An Algorithm for Probabilistic 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*

March 10, 1994

Technical Report 93-06-03

Abstract

We define the probabilistic planning problem in terms of a probability distribution over initial world states, a boolean combination of propositions representing the goal, 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.

In this paper, we present BURIDAN, an implemented least-commitment planner that solves problems of this form. We prove that the algorithm is both sound and complete. We then explore BURIDAN's efficiency by contrasting four algorithms for plan evaluation, using a combination of analytic methods and empirical experiments. We also describe the interplay between generating plans and evaluating them, and discuss the role of search control in probabilistic planning.

---

[*] We gratefully acknowledge the comments and suggestions of Tony Barrett, Tom Dean, Denise Draper, Mike Erdmann, Keith Golden, Rex Jacobovits, Oren Etzioni, Neal Lesh, Mike Wellman, Mike Williamson, and the anonymous reviewers. 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.

Contents

1 Introduction 1
1.1 Action representation 1
1.2 The planning algorithm 3
1.3 Discussion 4
1.4 Example 5
1.5 Alternative assessment algorithms 7
1.6 Contributions 8

2 A Semantics for Probabilistic Planning 9
2.1 States and expressions 9
2.2 Actions and action sequences 9
2.3 Planning problems and solutions 10
2.4 Extending the example 11

3 The BURIDAN Algorithm 13
3.1 Data structures 13
3.2 The BURIDAN algorithm: top-level 15
3.3 Plan refinement 15
3.4 Plan assessment 18

4 Formal Properties 22
4.1 Soundness 22
4.2 Completeness 23

5 Efficient Plan Assessment 24
5.1 The FORWARD assessment algorithm 24
5.2 The QUERY assessment algorithm 25
5.3 The NETWORK assessment algorithm 25
5.4 The REVERSE assessment algorithm 27
5.5 Empirical confirmation 28

6 The Assess-Refine Interface 34
6.1 Reasoning about partial orders 36
6.2 Search control 37
6.3 Summary 37

7 Related Work 39
7.1 Probabilistic planning 39
7.2 Robotic motion planning 40
7.3 Graphical decision models 40
7.4 Probabilistic temporal reasoning 41

7.5 Action representation and plan evaluation 41
7.6 Classical planning 41

8 Conclusions 42
8.1 Implementation 42
8.2 Summary 42
8.3 Future work 42

A Proof of Completeness 44

B The REVERSE Assessment Algorithm 48

1 Introduction

Classical planning assumes complete and deterministic information about the world state and the effects of actions. These assumptions are inappropriate for many domains: turning the ignition key might usually start one's old car, but occasionally fail for unknown reasons. Even if a deterministic model is possible for a given domain, it might be too complex to be useful. For example, when deciding between an indoor and an outdoor site for a wedding, one is likely to use a probabilistic model to forecast the weather rather than project the cloud dynamics. The initial world state is also a source of uncertainty: will the freeways be crowded?

This paper presents a planning algorithm that does not depend on the assumptions of complete and deterministic information. We use a probability distribution over possible world states to model imperfect information about the initial world state, and we model actions using a conditional probability distribution over changes to the world.

Adopting a probabilistic model complicates the definition of plan success. Instead of terminating when it builds a plan that *provably* achieves the goal, our planner terminates when it builds a plan that is *sufficiently likely* to succeed: our algorithm produces a plan such that the probability of the plan achieving the goal is no less than a user-supplied probability threshold, if such a plan exists.

The work reported here makes several contributions. First, we define a symbolic action representation and provide it probabilistic semantics. Second, we describe an implemented algorithm, BURIDAN,[1] for probabilistic planning. Third, we prove the planner both sound and complete. Fourth, we compare the efficiency of four different probabilistic assessment algorithms both analytically and with empirical experiments. Finally, we explore the interface between the process of generating plans and the process of evaluating them.

1.1 Action representation

Following [29], we extend the standard STRIPS [23] representation to allow conditional and probabilistic effects. In STRIPS, an action is "enabled" if its preconditions are satisfied when the action is executed, in which case the action has a deterministic effect. If the preconditions do not hold, the action is "disabled," and executing it is an error or meaningless. This simple model is not sufficient for representing actions with multiple possible consequences. BURIDAN models actions that can be executed in *any* world state, with the effect of executing the action depending on the execution-time state and on random chance.

Consider the following simple action from a robot planning domain. Suppose that a robot's grasping operation is not always successful. We model this action's effects as depending both on the state of the world at execution time and on random chance. Specifically, we model the uncertainty of this pickup action by describing it in terms of four *consequences*. In two of the consequences, the robot will be holding the block after executing the action, but in the other two the world state doesn't change. To each consequence we assign a probability which depends on the state of the world when the action is executed. For example, we might encode the fact that if the gripper is dry then the block is successfully grasped 95% of the time, but if the gripper is wet then the block is grasped only 50% of the time. Figure 1 shows our representation of the pickup action.

---

[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.

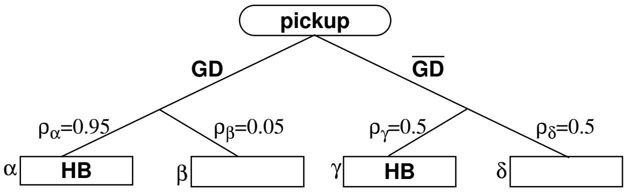


Figure 1: The pickup action. GD means "gripper dry;" HB means "holding block."

Propositions like GD and HB ("gripper dry" and "holding block") characterize the relevant part of the world's state. The ρ_ι encode the conditional probabilities that the corresponding consequence is realized when the action is executed. For example, $\rho_\alpha = 0.95$ indicates that consequence α is realized with probability 0.95 given that GD holds when the action is executed.

As shown in Figure 1, actions are encoded with binary trees. The leaves of the tree are the action's *effects*, the set of changes made to the world state if the corresponding trigger holds when the action is executed. The labels on the path from the root encode the consequence's *trigger*, a conjunction expressing the conditions under which this consequence occurs. For example, executing pickup when the gripper is dry (GD), would likely (probability .95) cause the robot to be holding the block (HB).[2] Like STRIPS's add- and delete-lists, consequences describe changes to the world state rather than entire states. As shown in the figure, we index an action's consequences with α, β, *etc.* The binary tree representation enforces the constraint that the triggers for all consequences of an action are mutually exclusive and exhaustive: exactly one will be realized during execution.

In classical planning, a world state is described with a set of propositions. Since BURIDAN's domains are probabilistic, we characterize the agent's knowledge of the world not as a single state but rather as a probability distribution over possible states. In the classical paradigm, actions cause a transition from one state to another; BURIDAN's actions induce a transition from one probability distribution to another.

Graphical depictions of actions like Figure 1 might give the mistaken impression that we are assigning probabilities directly to *propositions* in an action's consequences without regard to the state of the world at execution time. This is not the case: we are assigning probabilities to *possible world states* in which propositions are deterministically true or false. Section 2 provides a formal semantics for our action representation.

---

[2] Since no set of effects contains $\overline{\text{HB}}$, our simple model of robot grippers does not capture the phenomenon of dropping an already held block when attempting a pickup. Of course, it would be easy to elaborate our model to account for this phenomenon by introducing HB to the triggers of the action so that the effect of pickup depends on whether something is already held.

1.2 The planning algorithm

The job of a probabilistic planning algorithm is to construct a sequence of actions such that executing each action in turn, starting from some initial probability distribution over states, results in a final distribution in which the goal expression holds with sufficient probability, where sufficiency is defined with respect to a user-supplied probability threshold.

BURIDAN searches through a space of *plans* until it finds one that achieves the goal with sufficient probability. Each plan consists of a set of *actions*, a partial *temporal ordering relation* over the actions, a set of *causal links*, and a set of *subgoals* (each a proposition-action pair). The first two items are straightforward; the last two require some explanation.

A *causal link* [39] $\mathsf{A}_{i,\iota}\xrightarrow{\mathsf{p}}\mathsf{A}_j$ caches the planner's reasoning that proposition p could be true at the time action A_j (the link's *consumer*) is executed because consequence ι of action A_i (the link's *producer*) makes it true. The link is said to *provide causal support* for p. To realize this support, the planner must try to increase the probability that consequence ι of A_i is realized and prevent p from being made false by other actions. BURIDAN attempts the former by providing additional causal support to the triggers of A_i's consequence ι.

The final component of a plan—the set of subgoals—is used for this purpose. The idea behind these pairs is analogous to a goal agenda in a classical planner: if p is a subgoal for action A_j (written $\mathsf{p}@\mathsf{A}_j$), then BURIDAN seeks to increase the probability of p at the time that A_j is executed. For example, one way to increase this probability is to add to the plan a new action that makes p true. Whenever BURIDAN does this (suppose it adds A_i whose consequence ι makes p true), it records the decision with a causal link $\mathsf{A}_{i,\iota}\xrightarrow{\mathsf{p}}\mathsf{A}_j$. BURIDAN then makes each proposition in consequence ι's trigger a subgoal for A_i. When planning starts, the set of subgoals is initialized to the set of goal propositions tagged with a dummy action denoting the end of the plan. In summary, subgoals serve to focus BURIDAN's attention towards improvements to a plan that will tend to increase the probability of goal satisfaction.

We say that an action A_k *threatens* a causal link $\mathsf{A}_{i,\iota}\xrightarrow{\mathsf{p}}\mathsf{A}_j$ if some consequence of A_k asserts $\overline{\mathsf{p}}$ and if A_k might occur between A_i and A_j. Threatened links signify that BURIDAN's commitments might not be met, so the planner must take evasive action. For example, BURIDAN can try to constrain the threatening action so that it must be executed before A_i or after A_j, thereby eliminating the threat.

Planning starts with the *null plan*: a plan consisting of just two special actions initial and goal, with the constraint that initial be executed first and goal last (in the figures, we use the convention that time progresses from left to right). These special actions encode the probability distribution over initial world states and the goal expression, respectively. For example, consider a world with one block; suppose that initially the block is not held and the gripper is dry with probability 0.7. The initial action corresponding to this distribution is shown on the left side of Figure 2. Each consequence of initial describes one of these two possible world states. The agent's goal is encoded with a distinguished action goal. Suppose that the goal is to be holding the block, HB; the right side of Figure 2 shows the goal action for this goal expression. goal has a single consequence that produces SUCCESS, triggered by the goal expression.

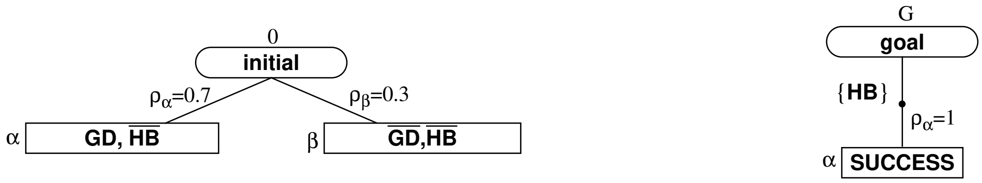


Figure 2: The null plan encodes the initial state distribution and the goal.

BURIDAN searches the space of partial plans, performing two operations at each visited node:

1. *Plan Assessment:* Determine whether the probability that the current plan achieves the goal exceeds the probability threshold, terminating successfully if so.
2. *Plan Refinement:* Otherwise, try to increase the probability of goal satisfaction by refining the current plan:
    - Nondeterministically choose a subgoal $p@A_j$, and add a causal link to A_j from some new or existing action A_i that can produce p, in an attempt to increase the probability of p when A_j is executed; or
    - Nondeterministically choose an existing causal link that is *threatened*, and resolve the threat.

    Signal failure if there are no possible refinements, otherwise continue by looping with the new partial plan.

In Section 1.4 we illustrate these two operations using the example described earlier, but first we discuss some significant differences between the BURIDAN planning algorithm and other least-commitment planners.

1.3 Discussion

Refining a plan with conditional and probabilistic operators differs from classical plan refinement (*e.g.* SNLP [39]) in three important ways.

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 $A_{i,\iota}\xrightarrow{p}A_j$ ensures that p will be true for action A_j only if trigger t^i_ι 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.

A final difference between classical planners and BURIDAN concerns the relationship between BURIDAN's subgoals and a classical planner's goal agenda [45]. In a classical planner, every entry on the agenda must be made true before the plan can be considered a solution, but in the case of a probabilistic planner this is no longer the case. Thus BURIDAN need not consider all subgoals to devise a plan that achieves its goal with sufficient probability. Indeed, if the threshold is zero, it need consider none at all!

1.4 Example

Recall the example: a robot whose gripper is possibly wet (with probability 0.3) needs to be holding a block. Alas, the pickup action is unreliable, especially when the gripper is wet. Suppose that there is also a dry action that usually (with probability 0.8) succeeds. We now illustrate the two steps of our planning algorithm with this example, assuming that a plan must be constructed that works 90% of the time. For expository purposes, we illustrate BURIDAN making the correct nondeterministic choices; in reality, considerable search is necessary to find a solution (see Section 6).

1.4.1 Plan refinement

Plan refinement starts with the null plan shown in Figure 2. Since HB is not true in any state in the initial distribution, BURIDAN adds an instance of the pickup action to the plan because this is the only action that can make HB true. BURIDAN creates a link from pickup's α consequence to the goal action. This link caches the planner's reasoning that HB will be true because pickup makes it true, as long as the conditions under which pickup produces HB are satisfied and no intermediate actions produce $\overline{\mathsf{HB}}$. BURIDAN must thus try to bring about the circumstances that cause pickup to produce HB. In general BURIDAN can not *guarantee* that an action has a particular consequence, but the planner can add further refinements to make the desired consequence *more likely.* In our example this means trying to make the gripper dry when pickup is executed. BURIDAN can make GD true in two ways. GD is true initially with probability 0.7, so the first option is to add a causal link from initial. As we shall show in the next section, this simple plan—consisting of a single pickup action—has probability 0.815, which is less than the probability threshold 0.9. So BURIDAN adds additional causal support for GD by inserting and linking from a dry action. Figure 3 shows the resulting plan. This plan achieves HB with probability 0.923, which exceeds the threshold of 0.9, so BURIDAN has successfully found a solution.

