{\mathsf{GC}},\mathsf{BP}\} \mid \tilde{s}_I, \mathcal{A}] &= 0.0035 \\
P[\{\overline{\mathsf{GD}},\overline{\mathsf{HB}},\overline{\mathsf{GC}},\mathsf{BP}\} \mid \tilde{s}_I, \mathcal{A}] &= 0.015
\end{aligned}$$

The probability of the goal expression is then computed by summing over those states in which the goal holds. $\mathcal{G} = \{\mathsf{HB},\mathsf{GC},\mathsf{BP}\}$ is true in the first two states, so we have $P[\mathcal{G} \mid \tilde{s}_I, \mathcal{A}] = 0.5985 + 0.135 = 0.7335$.

Formal Properties

A least-commitment planner produces as output a partial order over actions. Such a planner is *sound* if every consistent total order of these actions is a solution to the input problem. The planner is *complete* if it always returns a solution plan if such a plan exists. In the longer paper we prove that BURIDAN is both sound and complete.

[4] In this section we discuss one simple assessment strategy; later we introduce a variety of more complicated algorithms.

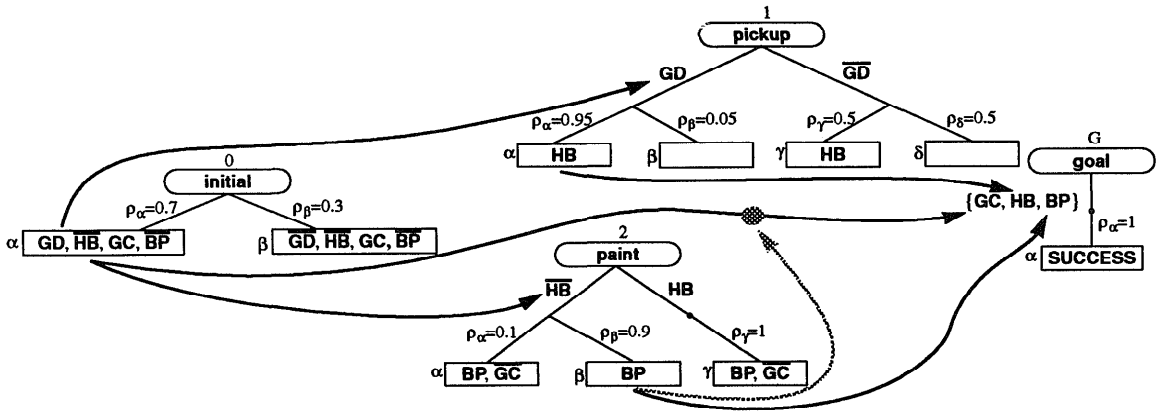

Figure 3: An partial solution to the example problem. Gray arrows indicate threats resolved by confrontation.

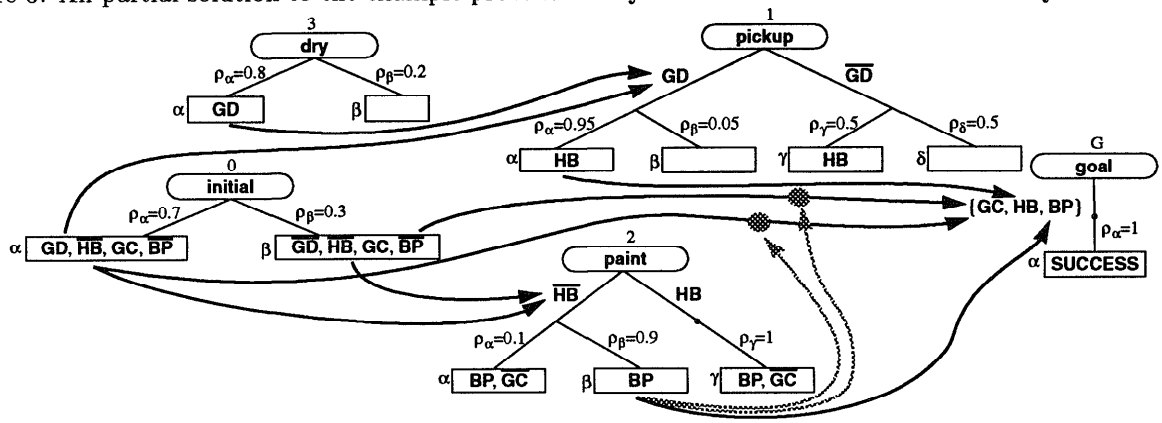

Figure 4: A plan that is sufficiently likely to succeed.

Efficient Plan Assessment

The FORWARD assessment strategy, while simple, can be quite inefficient, since the number of states with nonzero probability can grow exponentially with the length of the plan. This inefficiency motivates a second focus of our research, an exploration of alternative assessment algorithms. Since the general plan assessment problem is NP-hard (Chapman 1987; Cooper 1990) we cannot expect to produce an assessment algorithm that runs efficiently on every problem in every domain. However, by exploiting the structure of the actions, goals and state space, we can sometimes realize tremendous efficiency gains. We have implemented three alternative plan assessment algorithms.

While the size of the state distribution may grow exponentially, in general not all of the distinctions between the different states will be relevant to whether the goal is true. So one alternative assessment strategy, called QUERY, tries to divide states into subsets based on the truth of the goal. At best QUERY needs to reason about only two subsets of states: those in which the goal is true and those in which it is false.

Rather than manipulating states explicitly, assessment algorithms can reason directly about the propositions that comprise the states. Thus the third assessment algorithm, NETWORK, translates the action descriptions into a probabilistic network, each node of which corresponds to a single proposition at some point during the execution of the plan. NETWORK then solves this network using standard propagation techniques.