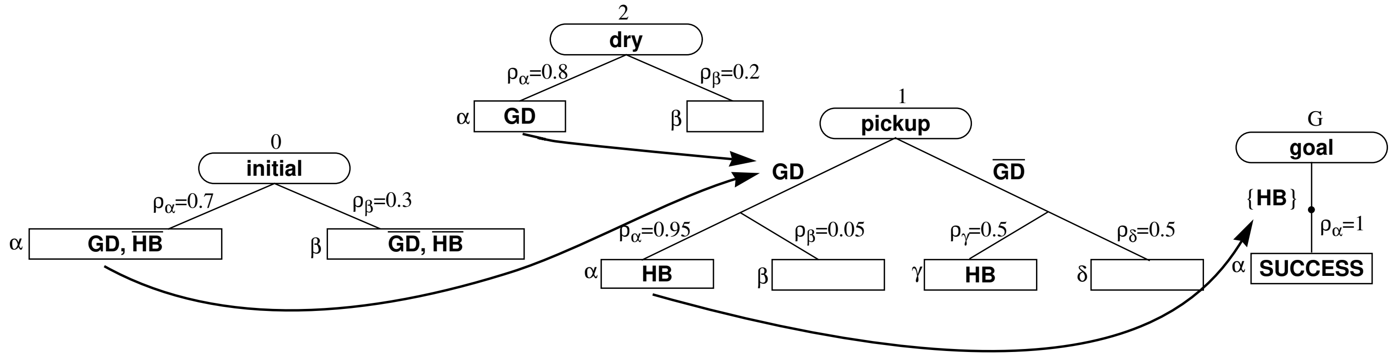


Figure 3: This plan for holding the block, HB, works at least 90% of the time despite the fact that gripper dryness, GD, is not guaranteed.

1.4.2 Plan assessment

We have implemented and analyzed four algorithms that compute the probability that a plan achieves its goal; Section 5 discusses the performance tradeoffs among them. Here we illustrate the simplest assessment strategy, FORWARD, which directly implements the definition of plan success implied above. Recall that executing an action induces a transition from one probability distribution over states to another. FORWARD takes an action sequence and "executes" each action in turn, and then computes the probability of the goal expression in the final distribution.[3]

Recall the simple one-action plan described above: picking up the block *without* first drying the gripper. The initial distribution consists of two states that differ only in whether GD holds: $\{\mathsf{GD}, \overline{\mathsf{HB}}\}$ and $\{\overline{\mathsf{GD}}, \overline{\mathsf{HB}}\}$ (for conciseness we write the state using set notation). The initial probability distribution over states is: $\mathrm{P}[\{\mathsf{GD}, \overline{\mathsf{HB}}\}] = 0.7$, $\mathrm{P}[\{\overline{\mathsf{GD}}, \overline{\mathsf{HB}}\}] = 0.3$. The probability distribution resulting from executing pickup in this initial distribution consists of the four states:

1. $\{\mathsf{GD}, \mathsf{HB}\}$ is the resulting state if the initial state is in fact $\{\mathsf{GD}, \overline{\mathsf{HB}}\}$ and the pickup action is "successful," *i.e.*, consequence of α of pickup is realized. (The other "successful" consequence, γ, can not happen in this initial state since one of its triggers, $\overline{\mathsf{GD}}$, is definitely false.) The probability of this new state is the probability of this initial state times the probability that pickup has consequence α *given* this initial state, $0.7 \times 0.95 = 0.665$.

---

[3] If the partially ordered plan is consistent with *multiple* total orders, then FORWARD performs the computation for each total order and returns the minimum.

2. $\{\mathsf{GD}, \overline{\mathsf{HB}}\}$ is realized if the initial state is $\{\mathsf{GD}, \overline{\mathsf{HB}}\}$ and pickup results in consequence β; this state has probability $0.7 \times 0.05 = 0.035$.

3. $\{\overline{\mathsf{GD}}, \mathsf{HB}\}$ is realized if the initial state is $\{\overline{\mathsf{GD}}, \overline{\mathsf{HB}}\}$ and pickup results in consequence γ; this state has probability $0.3 \times 0.5 = 0.15$.

4. $\{\overline{\mathsf{GD}}, \overline{\mathsf{HB}}\}$ is realized if the initial state is $\{\overline{\mathsf{GD}}, \overline{\mathsf{HB}}\}$ and pickup results in consequence δ; this state has probability $0.3 \times 0.5 = 0.15$.

Since pickup is the plan's only action, we now assess the goal HB with respect to this final state distribution. HB is true in the first and third states listed above, so the probability is the sum, 0.815, as reported earlier.

1.5 Alternative assessment algorithms

The FORWARD assessment strategy, while simple, can be quite inefficient. For example, there exist domains in which the number of states with non-zero probability grows exponentially with the length of the plan. This inefficiency motivates a second focus of our research, an investigation of alternative assessment algorithms. Since the general plan assessment problem is NP-hard [5, 8], we can not hope to produce an assessment algorithm that runs efficiently in every domain. However, by exploiting the structure of the actions, goals and state space, we can sometimes realize tremendous efficiency gains. For example, while the number of states with non-zero probability may grow exponentially in the size of the plan, in general not all of the distinctions between the different states will be relevant to the question of whether the goal proposition holds. So one alternative assessment strategy, called QUERY, limits the growth of the state distribution by distinguishing states based on the value of only the subset of propositions relevant to the goal conjunction.

Another class of algorithms reasons not about the actual state space but rather about the propositions that define the space. The insight is that while the number of states may grow exponentially, the number of propositions within a domain is constant. So a third assessment algorithm, which we call NETWORK, maps the actions to a network of probabilistic constraints over the propositions, and then solves these constraints directly. The resulting networks tend to be more complicated than they need to be, however. For example, the network contains arcs and nodes that encode the fact that the truth value of a proposition remains unchanged across an action that does not affect it. But note that the plan itself contains explicit information about persistence: causal links are essentially a cache for deductions about persistence. We thus define a fourth algorithm, REVERSE, that traverses the plan's causal link structure to do plan assessment. REVERSE and QUERY are also different from the other two assessment algorithms in an important way. FORWARD and NETWORK explicitly examine every totally ordered sequence of actions consistent with the plan (resulting in a large performance penalty), while REVERSE and QUERY can directly evaluate a partially ordered sequence of actions. We have implemented all four assessment algorithms; in Section 5 we present an analytical and empirical study of the tradeoffs among the various alternatives.

In Section 6 we consider the assessment algorithms in a larger context. We demonstrate that speed of assessment does not always correlate with planning speed because some assessors compute better bounds on the exact probability than others. Preliminary empirical results show that an improved interface between plan assessment and plan refinement can lead to significant speedup.

1.6 Contributions

This paper describes our implemented, provably correct probabilistic planning algorithm. We have tested it on many small examples, including the simple Slippery Gripper example just described, an extension of this example that will be used throughout this paper to describe our algorithm in detail, and the Bomb and Toilet example [41] (see Section 6). We make the following advances to the field of planning:

1. We define an expressive action representation for which we provide a probabilistic semantics (Section 2).
2. We describe BURIDAN, an implemented algorithm for probabilistic planning (Section 3).
3. We prove the planner both sound and complete (Section 4).
4. We compare the efficiency of four different probabilistic assessment algorithms both analytically and empirically (Section 5) and explore the relationship between the processes of plan refinement and plan assessment (Section 6).

2 A Semantics for Probabilistic Planning

The task of this section is to define a planning problem, and what it means to solve one. We begin by defining states and expressions, then actions and sequences of actions, and finally the planning problem and its solution.

2.1 States and expressions

A *state* is a complete description of the world at a single point in time. A state is described using a set of *propositions* in which every proposition appears exactly once, possibly negated.[4] Uncertainty about the world is represented using a random variable over states. 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

$$\mathrm{P}[\mathcal{E}\,|\,s] = \begin{cases} 1 & \text{if } \mathcal{E} \subseteq s \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

2.2 Actions and action sequences

Our model of action, taken from [29, 30, 28], combines a symbolic model of the changes the action makes to propositions with probabilistic parameters that represent chance (unmodeled) influences. Figure 1 is a 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, 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).

We can make this intuitive definition more precise as follows.

Definition 1 (Action) *An* action *is a set of* consequences $\{\langle \mathsf{t}_\alpha, \rho_\alpha, \mathsf{e}_\alpha\rangle, \ldots, \langle \mathsf{t}_\eta, \rho_\eta, \mathsf{e}_\eta\rangle\}$ *For each* ι, t_ι *is an expression called the consequence's* trigger, $0 \leq \rho_\iota \leq 1$, *and* e_ι *is a set of literals called the* effects. *The triggers must be mutually exclusive and exhaustive:*

$$\forall \iota \quad . \quad \sum_s \rho_\iota \mathrm{P}[\mathsf{t}_\iota\,|\,s] = 1 \tag{2}$$

$$\forall s, \iota, \kappa \quad . \quad \mathsf{t}_\iota \neq \mathsf{t}_\kappa \Rightarrow \mathrm{P}[\mathsf{t}_\iota \cup \mathsf{t}_\kappa\,|\,s] = 0 \tag{3}$$

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

---

[4] 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. Also see Section 5.

The representation for the pickup action is thus

$$\{\langle\{\mathsf{GD}\}, 0.95, \{\mathsf{HB}\}\rangle, \langle\{\mathsf{GD}\}, 0.05, \{\}\rangle, \langle\{\overline{\mathsf{GD}}\}, 0.5, \{\mathsf{HB}\}\rangle, \langle\{\overline{\mathsf{GD}}\}, 0.5, \{\}\rangle\}.$$

A consequence defines through its set of effects a (deterministic) transition from a state to a state, defined by a function RESULT, similar to add- and delete-lists in STRIPS.

Definition 2 (Change effected by a consequence) *Let s be a state and* e *be a set of literals. Then* $\textsc{result}(\mathsf{e}, s)$ *is defined as follows: for each proposition* p,

- *If* $\mathsf{p} \in \mathsf{e}$ *then* $\mathsf{p} \in \textsc{result}(\mathsf{e}, s)$ *and* $\overline{\mathsf{p}} \notin \textsc{result}(\mathsf{e}, s)$.
- *If* $\overline{\mathsf{p}} \in \mathsf{e}$ *then* $\overline{\mathsf{p}} \in \textsc{result}(\mathsf{e}, s)$ *and* $\mathsf{p} \notin \textsc{result}(\mathsf{e}, s)$.
- *Otherwise* $\mathsf{p} \in \textsc{result}(\mathsf{e}, s)$ *iff* $\mathsf{p} \in s$, *and* $\overline{\mathsf{p}} \in \textsc{result}(\mathsf{e}, s)$ *iff* $\overline{\mathsf{p}} \in s$.

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

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

Note that because an action's triggers are mutually exclusive and exhaustive we have that $\sum_{s'} \mathrm{P}[s' \,|\, s, \mathsf{A}] = 1$ for all states s and all actions A.

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:

$$\mathrm{P}[s' \,|\, s, \langle\rangle] = \begin{cases} 1 & \text{if } s' = s \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

$$\mathrm{P}\left[s' \,|\, s, \langle \mathsf{A}_i \rangle_{i=1}^N\right] = \sum_u \mathrm{P}[u \,|\, s, \mathsf{A}_1] \mathrm{P}\left[s' \,|\, u, \langle \mathsf{A}_i \rangle_{i=2}^N\right] \tag{6}$$

where $\langle \mathsf{A}_i \rangle_{i=j}^k = \langle\rangle$ if $j > k$.

Finally, we define the probability that an expression is true after an action sequence is executed beginning in some state, and the probability of an expression after executing an action sequence given an initial probability distribution over states:

$$\mathrm{P}\left[\mathcal{E} \,|\, s, \langle \mathsf{A}_i \rangle_{i=1}^N\right] = \sum_{s'} \mathrm{P}\left[s' \,|\, s, \langle \mathsf{A}_i \rangle_{i=1}^N\right] \mathrm{P}[\mathcal{E} \,|\, s'] \tag{7}$$

$$\mathrm{P}\left[\mathcal{E} \,|\, \tilde{\mathsf{s}}_I, \langle \mathsf{A}_i \rangle_{i=1}^N\right] = \sum_s \mathrm{P}\left[\mathcal{E} \,|\, s, \langle \mathsf{A}_i \rangle_{i=1}^N\right] \mathrm{P}[\tilde{\mathsf{s}}_I = s] \tag{8}$$

2.3 Planning problems and solutions

We have given a semantics for actions in probabilistic domains. So far this discussion has been quite detached from the use of this semantics in a planning context. We are now in a position to define the input-output behavior of a probabilistic planning algorithm. The input is a planning problem.

Definition 3 (Planning Problem) *A* planning problem *is a 4-tuple* $\langle \tilde{\mathsf{s}}_I, \mathcal{G}, \tau, \Lambda \rangle$*, where* $\tilde{\mathsf{s}}_I$ *is a random variable over states,* $\mathcal{G}$ *is a expression,* $0 \leq \tau \leq 1$*, and* Λ *is a set of actions.*

The intent is the following:

1. knowledge of the world at execution time is characterized by $\tilde{\mathsf{s}}_I$;
2. the goal to be achieved is an expression $\mathcal{G}$;
3. τ is the probability threshold for goal satisfaction; and
4. Λ is the set of actions from which solutions may be constructed.

Given these inputs, an algorithm for probabilistic planning must compute a totally ordered sequence of actions such that executing each action in turn induces a probability distribution over states in which the goal holds with probability no less than the threshold. Thus our final task is to give a precise meaning to the sentence "The action sequence $\langle \mathsf{A}_i \rangle_{i=1}^N$ is a solution to the planning problem $\langle \tilde{\mathsf{s}}_I, \mathcal{G}, \tau, \Lambda \rangle$." Intuitively, an action sequence is successful if the execution of the sequence achieves the goal with probability no less than the threshold.

Definition 4 (Solution) *Let* $\Delta = \langle \tilde{\mathsf{s}}_I, \mathcal{G}, \tau, \Lambda \rangle$ *be a planning problem, and* $\langle \mathsf{A}_i \rangle_{i=1}^N$ *be a (possibly empty) sequence of actions.* $\langle \mathsf{A}_i \rangle_{i=1}^N$ *is a* solution *to* Δ *iff each* $\mathsf{A}_i \in \Lambda$ *and*

$$\mathrm{P}\Big[\mathcal{G} \,|\, \tilde{\mathsf{s}}_I, \langle \mathsf{A}_i \rangle_{i=1}^N\Big] \geq \tau \tag{9}$$

2.4 Extending the example

The simple example described in Section 1.4 illustrates some aspects of our planning algorithm, but to fully describe BURIDAN it is helpful to consider an extended version.

Recall that we have defined two actions: pickup and dry are shown in Figures 1 and 3. Figure 4 illustrates a third action, paint, that paints the block (BP) but sometimes causes the gripper to become dirty ($\overline{\mathsf{GC}}$). For the problem's goal, we demand that in addition to holding the block (HB), the robot needs to have it painted (BP) as well, while keeping its gripper clean (GC).

To describe this example as a probabilistic planning problem we proceed as follows. Suppose that initially the block is not being held, the gripper is clean, the block is unpainted, and the gripper is dry with probability 0.7. Thus we have two initial states with non-zero probability, 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: $\mathrm{P}[\tilde{\mathsf{s}}_I = s_1] = 0.7$; $\mathrm{P}[\tilde{\mathsf{s}}_I = s_2] = 0.3$. The goal is straightforward: $\mathcal{G} = \{\mathsf{HB}, \mathsf{BP}, \mathsf{GC}\}$. If we are willing to consider plans with a twenty percent chance of failure, then we set the probability threshold $\tau = 0.8$. Finally, we want the solution built from the three actions defined above: $\Lambda = \{\mathsf{pickup}, \mathsf{paint}, \mathsf{dry}\}$. These four components together constitute the input to the planning algorithm.

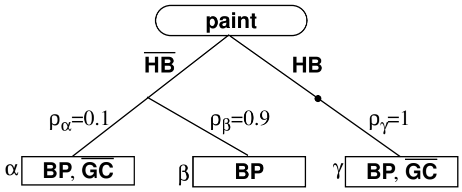

paint
$\overline{\text{HB}}$
HB
ρ_α=0.1
ρ_β=0.9
ρ_γ=1
α
BP, $\overline{\text{GC}}$
β
BP
γ
BP, $\overline{\text{GC}}$

Figure 4: The paint action. HB means "holding block;" BP means "block painted;" GC means "gripper clean."

3 The BURIDAN Algorithm

Given a planning problem, BURIDAN searches through a space of *partial plans*, terminating when it finds one corresponding to a solution. Each plan consists of a set of *actions*,[5] a partial *temporal ordering relation* "$<$" over the actions, a set of *causal links*, and a set of *subgoals*. There are significant differences between the last two items and the analogous SNLP concepts.

A causal 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 some consequence of A_k asserts $\overline{\mathsf{p}}$, and if A_k can be ordered between A_i and A_j.

A plan's set of subgoals consists of the literals in the plan that BURIDAN could try to make true at particular points in time. A subgoal is a literal annotated with a particular action, written $\mathsf{p}@\mathsf{A}_i$. BURIDAN adopts $\mathsf{p}@\mathsf{A}_i$ as a subgoal of a plan if A_i is the producer for some link in the plan, and p is a trigger of the producing consequence. More formally, BURIDAN adopts $\mathsf{p}@\mathsf{A}_i$ as a subgoal if $\mathsf{A}_{i,\iota}\xrightarrow{\mathsf{q}}\mathsf{A}_j$ is one of the plan's links and $\mathsf{p} \in \mathsf{t}_i^{\iota}$.[6] The set of subgoals is initialized to include all top-level goals. In Section 1.3 we discussed important differences between BURIDAN's notion of a subgoal and the set of open conditions in classical planners such as SNLP.

BURIDAN searches for a solution by performing two operations at each node in the space of plans:

1. *Plan Assessment:* Compute the probability that the current plan will achieve the goal. If the probability is high enough, then the plan is a solution, and planning terminates successfully.

2. *Plan Refinement:* Otherwise, try to increase the probability of goal satisfaction by refining the current plan. Each refinement generates a new partial plan. Signal failure if there are no possible refinements, otherwise nondeterministically choose a new partial plan, and loop.

We describe each of these operations below, but first we describe the details of BURIDAN's representations.

3.1 Data structures

As we mentioned above, a causal link caches the planner's reasoning that a particular proposition could be made true for an action because some consequence of a particular action makes it true.

---

[5] The set actually contains action *instances* because a plan may have more than one instance of a particular action. But since our representation is propositional this distinction is unimportant, and we use the term "action" to refer both to actions and instances.

[6] Subgoals are also used to implement *confrontation*; see Section 3.3.

Definition 5 (Causal Link) *A* causal link *is a 4-tuple* $\langle \mathsf{A}_i, \iota, \mathsf{p}, \mathsf{A}_j \rangle$*, written* $\mathsf{A}_{i,\iota} \xrightarrow{\mathsf{p}} \mathsf{A}_j$*. Consequence* ι *of action* A_i *is the link's* producer*;* A_j *is the link's* consumer*;* p *is the proposition* supported *by the link.*

Each node in the space BURIDAN searches is a plan.

Definition 6 (Plan) *A* plan *is 4-tuple* $\langle \mathcal{A}, \mathcal{O}, \mathcal{L}, \mathcal{S} \rangle$*, where* $\mathcal{A}$ *is a set of actions,* $\mathcal{O}$ *is a set of temporal ordering constraints over* $\mathcal{A}$ *(each element of which is of the form* $\mathsf{A}_i < \mathsf{A}_j$*),* $\mathcal{L}$ *is a set of causal links, and* $\mathcal{S}$ *is a set of* subgoals *(each of the form* $\mathsf{p}@\mathsf{A}_i$*).*

An action threatens a link when executing the action between the link's producer and its consumer might decrease the probability of the proposition supported by the link.

Definition 7 (Threat) *Let* $\langle \mathcal{A}, \mathcal{O}, \mathcal{L}, \mathcal{S} \rangle$ *be a plan, and let* $\mathsf{A}_{i,\iota} \xrightarrow{\mathsf{p}} \mathsf{A}_j \in \mathcal{L}$ *be one of the plan's links and* $\mathsf{A}_k \in \mathcal{A}$ *be one of the plan's actions.* A_k threatens $\mathsf{A}_{i,\iota} \xrightarrow{\mathsf{p}} \mathsf{A}_j$ *iff* $\mathsf{A}_i < \mathsf{A}_k < \mathsf{A}_j$ *is consistent with* $\mathcal{O}$*, and if there is some consequence* κ *of* A_k *such that* $\overline{\mathsf{p}} \in \mathsf{e}_\kappa^k$*.*

BURIDAN encodes the initial probability distribution over states with a distinguished action initial. Each consequence of initial encodes one possible initial state, and the probability associated with the consequence encodes the state's probability in the initial distribution $\tilde{\mathsf{s}}_I$.

Definition 8 (Action corresponding to a probability distribution over states) *Let* $\tilde{\mathsf{s}}_I$ *be a random variable encoding a probability distribution over states. The* action corresponding to $\tilde{\mathsf{s}}_I$ *is* $\textsc{initial}(\tilde{\mathsf{s}}_I) = \{\ldots, \langle \{\}, \rho_\iota, \mathsf{e}_\iota \rangle, \ldots\}$ *such that for each* ι*,* $\rho_\iota = \mathrm{P}[\tilde{\mathsf{s}}_I = \mathsf{e}_\iota]$*.*

It is straightforward to prove that $\textsc{initial}(\tilde{\mathsf{s}}_I)$ satisfies the formal definition of an action.

The goal is also encoded with a special action goal. It has a distinguished consequence marked SUCCESS and it is triggered by the goal expression.

Definition 9 (Action corresponding to a goal) *Let* $\mathcal{G} = \{\mathsf{p}_1, \ldots, \mathsf{p}_n\}$ *be an expression. The* action corresponding to $\mathcal{G}$ *is* $\textsc{goal}(\mathcal{G}) = \{\langle \mathcal{G}, 1, \{\mathsf{SUCCESS}\} \rangle\}$*.*

Note that this definition of $\textsc{goal}(\mathcal{G})$ does not satisfy the formal definition of an action, but it is straightforward to construct a more complicated definition that does satisfy the definition.[7]

Recall the example from Section 2.4: initially the block is neither painted nor held ($\overline{\mathsf{BP}}$ and $\overline{\mathsf{HB}}$) and the gripper is certainly clean (GC), but there is only partial information about whether the gripper is dry; in one state (with probability 0.7) GD holds, but in the other it does not. Recall also that the goal is have the block painted, the gripper clean, and block held: $\mathcal{G} = \{\mathsf{BP}, \mathsf{GC}, \mathsf{HB}\}$. Figure 5 shows the initial and goal actions for this planning problem.

---

[7] A formally correct goal action has $\eta = |\mathcal{G}| + 1$ consequences defined as follows: each probability term is 1.0, and the first trigger is the negation of the first goal literal, the second trigger is the first goal literal and the negation of the second, ..., the $|\mathcal{G}|$'th trigger is the first $|\mathcal{G}| - 1$ goal literals and the negation of the $|\mathcal{G}|$'th, and the η'th trigger is the entire goal expression.

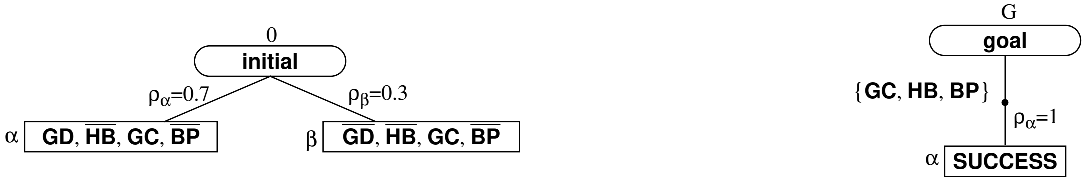

0
initial
ρ_α=0.7
ρ_β=0.3
α
GD, $\overline{\text{HB}}$, GC, $\overline{\text{BP}}$
β
$\overline{\text{GD}}$, $\overline{\text{HB}}$, GC, $\overline{\text{BP}}$
G
goal
{GC, HB, BP}
ρ_α=1
α
SUCCESS

Figure 5: initial and goal encode the initial probability distribution over states and the goal.

Now we can describe the root node of the space of plans, from which BURIDAN starts searching for a solution.

Definition 10 (Null plan) *Let* $\langle \tilde{\mathsf{s}}_I, \mathcal{G}, \tau, \Lambda \rangle$ *be a planning problem.* NULL-PLAN$(\langle \tilde{\mathsf{s}}_I, \mathcal{G}, \tau, \Lambda \rangle) = \langle \mathcal{A}, \mathcal{O}, \mathcal{L}, \mathcal{S} \rangle$ *is a plan constructed from* $\langle \tilde{\mathsf{s}}_I, \mathcal{G}, \tau, \Lambda \rangle$ *as follows:*

$$\begin{aligned}
\mathcal{A} &= \{\mathsf{A}_0, \mathsf{A}_G\} \\
\mathsf{A}_0 &= \mathsf{initial} \quad = \quad \text{INITIAL}(\tilde{\mathsf{s}}_I) \\
\mathsf{A}_G &= \mathsf{goal} \quad = \quad \text{GOAL}(\mathcal{G}) \\
\mathcal{O} &= \{\mathsf{A}_0 < \mathsf{A}_G\} \\
\mathcal{L} &= \emptyset \\
\mathcal{S} &= \{\mathsf{p}@\mathsf{A}_G \mid \mathsf{p} \in \mathcal{G}\}
\end{aligned}$$

A_0 must be the first action in any plan and A_G must be the last; the ordering constraints of the null plan enforce this invariant. To preserve this constraint, similar ordering constraints are added when each new action is introduced into all refinements of this plan. The set of subgoals is initialized to the set of goal propositions annotated with A_G, the time just after all planned actions have been executed.

3.2 The BURIDAN algorithm: top-level

We are now in a position to describe the BURIDAN algorithm; see Table 1. Given a planning problem $\langle \tilde{\mathsf{s}}_I, \mathcal{G}, \tau, \Lambda \rangle$, BURIDAN first converts the problem to the corresponding null plan. The null plan is then iteratively refined by calling the REFINE subroutine. REFINE nondeterministically chooses one possible refinement; if none are available then the planning problem has no solution and BURIDAN terminates. When the assessment algorithm (ASSESS) determines that the plan's probability of success is no less than τ, an arbitrary totally ordered sequence of the plan's actions is returned. Section 3.3 explains plan refinement in detail; Section 3.4 describes assessment.

3.3 Plan refinement

BURIDAN's plan refinement procedure (Table 2) considers all possible successors of a particular plan, and nondeterministically chooses one to return. There are two ways to refine a

```
BURIDAN($\langle \tilde{s}_I, \mathcal{G}, \tau, \Lambda \rangle$)
  1.  $\langle \mathcal{A}, \mathcal{O}, \mathcal{L}, \mathcal{S} \rangle \leftarrow$ NULL-PLAN($\langle \tilde{s}_I, \mathcal{G}, \tau, \Lambda \rangle$)
  2.  Do forever
          If ASSESS($\langle \mathcal{A}, \mathcal{O}, \mathcal{L}, \mathcal{S} \rangle$) $\geq \tau$, then
  a.          Return TOTAL-ORDER($\mathcal{A} - \{A_0, A_G\}, \mathcal{O}$)
          else
  b.          $\langle \mathcal{A}, \mathcal{O}, \mathcal{L}, \mathcal{S} \rangle \leftarrow$ REFINE($\langle \mathcal{A}, \mathcal{O}, \mathcal{L}, \mathcal{S} \rangle, \Lambda$)
              If REFINE signalled failure, then
  c.              Signal failure
```

Table 1: The BURIDAN algorithm: top-level.

plan: either by resolving a *threat* to a causal link, or by adding a new link to (potentially) increase the probability that a subgoal proposition holds when its annotating action is executed. BURIDAN stores the set of subgoals in the $\mathcal{S}$ component of each plan, but the set of threatened links is easily computed dynamically from the plan's links and actions.

```
REFINE($\langle \mathcal{A}, \mathcal{O}, \mathcal{L}, \mathcal{S} \rangle$,$\Lambda$)
  1.  Choose FLAW from $\mathcal{S}$ or the set of threatened links.
  2.  If FLAW is $p@A_j$ then add support:
      a.  Nondeterministically add a new action $A_i$ from $\Lambda$ to $\mathcal{A}$ (also adding
          $A_0 < A_i < A_G$ to $\mathcal{O}$), or choose an existing $A_i$ from $\mathcal{A}$ such that $A_i$
          has a consequence $\iota$ (chosen nondeterministically) that asserts $p$.
      b.  Add $A_{i,\iota} \xrightarrow{p} A_j$ to $\mathcal{L}$ and $A_i < A_j$ to $\mathcal{O}$.
      c.  Add $q@A_i$ to $\mathcal{S}$, for each $q \in t^i_\iota$.
      d.  Signal failure if none of these options are possible for the current plan.
  3.  If FLAW is a threat to $A_{i,\iota} \xrightarrow{p} A_j$ by $A_k$, then nondeterministically choose:
      a.  Demotion: constrain $A_k < A_i$, or
      b.  Promotion: constrain $A_j < A_k$, or
      c.  Confrontation: commit to consequences of $A_k$ that do not make $p$ false:
          i.   Create a new safety proposition $s$,
          ii.  Modify $A_k$ so that its non-interfering consequences produce $s$,
          iii. Add $s@A_j$ to $\mathcal{S}$.
      d.  Signal failure if none of these options are possible for the current plan.
  4.  Return the resulting plan.
```

Table 2: The REFINE algorithm.