The resulting network tends to be more complicated than necessary, however, since it explicitly includes "persistence" links encoding the fact that a literal remains unchanged across the execution of an action that neither adds nor deletes it. But recall that persistence assumptions are already stored in a plan's causal-link structures: an unthreatened link is a guarantee that the link's proposition will persist from the time it is produced until it is consumed. This motivates a fourth algorithm, REVERSE, that computes a plan's success probability by manipulating the plan's causal link structure directly.

The longer version of this paper describes these algorithms in detail. Analysis of these algorithms shows no clear winner: in each case we can construct domains and problems in which one algorithm consistently outperforms the other.

Related Work and Conclusions

(Mansell 1993) and (Goldman & Boddy 1994) offer alternative approaches to applying classical planning algorithms to probabilistic action and state models.

(Dean *et al.* 1993), (Farley 1983), and (Koenig 1992) use fully observable Markov processes to model the planning problem. These systems generate an *execution policy*, a specification of what action the agent should perform in every world state, and assume that the agent is provided with complete and accurate information about the world state. BURIDAN produces a *plan*, a sequence of steps that is executed without examining the world at execution time, and assumes the agent will be provided with *no* additional information at execution time. A recent extension to BURIDAN, (Draper, Hanks, & Weld 1994a; 1994b), strikes a middle ground: the representation allows actions that provide possibly inaccurate information about the world at execution time, and actions in a plan can be executed contingent on the information provided by previous steps.

Work in decision science has dealt with planning problems (see (Dean & Wellman 1991, Chap. 7) for an introduction), but it has focused on solving a given probabilistic model whereas our algorithm interleaves the process of constructing and evaluating solutions. But see (Breese 1992) for recent work on model-building issues.

(Haddawy & Hanks 1992; 1993) motivate building a planner like BURIDAN, exploring the connection between building plans that probably satisfy goals and plans that are optimal in the sense of maximizing expected utility.

Although planning with conditional effects is not the primary focus of our work, BURIDAN also generalizes work on planning with deterministic conditional effects, *e.g.* in (Collins & Pryor 1992; Penberthy & Weld 1992). A deterministic form of confrontation is used in UCPOP (Penberthy & Weld 1992).

The BURIDAN planner integrates a probabilistic semantics for action with classical least-commitment planning techniques. BURIDAN accepts probabilistic information about the problem's initial state, and manipulates actions with conditional and probabilistic effects.

Our planner is fully implemented in Common Lisp and has been tested on many examples. BURIDAN takes about 4.5 seconds to find a solution to the problem presented in this paper. Send mail to *bug-buridan@cs.washington.edu* for information about BURIDAN source code.

References

Breese, J. 1992. Construction of belief and decision networks. *Computational Intelligence* 8(4).

Chapman, D. 1987. Planning for conjunctive goals. *Artificial Intelligence* 32(3):333–377.

Collins, G., and Pryor, L. 1992. Achieving the functionality of filter conditions in a partial order planner. In *Proc. 10th Nat. Conf. on A.I.*

Cooper, G. 1990. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence* 42.

Dean, T., and Wellman, M. 1991. *Planning and Control.* Morgan Kaufmann.

Dean, T., Kaelbling, L., Kirman, J., and Nicholson, A. 1993. Planning with deadlines in stochastic domains. In *Proc. 11th Nat. Conf. on A.I.*

Draper, D., Hanks, S., and Weld, D. 1994a. A probabilistic model of action for least-commitment planning with information gathering. In *Proc., Uncertainty in AI.* Submitted.

Draper, D., Hanks, S., and Weld, D. 1994b. Probabilistic planning with information gathering and contingent execution. In *Proc. 2nd Int. Conf. on A.I. Planning Systems.*

Farley, A. 1983. A Probabilistic Model for Uncertain Problem Solving. *IEEE Transactions on Systems, Man, and Cybernetics* 13(4).

Goldman, R. P., and Boddy, M. S. 1994. Epsilon-safe planning. forthcoming.

Haddawy, P., and Hanks, S. 1992. Representations for Decision-Theoretic Planning: Utility Functions for Dealine Goals. In *Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning.*

Haddawy, P., and Hanks, S. 1993. Utility Models for Goal-Directed Decision-Theoretic Planners. Technical Report 93–06–04, Univ. of Washington, Dept. of Computer Science and Engineering. Submitted to *Artificial Intelligence.* Available via FTP from `pub/ai/` at `cs.washington.edu`.

Hanks, S. 1990. Practical temporal projection. In *Proc. 8th Nat. Conf. on A.I.*, 158–163.

Hanks, S. 1993. Modeling a Dynamic and Uncertain World II: Action Representation and Plan Evaluation. Technical report, Univ. of Washington, Dept. of Computer Science and Engineering.

Koenig, S. 1992. Optimal probabilistic and decision-theoretic planning using markovian decision theory. UCB/CSD 92/685, Berkeley.

Kushmerick, N., Hanks, S., and Weld, D. 1993. An Algorithm for Probabilistic Planning. Technical Report 93-06-03, Univ. of Washington, Dept. of Computer Science and Engineering. To appear in *Artificial Intelligence.* Available via FTP from `pub/ai/` at `cs.washington.edu`.

Mansell, T. 1993. A method for planning given uncertain and incomplete information. In *Proc. 9th Conf. on Uncertainty in Artifical Intelligence.*

McAllester, D., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *Proc. 9th Nat. Conf. on A.I.*, 634–639.

Penberthy, J., and Weld, D. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning*, 103–114. Available via FTP from `pub/ai/` at `cs.washington.edu`.