Link creation in BURIDAN is similar to the corresponding refinement in SNLP or UCPOP, but there are some important differences. We have already remarked that it may be desirable to add two independent actions to make a proposition true and that doing so will result in two causal links supporting the proposition. A planner that does not allow actions with disjunctive effects need not consider multiple causal support, although it may choose to do so for efficiency reasons [33]. The whole notion of causal support is more complex in the probabilistic case. For example, linking to a new action, even if it does not involve threats, may *not* increase the probability of goal satisfaction. Suppose that consequences from two different actions are used to support the same subgoal. If the triggers of the two supporting consequences are identical and are supported from the same source, then they are probabilistically dependent. In this case the support they lend is not additive. For example, suppose that turning the ignition key always starts the car just in case its battery is charged, but the agent doesn't know whether the battery is healthy or dead. In this case, is clearly doesn't help to turn the key more than once. This explains why BURIDAN (unlike SNLP or UCPOP) needs a separate assessment routine — no "local" computation (*i.e.*, one that does not look at the causal structure of the entire plan) suffices to determine plan success. Said another way, whereas in the classical paradigm causal links can eliminate the need for dynamically computing the Modal Truth Criterion, this can not be avoided in the probabilistic case.

Link promotion and demotion are identical to threat resolution in classical planners, so we do not discuss them further.[8] Confrontation is a significant departure from SNLP, however, and it deserves some additional explanation. The probability that link $\mathsf{A}_{i,\iota}\xrightarrow{\mathsf{p}}\mathsf{A}_j$ succeeds in producing p for A_j is the probability that executing action A_i actually results in consequence ι *and* that no action between A_i and A_j makes p false. Since BURIDAN need only produce a plan that succeeds with probability no less than τ, it might be acceptable to allow a threatening action to remain between the link's producing and consuming actions as long as it makes p false with sufficiently low probability. *Confrontation* resolves a threat in exactly this manner. Confrontation involves noting which consequences of the threatening action do not pose a threat to the link (the *non-interfering* consequences), and attempting to increase the probability that one of these consequences is realized when the threatening action is executed. Specifically, BURIDAN confronts a threat by modifying the threatening action so that each non-interfering consequence produces a newly created proposition s, unique to the threat, called a *safety* proposition. The safety proposition, annotated with the consumer of the link, is then adopted as an additional subgoal. Since only the non-interfering consequences of A_k can produce s, planning for the safety condition amounts to planning to make a non-interfering consequence of the threatening action occur. Of course, if an action has *no* non-interfering consequences then confrontation is inappropriate; the algorithm must in this case either promote or demote the threat instead.

Example: We now demonstrate how the plan refinement algorithm constructs a plan that will succeed with probability at least 0.8 in satisfying its goal to be holding a painted block with a clean gripper. As in the example of Section 1, we simplify the presentation by assuming that REFINE makes the correct sequence of nondeterministic choices; as discussed in Section 6, BURIDAN takes about 4.5 seconds to find a solution with brute-force search.

1. Planning starts with the null plan, shown in Figure 5. The subgoals for this plan are the goal propositions annotated with the goal action: $\mathcal{S} = \{\mathsf{HB}@\mathsf{A}_G, \mathsf{BP}@\mathsf{A}_G, \mathsf{GC}@\mathsf{A}_G\}$. BURIDAN chooses to support the first subgoal, $\mathsf{HB}@\mathsf{A}_G$, by adding an instance of the **pickup** action, A_1, and linking to this action's α consequence with the link $\mathsf{A}_{1,\alpha}\xrightarrow{\mathsf{HB}}\mathsf{A}_G$. BURIDAN supports the desired consequence α of A_1 by adopting $\mathsf{GD}@\mathsf{A}_1$ as a subgoal. Support for this subgoal is then provided by linking directly to the initial action A_0 with the link $\mathsf{A}_{0,\alpha}\xrightarrow{\mathsf{GD}}\mathsf{A}_1$. The resulting plan is shown in Figure 6. The assessor determines that this plan is inadequate, so refinement continues.

---

[8] We ignore *separation* (the addition of variable-binding constraints [5]) since BURIDAN is propositional. Lifting techniques [39] could be used to extend BURIDAN to a more expressive language.

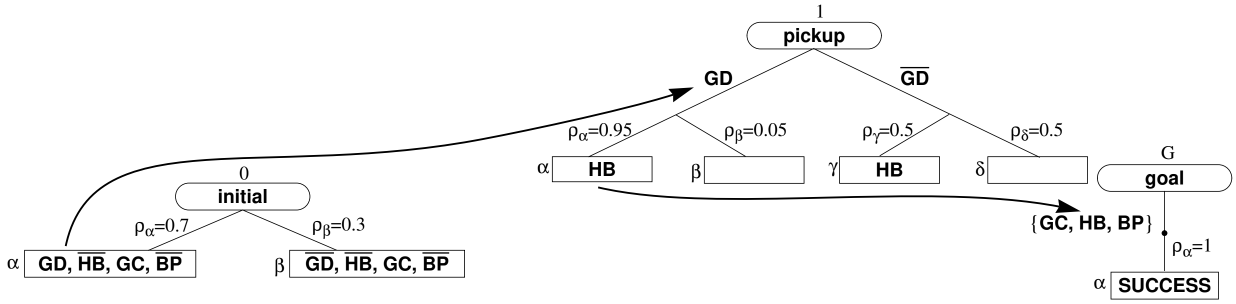


Figure 6: Support for holding the block.

2. BURIDAN next supports the subgoal of having the block painted, $\mathsf{BP}@\mathsf{A}_G$, by adding a new paint action, A_2, and adding the link $\mathsf{A}_{2,\beta}\xrightarrow{\mathsf{BP}}\mathsf{A}_G$. Consequence β of paint is realized only if the block is not held. The planner thus adopts $\overline{\mathsf{HB}}@\mathsf{A}_2$ as a subgoal, support for which is added with a link from initial: $\mathsf{A}_{0,\alpha}\xrightarrow{\overline{\mathsf{HB}}}\mathsf{A}_2$. pickup and paint are unordered, so pickup threatens this new link: if pickup is executed before paint then the block will be held when paint is executed, violating the $\overline{\mathsf{HB}}$ trigger of paint's consequence β. BURIDAN resolves this threat by promoting pickup with the constraint $\mathsf{A}_2 < \mathsf{A}_1$.

BURIDAN then supports the goal of having a clean gripper by providing support to $\mathsf{GC}@\mathsf{A}_G$ with the link $\mathsf{A}_{0,\alpha}\xrightarrow{\mathsf{GC}}\mathsf{A}_G$. But paint threatens this link: paint asserts $\overline{\mathsf{GC}}$ if either consequence α or consequence γ is realized. The only option is confrontation, which involves adding a new safety proposition s_1 to e_β^2, the only consequence of A_2 that does not cause $\overline{\mathsf{GC}}$. The resulting plan is shown in Figure 7. The gray circle on the link indicates that the threat has been resolved by confrontation.

3. Figure 7's plan has probability 0.7335 (as computed in Section 3.4). This value is less than $\tau = 0.8$ so BURIDAN tries to make the goal more probable by drying the gripper before trying to pick up the block. This is done by adding a new dry action, A_3, and adding the link $\mathsf{A}_{3,\alpha}\xrightarrow{\mathsf{GD}}\mathsf{A}_1$. BURIDAN also adds additional support for the goal of having the gripper clean, $\mathsf{GC}@\mathsf{A}_G$, by linking from initial with $\mathsf{A}_{0,\beta}\xrightarrow{\mathsf{GC}}\mathsf{A}_G$. This link is threatened by A_2 just as $\mathsf{A}_{0,\alpha}\xrightarrow{\mathsf{GC}}\mathsf{A}_G$ was, and the threat is resolved in the same way, by confrontation; the safety proposition for this threat is s_2. The resulting plan is shown in Figure 8.

This plan has a probability of 0.831, which exceeds τ, so BURIDAN has succeeded in finding a solution.

3.4 Plan assessment

The plan assessment algorithm decides whether the probability of success for a plan exceeds the threshold τ. Soundness demands only that the assessor never incorrectly identify a plan

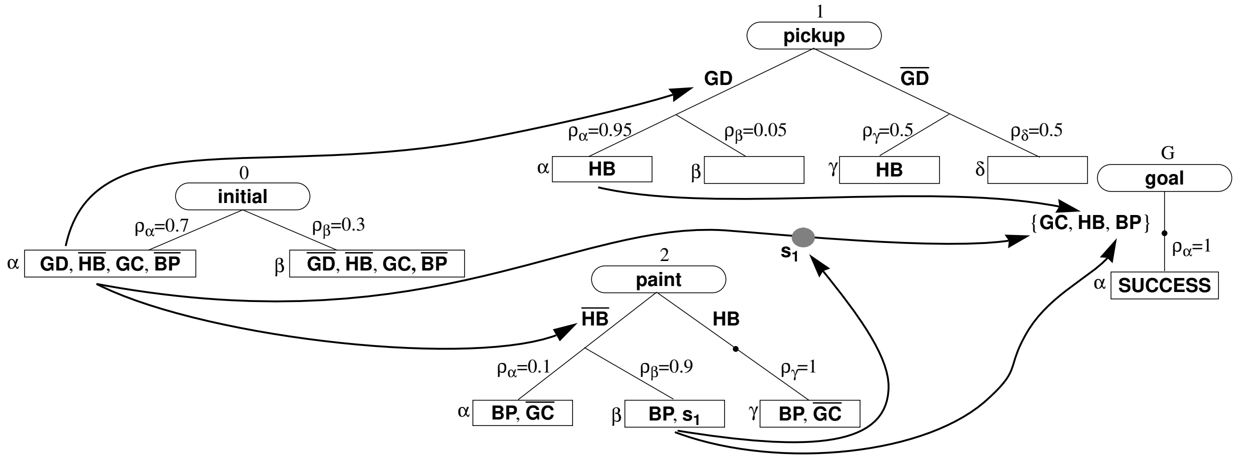

1
pickup
GD
$\overline{GD}$
ρα=0.95
ρβ=0.05
ργ=0.5
ρδ=0.5
α HB
β
γ HB
δ
G
goal
0
initial
ρα=0.7
ρβ=0.3
{$\overline{GC}$, HB, BP}
α GD, $\overline{HB}$, GC, $\overline{BP}$
β $\overline{GD}$, $\overline{HB}$, GC, $\overline{BP}$
s1
ρα=1
α SUCCESS
2
paint
$\overline{HB}$
HB
ρα=0.1
ρβ=0.9
ργ=1
α BP, $\overline{GC}$
β BP, s1
γ BP, $\overline{GC}$

Figure 7: Support for painting the block while keeping the gripper clean.

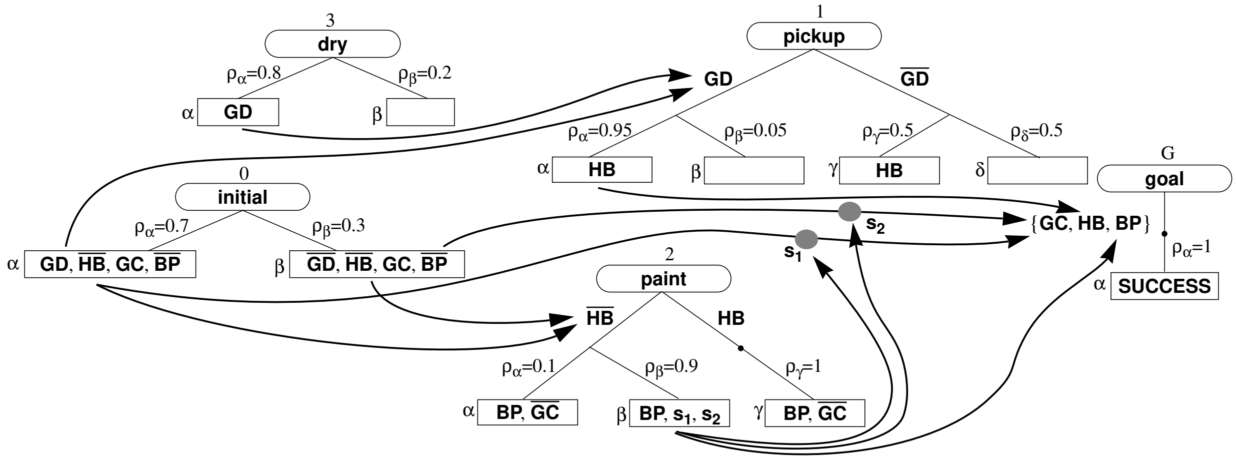

3
dry
ρα=0.8
ρβ=0.2
α GD
β
1
pickup
GD
$\overline{GD}$
ρα=0.95
ρβ=0.05
ργ=0.5
ρδ=0.5
α HB
β
γ HB
δ
G
goal
0
initial
ρα=0.7
ρβ=0.3
s2
{$\overline{GC}$, HB, BP}
α GD, $\overline{HB}$, GC, $\overline{BP}$
β $\overline{GD}$, $\overline{HB}$, GC, $\overline{BP}$
s1
ρα=1
α SUCCESS
2
paint
$\overline{HB}$
HB
ρα=0.1
ρβ=0.9
ργ=1
α BP, $\overline{GC}$
β BP, s1, s2
γ BP, $\overline{GC}$

Figure 8: Additional support for a dry gripper.

as a solution—that is, that it never identify a plan as a solution when in fact the plan's success probability is less than τ. The algorithms we implemented are somewhat more general, computing a lower bound on the exact probability of success. In this section we describe only the FORWARD assessment algorithm, a straightforward implementation of the definition of a solution to a planning problem (Definition 4); in Section 5 we describe three different algorithms.

Table 3 presents the FORWARD algorithm. To explain the algorithm, we first define the data structure it uses to represent a probability distribution over states.

Definition 11 (State distribution) *A* state distribution $\mathcal{SD} = \{\langle s_1, \rho_1\rangle, \ldots, \langle s_n, \rho_n\rangle\}$ *is a set of pairs such that each s_i is a state, $0 \le \rho_i \le 1$, and*

$$\sum_{\langle s_i, \rho_i\rangle \in \mathcal{SD}} \rho_i = 1 \qquad (10)$$

For example, the state distribution corresponding to the probability distribution over initial world states in the example is the set $\{\langle\{\mathsf{GD}, \overline{\mathsf{HB}}, \mathsf{GC}, \overline{\mathsf{BP}}\}, 0.7\rangle, \langle\{\overline{\mathsf{GD}}, \overline{\mathsf{HB}}, \mathsf{GC}, \overline{\mathsf{BP}}\}, 0.3\rangle\}$. Formally, an action induces a transition from one probability distribution over states to another. FORWARD uses the EXEC function to represent this transition in terms of state distributions.

Definition 12 (Execution of an action in a state distribution) *Let* $\mathcal{SD} = \{\langle s_1, \rho_1\rangle, \ldots, \langle s_n, \rho_n\rangle\}$ *be a state distribution and* $\mathsf{A} = \{\langle \mathsf{t}_\alpha, \rho_\alpha, \mathsf{e}_\alpha\rangle, \ldots, \langle \mathsf{t}_\eta, \rho_\eta, \mathsf{e}_\eta\rangle\}$ *be an action. Then the* execution of A in $\mathcal{SD}$ *is the set:*

$$\text{EXEC}(\mathsf{A}, \mathcal{SD}) = \left\{ \begin{array}{ccc} \langle s_{1,\alpha}, \rho_{1,\alpha}\rangle, & \ldots & , \langle s_{1,\eta}, \rho_{1,\eta}\rangle, \\ & \vdots & \\ \langle s_{n,\alpha}, \rho_{n,\alpha}\rangle, & \ldots & , \langle s_{n,\eta}, \rho_{n,\eta}\rangle \end{array} \right\} \tag{11}$$

where

$$\langle s_{i,\iota}, \rho_{i,\iota}\rangle = \langle \text{RESULT}(\mathsf{e}_\iota, s_i), \rho_i \rho_\iota \mathrm{P}[\mathsf{t}_\iota \,|\, s_i]\rangle \tag{12}$$

It is straightforward to prove that $\text{EXEC}(\mathsf{A}, \mathcal{SD})$ is a state distribution. Finally, FORWARD needs to evaluate the probability of an expression in a state distribution:

$$\mathrm{P}[\mathcal{E} \,|\, \mathcal{SD}] = \sum_{\langle s_i, \rho_i\rangle \in \mathcal{SD}} \rho_i \, \mathrm{P}[\mathcal{E} \,|\, s_i] \tag{13}$$

FORWARD uses Equations 11 and 12 to compute the successive state distributions that result while projecting the effects of a plan's actions. For efficiency, FORWARD prunes zero-probability states and combines members of the state distribution that refer to the same state. After all actions have been projected, the goal expression is evaluated using Equation 13.

Complicating the assessment process is the fact that a solution is defined in terms of a totally ordered sequence of actions, while 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.[9]

Example: We now illustrate how the FORWARD plan assessment algorithm computes the probability of success for the plan shown in Figure 7. This plan involves first painting the block and then picking it up.

The initial state distribution $\mathcal{SD}_0$ consists of two states with non-zero probability. They differ only in whether or not the gripper is dry:

$$\mathcal{SD}_0 = \{\langle\{\mathsf{GD}, \overline{\mathsf{HB}}, \mathsf{GC}, \overline{\mathsf{BP}}\}, 0.7\rangle, \langle\{\overline{\mathsf{GD}}, \overline{\mathsf{HB}}, \mathsf{GC}, \overline{\mathsf{BP}}\}, 0.3\rangle\}$$

---

[9] Section 6.1 considers the possibility of computing the *maximum* over all total orders, corresponding to the best probability that could be expected from *any* consistent total order.

```
ASSESS($\mathcal{P}$)
    1. Return FORWARD($\mathcal{P}$)
```

```
FORWARD($\langle \mathcal{A}, \mathcal{O}, \mathcal{L}, \mathcal{S} \rangle$)
    1. Let $\mathcal{SD}$ be the state distribution corresponding to $\mathsf{A}_0 \in \mathcal{A}$.
    2. Let $\mathcal{G}$ be $\mathrm{t}_1^G$ (the trigger of $\mathsf{A}_G$'s SUCCESS consequence).
    3. SCENARIOS $\leftarrow$ SET-OF-ALL-TOTAL-ORDERS($\mathcal{A} - \{\mathsf{A}_0, \mathsf{A}_G\}, \mathcal{O}$).
    4. If SCENARIOS $= \emptyset$ then MIN $\leftarrow 0$ else MIN $\leftarrow 1$.
    5. For each sequence $\langle \mathsf{A}_i \rangle_{i=1}^{N} \in$ SCENARIOS, use Equations 11 and 12 to calculate
           $\mathrm{EXEC}(\mathsf{A}_N, \mathrm{EXEC}(\mathsf{A}_{N-1}, \cdots \mathrm{EXEC}(\mathsf{A}_1, \mathcal{SD}) \cdots)),$
       the state distribution resulting from executing each action in turn. For efficiency, "compress" the
       intermediate distributions by eliminating states with zero probability and merging identical states.
    6. Use Equation 13 to compute the probability of $\mathcal{G}$ in the final state distribution; update MIN whenever
       the sum is lower.
    7. Return MIN.
```

Table 3: FORWARD is the simplest of our four plan assessment algorithms.

Projecting the effects of paint in $\mathcal{SD}_0$ results in a state distribution with four elements:

$$\mathcal{SD}_1 = \mathrm{EXEC}(\mathsf{paint}, \mathcal{SD}_0) = \left\{ \begin{array}{c} \langle\{\mathsf{BP}, \overline{\mathsf{GC}}, \overline{\mathsf{HB}}, \overline{\mathsf{GD}}\}, 0.03\rangle, \\ \langle\{\mathsf{BP}, \mathsf{GC}, \overline{\mathsf{HB}}, \overline{\mathsf{GD}}\}, 0.27\rangle, \\ \langle\{\mathsf{BP}, \overline{\mathsf{GC}}, \overline{\mathsf{HB}}, \mathsf{GD}\}, 0.07\rangle, \\ \langle\{\mathsf{BP}, \mathsf{GC}, \overline{\mathsf{HB}}, \mathsf{GD}\}, 0.63\rangle \end{array} \right\}$$

We then project pickup:

$$\mathcal{SD}_2 = \mathrm{EXEC}(\mathsf{pickup}, \mathcal{SD}_1) = \left\{ \begin{array}{c} \langle\{\mathsf{GD}, \overline{\mathsf{HB}}, \mathsf{GC}, \mathsf{BP}\}, 0.0315\rangle, \\ \langle\{\mathsf{GD}, \mathsf{HB}, \mathsf{GC}, \mathsf{BP}\}, 0.5985\rangle, \\ \langle\{\mathsf{GD}, \overline{\mathsf{HB}}, \overline{\mathsf{GC}}, \mathsf{BP}\}, 0.0035\rangle, \\ \langle\{\mathsf{GD}, \mathsf{HB}, \overline{\mathsf{GC}}, \mathsf{BP}\}, 0.0665\rangle, \\ \langle\{\overline{\mathsf{GD}}, \overline{\mathsf{HB}}, \mathsf{GC}, \mathsf{BP}\}, 0.135\rangle, \\ \langle\{\overline{\mathsf{GD}}, \mathsf{HB}, \mathsf{GC}, \mathsf{BP}\}, 0.135\rangle, \\ \langle\{\overline{\mathsf{GD}}, \overline{\mathsf{HB}}, \overline{\mathsf{GC}}, \mathsf{BP}\}, 0.015\rangle, \\ \langle\{\overline{\mathsf{GD}}, \mathsf{HB}, \overline{\mathsf{GC}}, \mathsf{BP}\}, 0.015\rangle \end{array} \right\} \begin{array}{c} \\ \leftarrow \\ \\ \\ \\ \leftarrow \\ \\ \\ \end{array}$$

Finally, the goal expression is evaluated with respect to this state distribution. Since the goal holds in 2 of the 8 states (those marked in the previous equation), we get

$$\mathrm{P}[\{\mathsf{BP}, \mathsf{GC}, \mathsf{HB}\} \,|\, \mathcal{SD}_2] = 0.5985 + 0.135 = 0.7335$$

which is less than the threshold $\tau = 0.8$, so as described in the previous section, planning continues until producing the plan shown in Figure 8.

4 Formal Properties

We now prove that the BURIDAN planning algorithm is sound and complete. We say that a probabilistic planner is *sound* if it never returns an action sequence whose chance of success is less than the threshold τ demands. We call such a planner *complete* if it finds such a sequence whenever a solution exists. Note that this does not require the planner to recognize futility when no solution exists and that BURIDAN could loop in this case.[10]

As we shall see, the explicit correspondence between the ASSESS and FORWARD algorithms and the underlying semantics makes the proof of soundness quite straightforward.

Proving completeness is more difficult — we need to establish two things: (i) that every action sequence leading to a solution is eventually considered by the planner, and (ii) that every solution passed to the assessor is recognized as a solution. The key to the first point is showing that a plan's set of subgoals identifies all the refinements that might increase the probability of achieving the goal. To establish the second point we start by observing that since BURIDAN's assessor implements Definition 4 directly, it calculates by definition the exact probability of any totally ordered sequence of actions. The trick is to establish that BURIDAN will add enough ordering constraints to raise a plan's minimum probability (taken over its total orders) over the threshold if it is possible to do so.

First we need to reconcile the notation introduced when we characterized a planning problem and solution formally with the data structures manipulated by the planner (in particular the assessment algorithm). The FORWARD assessor manipulates a data structure called a *state distribution*, which is a set of pairs of the form $\langle s_i, \rho_i \rangle$. There is an obvious equivalence between a state distribution and a probability distribution over states $\tilde{\mathsf{s}}$: $\langle s_i, \rho_i \rangle \in \mathcal{SD}$ is equivalent to $\mathrm{P}[\tilde{\mathsf{s}} = s_i] = \rho_i$.

Similarly the formal exposition represented actions as conditional probabilities of the form $\mathrm{P}[s' \,|\, s, \mathsf{A}]$ whereas FORWARD uses a function $\mathrm{EXEC}(\mathsf{A}, \mathcal{SD})$ that produces a state distribution. Once again the equivalence should be clear: if $\mathcal{SD}$ and $\tilde{\mathsf{s}}$ are equivalent, then $\langle s_i, \rho_i \rangle \in \mathrm{EXEC}(\mathsf{A}, \mathcal{SD})$ just in case $\rho_i = \sum_{s'} \mathrm{P}[s_i \,|\, s', \mathsf{A}] \mathrm{P}[\tilde{\mathsf{s}} = s']$

In the following proofs we mix the two notations (random variables over states and state distributions; conditional probabilities defining action execution and the EXEC function).

4.1 Soundness

We define soundness in terms of Definition 4. A planner is sound if it never returns a plan that is not a solution.

Theorem 1 (Soundness) *Let* $\Delta = \langle \tilde{\mathsf{s}}_I, \mathcal{G}, \tau, \Lambda \rangle$ *be a planning problem. If* BURIDAN(Δ) *returns the action sequence* $\langle \mathsf{A}_i \rangle_{i=1}^{N}$*, then* $\langle \mathsf{A}_i \rangle_{i=1}^{N}$ *is a solution to* Δ.

---

[10] It is not clear at this point whether the problem BURIDAN solves is fully decidable. On the one hand, results on classical planning with infinite state spaces [5, 19] do not apply because BURIDAN's state space is propositional and does not allow functions (thus is finite). On the other hand, the fact that BURIDAN must search through a space of *probability distributions* over states (which is infinite) complicates the problem.

Proof: BURIDAN's refinement and assessment algorithms make this proof straightforward. Note that BURIDAN can exit its infinite loop in only two ways. One of these exits (Line 2.c of Table 1) signals failure; since this return does not produce an action sequence, it does not satisfy the antecedent of the theorem and need not be considered. The other exit (Line 2.a) returns an action sequence consistent with the plan's partial order $\mathcal{O}$ only when the probability assessment produced by FORWARD is at least τ. Since FORWARD computes the minimum probability of achieving the goal taken over *all* total orders consistent with $\mathcal{O}$, the estimate returned by FORWARD is guaranteed to be *no higher* than the actual probability that $\mathcal{G}$ will be achieved by $\langle \mathsf{A}_i \rangle_{i=1}^{N}$. Transitivity ensures that $\langle \mathsf{A}_i \rangle_{i=1}^{N}$ is a solution and BURIDAN is sound. □

4.2 Completeness

There are several possible definitions of completeness. For example, one might require that the planner return *all* action sequences that achieve the goal with probability greater than the threshold. This definition is silly, however, because it requires the planner to augment a solution with irrelevant actions.

Our definition sidesteps the problem of irrelevant actions in the plan: we require that the planner find all *essential* solutions. An essential solution is an action sequence that is itself a solution but which fails to be a solution when any of its actions are removed. (The fact that an action sequence is essential does not mean that it is the *shortest* possible solution; there might be a completely different and much shorter sequence that also achieves the goal.)

Theorem 2 (Completeness) *Let $\Delta = \langle \tilde{\mathsf{s}}_I, \mathcal{G}, \tau, \Lambda \rangle$ be a planning problem and let $\langle \mathsf{A}_i \rangle_{i=1}^{N}$ be an essential solution (*i.e.*, no proper subsequence is also solution) of Δ. Then there exists a sequence of nondeterministic choices such that* BURIDAN(Δ) *will return $\langle \mathsf{A}_i \rangle_{i=1}^{N}$.*

We prove completeness by induction on the N, length of the essential solution. In the base case (a zero-length plan) we show that the algorithm correctly recognizes the case where the initial state satisfies the goal with sufficient probability. We then make the inductive hypothesis that the algorithm will find all essential plans of length less than N. The difficulty is in showing how the ability to generate $N-1$ step plans bears on a problem, Δ, whose essential solution has N steps. We do this by constructing a *modified* planning problem which *can* be solved in $N-1$ steps. Since the proof's details are complex, we relegate them to Appendix A.

5 Efficient Plan Assessment

Our plan refinement algorithm calls a plan assessment algorithm as a subroutine; that algorithm must compute the probability that the totally ordered completions of a partially ordered plan achieve the goal expression, or at least provide a lower bound on that probability. This section examines plan assessment in isolation.

Assessing an arbitrary partially ordered plan with conditional effects is NP-hard even when all probabilities are zero or one [5, 10]. While REFINE doesn't generate *arbitrary* partial orders, we believe that the additional complexity of probabilistic computations [8] can make assessment (even of a totally ordered plan) require time that is exponential in the length of the plan — the computation might require considering all combinations of consequences of every action in the plan.

So our aspirations are not to produce an algorithm that works efficiently for all planning problems; instead we present four alternative algorithms and demonstrate when each does and does not perform well. Future work might attempt to integrate the best aspects of these approaches given the expected characteristics of the domain in question. Studying assessment in isolation gives us insight into why various algorithms work or don't work, but the study is not an end unto itself. We are ultimately interested in how long it takes to generate a complete solution, not the per-plan time for refinement or assessment. As we show in Section 6, the fastest assessment algorithm does not necessarily lead to the fastest planner. By returning a better bound on the plan's success probability, a slower assessor can speed the overall planning process considerably.

5.1 The FORWARD assessment algorithm

We begin by describing the computational problems with the FORWARD algorithm described in Section 3.4. FORWARD is a straightforward implementation of our action semantics. Two features of the algorithm are important for this discussion. First, FORWARD projects each action through a state distribution, producing a new distribution. And second, since plan success is defined only for a totally ordered sequences of actions, FORWARD computes the success probability for every totally ordered sequence consistent with the input partial order, and returns the minimum.

Each of these features can lead to computational problems. First, there is a potential explosion in the size of the state distributions that are manipulated: if the original state distribution has M members, each action has η consequences, and the plan contains N actions, assessing even a single total order can generate a state distribution containing as many as $M\eta^N$ states.[11] The second computational problem concerns partially ordered actions: there are as many as $N!$ consistent total orders of the actions in an N-action plan, and the basic algorithm has to be applied once for each total order.

---

[11] Of course, if there are D propositions, then the state distribution can never have more than 2^D distinct states. However, if the initial distribution has a small number of states with non-zero probability, then executing each action can lead to growth that is exponential in the number of actions.

In summary, FORWARD might make distinctions among states in the state space, and among orderings among the total orders, that are irrelevant to whether the goal is achieved. These inefficiencies can lead to degraded performance; our second algorithm, QUERY, is designed to overcome both problems.

5.2 The QUERY assessment algorithm

The QUERY assessor is an adaptation of Hanks's [28, 30] projection algorithm which actually applies to a richer action representation than BURIDAN's, including continuous quantities and sets, and conditional (branching) plan execution. QUERY is goal directed—it tries to articulate the state space only when doing so is necessary to decide the state of a query proposition. The basic idea is to divide an action's consequences into equivalence classes based on how they affect the query proposition, and reason about the classes instead of about the individual consequences. For example, consider an action A with several consequences. Suppose that each of these consequences makes a goal proposition G true, but they differ on the changes they make to other propositions. If G is all that is relevant to plan success, QUERY will consider A to have a single consequence class that makes G true and is realized with probability one. We omit the details of the QUERY algorithm from this paper; see instead [28, 30].

5.3 The NETWORK assessment algorithm

FORWARD and QUERY are similar in that they represent world states explicitly: both manipulate structures that represent elements of the state space. An alternative is to dispense with an explicit representation of a state and represent its component propositions directly. Instead of reasoning about actions as transformations from state distributions to state distributions we instead reason about the circumstances under which an action makes a proposition true, makes it false, or leaves it unchanged.

This strategy suggests using a belief network [44] for assessment whose is similar to that proposed by Dean and Kanazawa [13] and Hanks [31]. Figure 9 shows such a network for a domain with propositions $\{\mathsf{p}_1, \ldots, \mathsf{p}_m\}$ and a plan with action sequence $\langle \mathsf{A}_i \rangle_{i=1}^{N}$.[12]

The graph consists of two types of nodes. First there is a node for each action: the node for A_i takes a value from the set $\{\alpha, \ldots, \eta\}$, where A_i has η consequences; the value ι represents the case where consequence ι of A_i is realized. Second, there is a "layer" of binary-valued nodes representing the propositions evaluated just after executing each action.

The nodes for the propositions just after A_i point to the action node for A_{i+1} if the proposition is one of A_{i+1}'s triggers. The action node then has an arc to the proposition nodes that it might affect. The state of every proposition at one stage also affects the state

---

[12] For simplicity, in Figure 9 we assume that each action potentially affects every proposition and *vice versa*, and that the goal expression mentions every proposition. The implementation of NETWORK adds arcs only between nodes that influence one another.

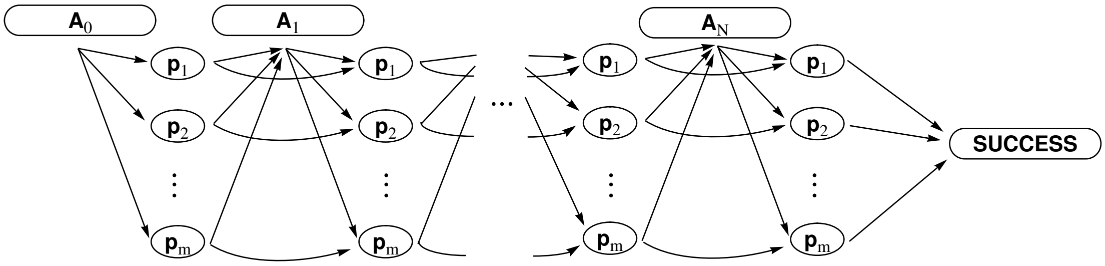


Figure 9: A probabilistic network for plan assessment.

of the same proposition at the next stage. Finally, there is a binary SUCCESS node that is true exactly when all propositions in the goal expression hold.

We may solve this network of constraints using standard propagation techniques [44], provided we supply appropriate numeric parameters for the model: a conditional probability table ("link matrix") indicating the probability that the node will take on one of its values, conditioned on the states of all the arcs that point to it. Specifically, NETWORK constructs a network such that:

- Action A_i will realize consequence ι with probability ρ^i_ι if that consequence's trigger propositions all hold.
- A proposition p is true after action A_i realizes consequence ι in one of two cases: either $\mathsf{p} \in \mathsf{e}^i_\iota$, or p was true before A_i was executed and $\overline{\mathsf{p}} \notin \mathsf{e}^i_\iota$. Note that the node for p just after the execution of A_i has two incoming arcs: one from the node representing A_i, indicating which consequence was realized, and the other indicating p's state immediately prior to A_i.
- The SUCCESS node is a conjunction of the propositions in $\mathcal{G}$ evaluated after execution of the last action.

Each of these boolean functions can easily be coded into an appropriate link matrix.

Section 5.5 reports on an implementation of NETWORK that uses the IDEAL [55] influence-diagram processor, using the Jensen clustering algorithm. We consistently see assessment time growing exponentially with the size of the plan.[13] One possible reason for this poor performance is that the causal network associated with a plan actually contains a lot of structure that is irrelevant to the goal, and this causes the propagation algorithm to do unnecessary work. For example, suppose that a goal proposition is actually made true by initial, and no action in the plan changes that proposition under any circumstances. The causal network must nonetheless propagate that persistence information through every stage of the plan's execution, computing the probability of every trigger of every action in the process.

---

[13] We are not promoting this method as the best candidate for solving the network—that is a topic for future research. Dean and Kanazawa [13] argue that a stochastic simulation technique might be more suitable, but also point out the absence of convergence bounds for these algorithms. Without a guarantee of convergence, our plan refinement algorithm is no longer sound nor complete.

This analysis suggests that a causal network could be built that eliminates this irrelevant structure; the REVERSE algorithm does just that.

5.4 The REVERSE assessment algorithm

The REVERSE assessment algorithm is based on the insight that the plan's causal link structure captures the information needed for assessment. Consider the causal link $\mathsf{A}_{i,\iota}\xrightarrow{\mathsf{p}}\mathsf{A}_j$. This link records the information that consequence ι of A_i makes p true, and that no intervening action makes p false as long as confronted threats to the link do not actually result in an interfering consequence. Note that p could also become true for some other reason; for example, some intervening action might make p true even though no link in the plan records this causal relationship. Thus directly examining a plan's causal structure yields *sufficient* conditions for goal satisfaction, and the assessed probability is a lower bound on the true probability of success.

Another important feature of REVERSE is that because the link structure guides assessment rather than just the plan's actions, REVERSE reasons directly about plans with partially ordered actions, unlike FORWARD and NETWORK which explicitly reason about every consistent total order. See also Section 6.1.

REVERSE uses a plan's causal link structure to construct an *assessment expression*, a boolean combination of terms that refer to earlier parts of the plan.[14] The idea is that a consequence's (conjunctive) trigger is true if *every* component literal is true, and a single literal is true if *any* of the incoming (disjunctive) links make the literal true. Initially the assessment expression is the trigger of the goal's SUCCESS consequence. The expression is then incrementally transformed by traversing the causal link structure according to the following rules:

- The assessment expression for a trigger is the *conjunction* of the assessment expressions of the subgoals corresponding to the trigger's conjuncts.
- The assessment expression for a subgoal is the *disjunction* of the assessment expressions for all the links supporting the subgoal.
- The assessment expression for a link is the assessment expression of the trigger for the link's producing outcome, conjoined with a conjunction of the assessment expressions of the subgoals of the safety condition associated with confronted threats.

The assessment expression is transformed using these rules until no more transformations are applicable. The probability of this expression can be then be computed directly. See Appendix B for a complete discussion of REVERSE.

---

[14] Note that this is *not* an "expression" in the sense of Section 2.1: an assessment expression can be an arbitrary boolean formula, and the terms in the formula are not propositions.

5.5 Empirical confirmation

So far we have motivated and described four plan assessment algorithms. We described one algorithm that directly implements the definition of success probability, and hinted at potential computational problems; we used these problems to suggest three alternative algorithms. Now we analyze in detail the performance differences between these algorithms.

We have built three illustrative domains, each intended to produce different behavior in FORWARD, QUERY, and REVERSE. Each domain involves an initial action, a goal action, and a "template" for defining additional actions. By varying one aspect of the domain (*e.g.* the goal or the probability threshold) we can vary the number of actions required to solve the problem. For each domain, we analyze the time taken by each algorithm to assess the solution plan, as a function of plan length.

5.5.1 A domain favoring FORWARD

One might think that since FORWARD and QUERY are exploring the same state space—the first blindly and the second in a manner sensitive to the query—that QUERY would always outperform FORWARD. The domain shown in Figure 10(a) shows this is false: in this domain FORWARD requires time linear in the length of the plan, while QUERY and REVERSE require time exponential in plan length. (NETWORK performs poorly in all of our domains, as discussed in Sections 5.3 and 5.5.4.)

Action A_i makes the goal proposition G true with probability 0.5 and makes propositions $\mathsf{p}_1, \ldots, \mathsf{p}_i$ false with probability 1.0. A successful N-action plan is of the form $\langle \mathsf{A}_i \rangle_{i=1}^{N}$; this sequence makes G true with probability $1 - \frac{1}{2^N}$. We therefore can vary the threshold τ to change the length of the plan: $\tau = 0$ requires a zero-action plan, $\tau = 0.5$ requires a one-action plan, and so on.

Figure 10(b) shows that FORWARD will project the N-action plan without a proliferation of states; FORWARD does well in this domain because after each action in the plan there are only two states with non-zero probability.

QUERY will have trouble with this domain because of a heuristic it uses when deciding what parts of the tree to make explicit: it considers actions from latest to earliest in deciding what consequence classes to build. Consider the case $N = 2$, so $\tau = 0.75$ and the successful plan has actions A_1 and A_2 in that order. When QUERY is asked to assess the probability of G it first considers A_2 and decides to split the action into two classes that differ on their effect on G. The first class consists of just the α consequence, and the second class is $\{\beta, \gamma\}$. At this point there are two possible completions to the plan, both with probability 0.5. G is true in the first but its value could be either true or false in the second, so the bound on G's probability is $[0.5, 1.0]$, which is ambiguous with respect to the threshold. QUERY next considers A_1 and splits its consequences into the same two classes, $\{\alpha\}$ and $\{\beta, \gamma\}$. Now there are four completions to the plan, each with probability 0.25, and G is true in three of them. So now the bound on G's probability is $[0.75, 0.75]$, and QUERY terminates. If QUERY had divided A_1 into classes first it would have realized that the three consequences of A_2 have the same effect on G's state in the α branch of the tree (*i.e.* given that G was already

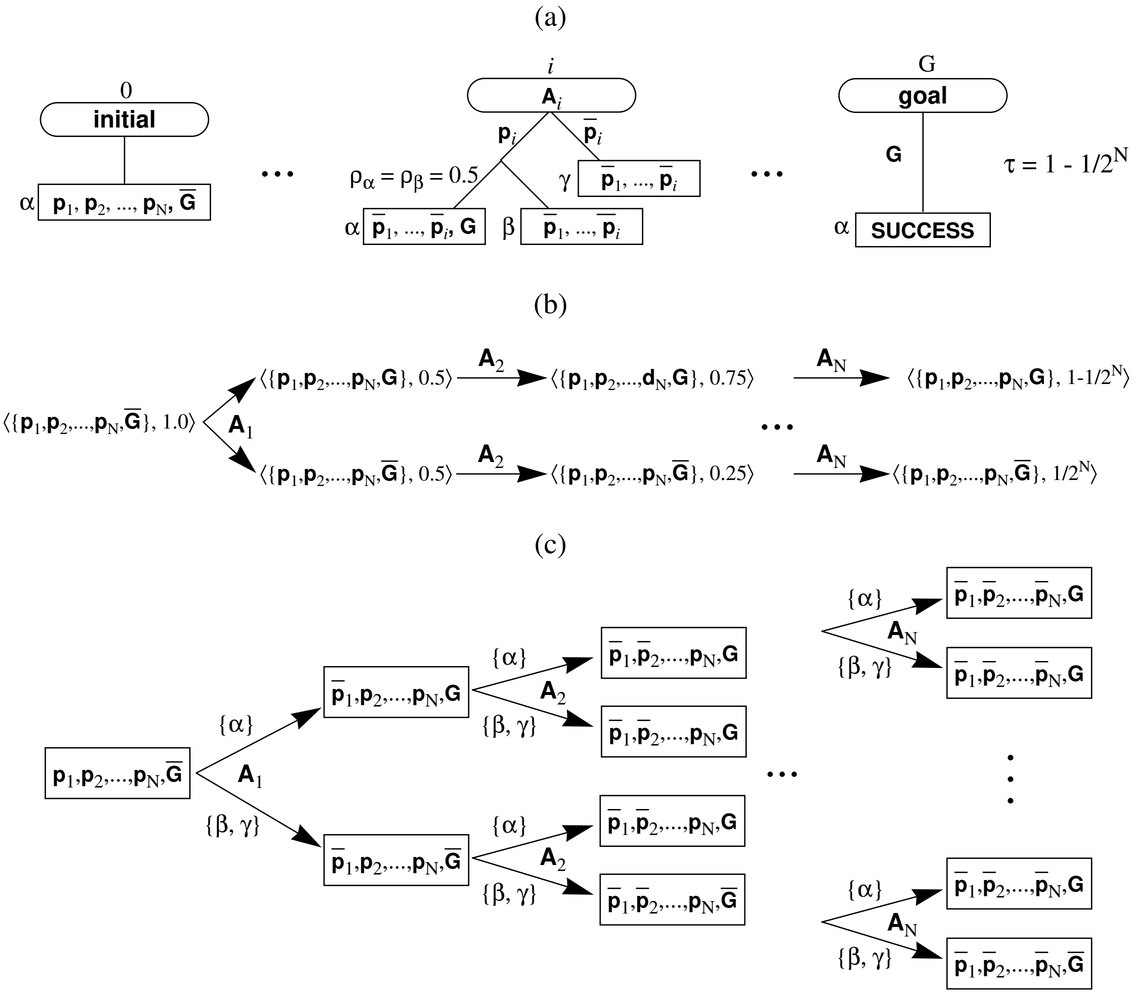

(a)
0
initial
α $p_1, p_2, ..., p_N, \overline{G}$
i
A_i
p_i
$\overline{p}_i$
$\rho_\alpha = \rho_\beta = 0.5$
γ $\overline{p}_1, ..., \overline{p}_i$
α $\overline{p}_1, ..., \overline{p}_i, G$
β $\overline{p}_1, ..., \overline{p}_i$
G
goal
G
$\tau = 1 - 1/2^N$
α SUCCESS
(b)
$\langle\{p_1,p_2,...,p_N,\overline{G}\}, 1.0\rangle$
A_1
$\langle\{p_1,p_2,...,p_N,G\}, 0.5\rangle$
A_2
$\langle\{p_1,p_2,...,d_N,G\}, 0.75\rangle$
A_N
$\langle\{p_1,p_2,...,p_N,G\}, 1\text{-}1/2^N\rangle$
$\langle\{p_1,p_2,...,p_N,\overline{G}\}, 0.5\rangle$
A_2
$\langle\{p_1,p_2,...,p_N,\overline{G}\}, 0.25\rangle$
A_N
$\langle\{p_1,p_2,...,p_N,\overline{G}\}, 1/2^N\rangle$
(c)
$p_1,p_2,...,p_N,\overline{G}$
A_1
$\{\alpha\}$
$\{\beta, \gamma\}$
$\overline{p}_1,p_2,...,p_N,G$
$\overline{p}_1,p_2,...,p_N,\overline{G}$
A_2
$\{\alpha\}$
$\{\beta, \gamma\}$
$\overline{p}_1,\overline{p}_2,...,p_N,G$
$\overline{p}_1,\overline{p}_2,...,p_N,G$
$\overline{p}_1,\overline{p}_2,...,p_N,G$
$\overline{p}_1,\overline{p}_2,...,p_N,\overline{G}$
A_N
$\{\alpha\}$
$\{\beta, \gamma\}$
$\overline{p}_1,\overline{p}_2,...,\overline{p}_N,G$
$\overline{p}_1,\overline{p}_2,...,\overline{p}_N,G$
$\overline{p}_1,\overline{p}_2,...,\overline{p}_N,G$
$\overline{p}_1,\overline{p}_2,...,\overline{p}_N,\overline{G}$

Figure 10: The domain shown in (a) is efficient for FORWARD because after projecting each action there are only two states with non-zero probability, as shown in (b); (c) illustrates how QUERY's tree branches unnecessarily at each action.

true). In that case the size of the tree would grow by only one completion for each new action in the plan. But no matter which predefined order is chosen for splitting, there will exist examples that are pathological for that heuristic.

REVERSE's performance will degrade because each additional action in the plan adds an additional link to the goal proposition, which adds another disjunct to the assessment expression, and as we describe in Appendix B, REVERSE takes time that is exponential in the number of disjuncts.

Figure 11 shows performance statistics for the domain shown in Figure 10, measuring the average time to perform a single assessment as the function of the number of actions in the plan.[15] As expected, FORWARD's assessment time grows linearly with plan size; the other algorithms required time exponential in the number of actions in the plan.

---

[15] All experiments were performed on a Sun SPARC-IPX. Since the inter-run variation was negligible, the

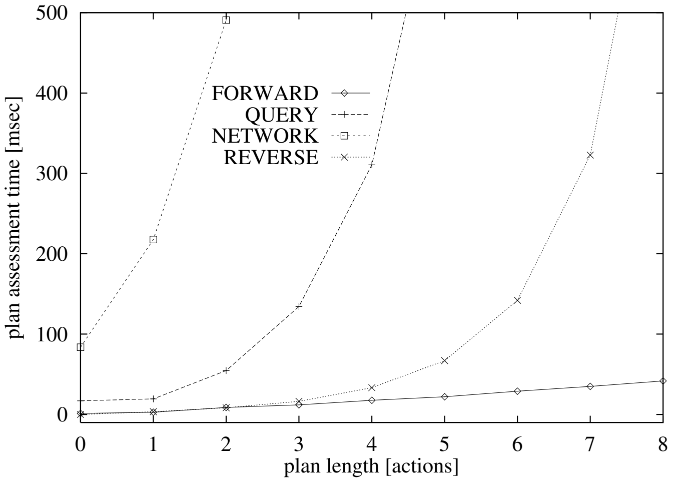


Figure 11: Average assessment time for plans in FORWARD's domain.

5.5.2 A domain favoring QUERY

Figure 12(a) shows a domain favorable to the QUERY algorithm. We vary the length of a solution plan by varying the number of conjuncts in the goal: a goal of the form $\{\mathsf{p}_1, \ldots, \mathsf{p}_N\}$ requires a plan with N actions.

Each action A_i has two consequences, but the distinction among the consequences is irrelevant to whether the plan satisfies the goal. None of the other algorithms recognize this feature of the domain: the state space explodes for FORWARD, and REVERSE must explicitly consider the disjunction of the two links that support each goal conjunct.

Figure 12(b) shows the explosion in the state space for FORWARD's projection. The result is a state distribution of size 2^N; each state has probability $\frac{1}{2^N}$ and the goal is true in all of them, thus the plan is successful with probability 1. QUERY's projection, on the other hand (Figure 12(c)) does not branch at all. Each of the N actions has a single relevant consequence class, in which the corresponding proposition is made true.

REVERSE has trouble with the domain once again because of disjunction in the assessment expression: each conjunct p_i in the goal expression has two links pointing to it, from e^i_α and e^i_β.

Figure 13 shows average assessment time for this domain, again as a function of plan size. QUERY's assessment time grows linearly with plan size, the others grow exponentially.

---

confidence intervals for average data were too small to plot.

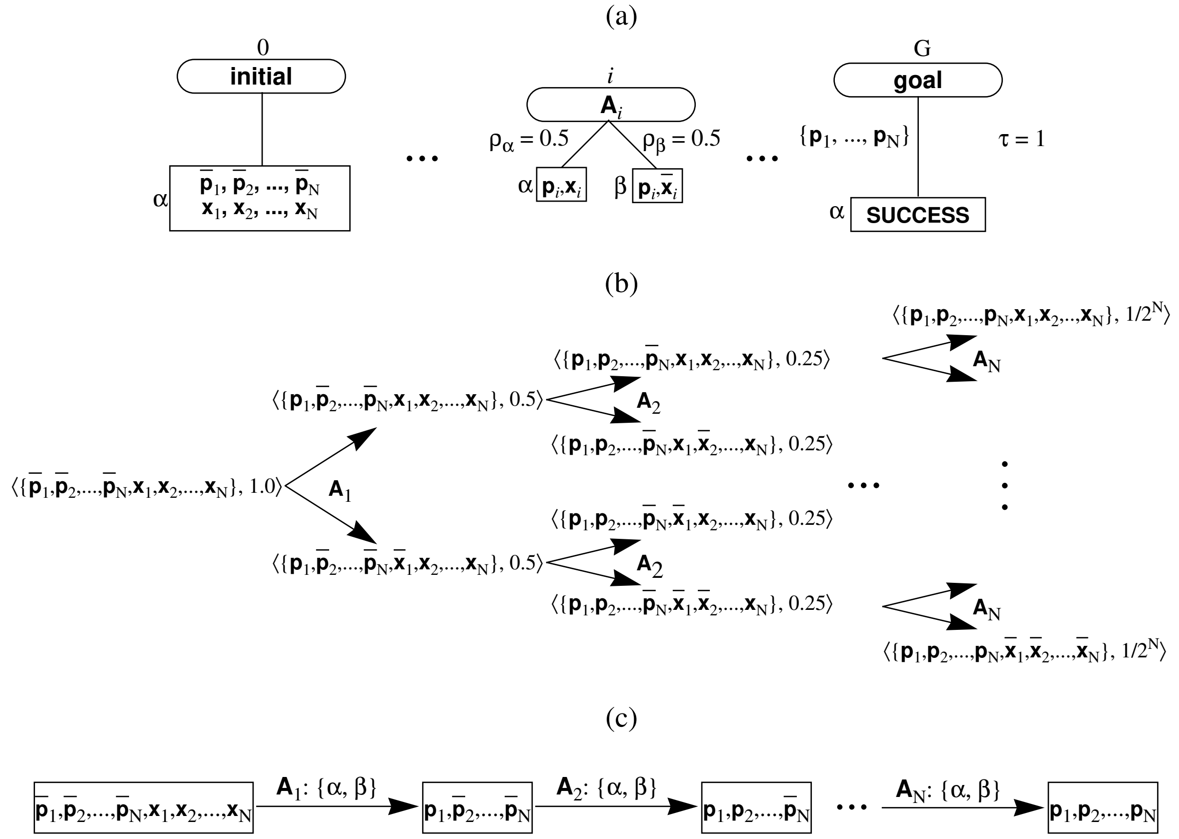


Figure 12: The domain shown in (a) is efficient for QUERY. Although FORWARD doubles the size of its state set after every action (b), QUERY makes no irrelevant distinctions (c).

5.5.3 A domain favoring REVERSE

We noted above that REVERSE has problems with plans that have multiple supporting causal links since these cause long assessment expressions. REVERSE works best in domains in which propositions do not require multiple support, *i.e.* in which each proposition is supported by a single causal link. Such is the case in the domain appearing in Figure 14. There is a single line of causal support from the α consequence of the initial action to goal. Thus the assessment expression is of constant length regardless of the plan's length. Both QUERY and FORWARD have to directly consider the entire collection of states with non-zero probability, which grows exponentially with the length of the plan.

Figure 15 confirms our expectations: assessment time for REVERSE increases linearly with plan size, while both QUERY and FORWARD are exponential. QUERY does somewhat better in the limit than does FORWARD because it can ignore the distinction between the two consequences that do not generate the proposition required in the next action—it produces two branches per action whereas FORWARD generates a three-fold increase in the size of the state distribution after every action.

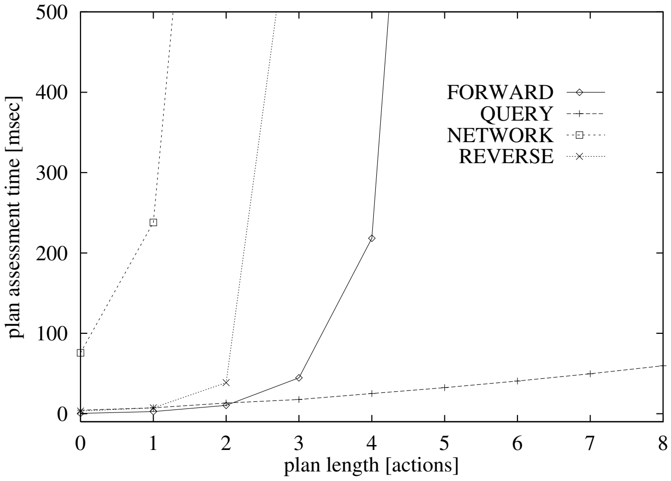
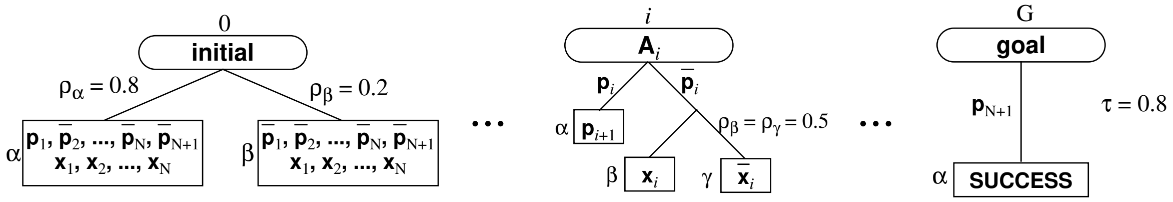


Figure 13: Average assessment time for plans in QUERY's domain.



0
initial
$\rho_\alpha = 0.8$
$\rho_\beta = 0.2$
α
$\mathbf{p}_1, \bar{\mathbf{p}}_2, \ldots, \bar{\mathbf{p}}_N, \bar{\mathbf{p}}_{N+1}$
$\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$
β
$\bar{\mathbf{p}}_1, \bar{\mathbf{p}}_2, \ldots, \bar{\mathbf{p}}_N, \bar{\mathbf{p}}_{N+1}$
$\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$
• • •
i
$\mathbf{A}_i$
$\mathbf{p}_i$
$\bar{\mathbf{p}}_i$
α
$\mathbf{p}_{i+1}$
$\rho_\beta = \rho_\gamma = 0.5$
β
$\mathbf{x}_i$
γ
$\bar{\mathbf{x}}_i$
• • •
G
goal
$\mathbf{p}_{N+1}$
$\tau = 0.8$
α
SUCCESS

Figure 14: An efficient domain for REVERSE.

5.5.4 The NETWORK algorithm

We did not discuss the NETWORK algorithm above, for reasons that should now be clear: its performance was dominated by the other algorithms, and it performed essentially the same on all examples. We noted above that a generic clustering algorithm is probably inappropriate for this application, since it does not exploit the Markov property of the network.

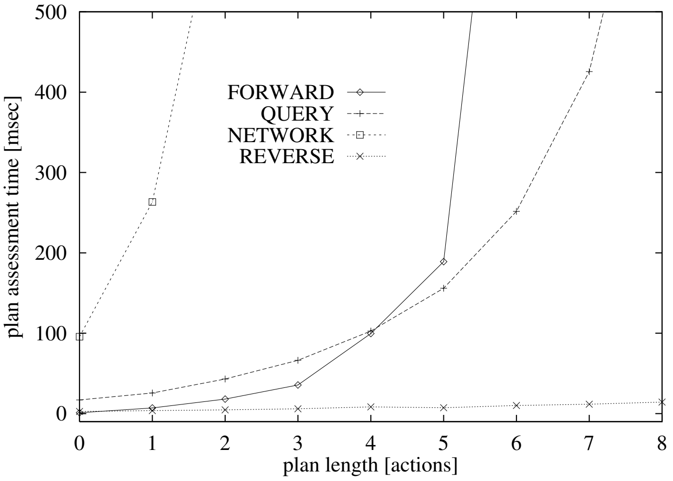


Figure 15: Assessment time for REVERSE's domain.

6 The Assess-Refine Interface

The experiments reported in Section 5.5 were valuable in confirming our intuition about the relationship between domain characteristics and the performance of assessment algorithms, but they should be interpreted narrowly for several reasons: they involved very small, carefully constructed domains, and they measured assessment time in isolation, without regard to the total time spent generating a successful plan.

As a preliminary effort toward a more thorough empirical study we tested the four algorithms on two additional problems. The first is the example used throughout this paper, which we will refer to as the Slippery Gripper problem. The second is an extension of Moore's [41] Bomb and Toilet problem, which we describe below:

> A robot is given two packages, and told that exactly one of them contains a bomb. It wants to defuse the bomb, and the only way to do so is to dunk the package containing the bomb in the toilet. Placing a package in the toilet might (with probability 0.05) clog the toilet, and that is to be avoided.

Suppose we want to achieve both goals—the bomb defused and the toilet unclogged—with probability at least 0.9. The obvious plan is to dunk both packages, guaranteeing that the bomb is defused and incurring only a small risk of clogging the toilet. Indeed, BURIDAN builds the plan shown in Figure 16.

Table 4 shows the total planning time required by the four assessment algorithms, on both the Bomb/Toilet and Slippery Gripper problems. The real surprise here is the poor performance for REVERSE: FORWARD runs about 100 times faster that REVERSE on the Slippery Gripper problem, and about 1000 times faster on the Bomb/Toilet problem. Was the assessment of the plans generated in solving these domains pathologically difficult for REVERSE? Table 5 indicates that this is not the case: although REVERSE was the fastest assessor for Slippery Gripper and the second fastest for Bomb/Toilet, it caused many additional plans to be generated and assessed, and this resulted in significantly slower planning performance.

	Problem	
Algorithm	Slippery Gripper	Bomb/Toilet
FORWARD	4.5	6.9
QUERY	8.0	64.9
NETWORK	179.9	152.0
REVERSE	404.9	6736.0

Table 4: Total planning time (CPU seconds) for two problems and four assessors.

In hindsight, the reason for this behavior is clear. Recall that a fundamental difference between REVERSE and the other algorithms is REVERSE's reliance on the plan's causal-link structure. When a plan doesn't contain all possible links between producing consequences

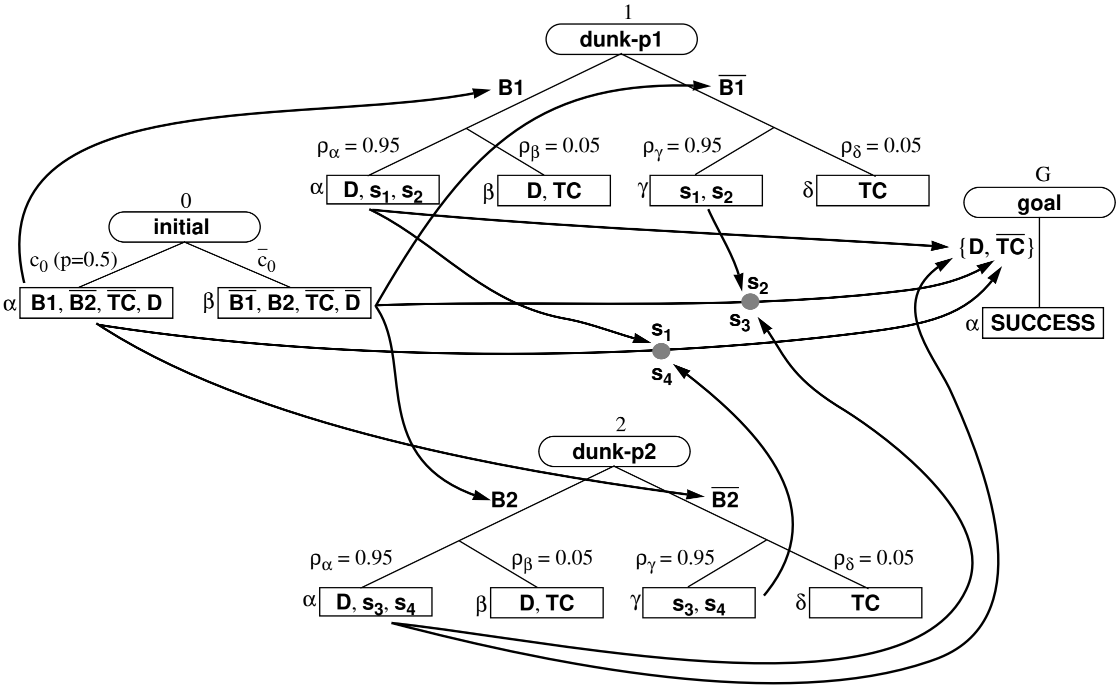

1
dunk-p1
B1
$\overline{B1}$
ρα = 0.95
ρβ = 0.05
ργ = 0.95
ρδ = 0.05
α D, s1, s2
β D, TC
γ s1, s2
δ TC
G
goal
{D, $\overline{TC}$}
α SUCCESS
0
initial
c0 (p=0.5)
$\overline{c_0}$
α B1, $\overline{B2}$, $\overline{TC}$, D
β $\overline{B1}$, B2, $\overline{TC}$, $\overline{D}$
s2
s3
s1
s4
2
dunk-p2
B2
$\overline{B2}$
ρα = 0.95
ρβ = 0.05
ργ = 0.95
ρδ = 0.05
α D, s3, s4
β D, TC
γ s3, s4
δ TC

Figure 16: BURIDAN's solution to the Bomb/Toilet problem. `B1` means "package 1 contains bomb;" `B2` means "package 2 contains bomb;" `TC` means "toilet clogged;" `D` means "bomb defused."

and consuming propositions, REVERSE underestimates the probability of goal achievement, whereas FORWARD, QUERY and NETWORK compute the exact value. When this happens, REVERSE might believe that the plan is not sufficiently likely to succeed, even though in fact the plan *does* represent a solution. In these cases, FORWARD, QUERY, and NETWORK can terminate planning much sooner than REVERSE which requires that REFINE add more causal links before REVERSE can compute a tight bound.

Our hope was that REVERSE would run faster using the cached information in the plan's link structure, and that speed would offset the fact that the planner might need to iterate a few more times to produce a complete plan. Although neither hope is manifested in these examples, these experiments should be interpreted with caution. Two factors tend to make REVERSE look worse than it otherwise might:

- These domains are very small, and contain very little irrelevant information. There are few operators, few propositions in the state space, and few consequences per action. All these factors conspire to make FORWARD look good: both QUERY and REVERSE spend computational effort trying to separate relevant aspects of the problem from irrelevant. If there are no irrelevant aspects to the problem, this effort is obviously wasted.

- Little attention was given to search-control issues. Our plan refinement algorithm used a simple search-control policy of favoring plans with fewer links and fewer actions. In

	Problem			
	Slippery Gripper		Bomb/Toilet	
Algorithm	Time/Assess	Number Assessed	Time/Assess	Number Assessed
FORWARD	11.9	119	8.2	239
QUERY	429.6	119	251.9	239
NETWORK	1521.0	119	636.7	239
REVERSE	10.6	4756	85.2	24420

Table 5: Average assessment time per plan (CPU microseconds), and number of plans assessed before returning a solution.

reality the policy amounted to breadth-first search through the plan space. Better search control would direct the refinement algorithm toward a complete plan more quickly. We address this issue briefly in Section 6.2.

6.1 Reasoning about partial orders

Recall that the assessor is given a partially ordered plan, yet it must reason about total-order completions of that plan. FORWARD and NETWORK deal with partial orders in the obvious way: they generate all completions, assess each one individually, and take the minimum. REVERSE can cope with partially ordered actions in the sense that once threats are eliminated from a plan, an action unordered with respect to a link can never decrease the probability that the proposition will be supported. Since REVERSE computes a lower bound, it can safely ignore any non-threatening actions that might be ordered within the scope of a link.

QUERY reasons quite deeply about partial orders: if the order of actions within a partial order cannot affect the value of a query proposition, it will compute the proposition's truth value without exploring any completions of that order. But in reasoning explicitly about partial orders, QUERY has usurped some of the functionality of the plan refinement algorithm. In particular, the bound QUERY returns on a plan's success is the minimum that can be guaranteed from *any* completion of the plan, whereas the bound REVERSE returns is on what can be guaranteed from *every* completion. QUERY tells the refinement algorithm that *some* completion of its current plan will succeed, but the planner has to figure out which one.[16]

Having the assessor reason about partial orders is a potentially powerful form of search control: the assessor can reason more efficiently about partial orders and notify the planner when it finds a successful plan, thus saving the planner from applying its slower, more general refinement methods to the same task. As a particularly simple example of such a strategy we modified FORWARD and NETWORK to return the *maximum* probability over all possible completions instead of the *minimum*.

---

[16] Alternatively we could extend the interface between assessor and refinement so the former would return the successful completion.

Algorithm	*Problem* Slippery Gripper	 Bomb/Toilet
FORWARD	4.5	6.9
FORWARD-MAX	0.53	7.0
NETWORK	179.9	152.0
NETWORK-MAX	43.6	151.3

Table 6: Total planning time (CPU seconds) is reduced when the assessor recognizes total orders that maximize goal probability, enabling early termination.

As Table 6 demonstrates, the ability for an assessor to hasten recognition of a totally ordered solution can reduce planning time considerably. The improvement is only realized for Slippery Gripper, because the order of the actions in this domain is significant (and therefore BURIDAN must make more ordering decisions before terminating) whereas the order of the two dunk actions in Bomb/Toilet is not significant (so BURIDAN can leave the actions unordered).

6.2 Search control

Although we have only started to address the question of search control, it is clear that an assessment algorithm might be able to provide information that would guide the process of plan refinement. We already saw one example where this information is readily available: the assessor tells the refinement algorithm that a solution can be found by imposing additional order on its current plan (*i.e.* without adding any new actions).

In fact, a powerful way to view the assessment task is as one of discovering flaws in the plan and communicating that information back to the planner [54, 27, 29]. Our QUERY algorithm builds a structure called a *scenario* that is the basis for assessment, but is also a temporal trace of the plan's execution. One can identify from this structure the point at which the plan's probability of success decreased, and why. This information could be exploited in deciding which refinement to apply next.

REVERSE could supply similar information: its assessment expression captures how likely various propositions are to be true at various points in the plan. One could trace back through the assessment structure to find those propositions that are either unlikely to be true, or likely to be clobbered. Once again this information could guide the establishment of new links, or the confrontation of threats.

6.3 Summary

This section explored the interplay between plan assessment and plan refinement. In hindsight, BURIDAN's simple architecture seems problematic. In order to increase planning performance, it will be necessary to create a more sophisticated interface between plan assessment and refinement. Many factors influence overall planning performance: speed of assessment, the tightness of probabilistic bounding calculations, and the type of search control guidance that the assessor can provide.

7 Related Work

Related work can be found in several areas: other AI approaches to probabilistic planning, robotic motion planning, decision models, and classical planning and plan evaluation techniques.

7.1 Probabilistic planning

Several early pieces of work [22, 42] cast planning in probabilistic or decision-theoretic terms, but did not provide concrete representations or algorithms to solve the problem. More recent work divides according to how the planning problem is defined, and how states and operators are represented.

Markov decision processes Several research efforts (*e.g.* [34, 12]) adopt a planning model based on fully observable Markov processes. There are two main differences between this work and ours. First of all, the algorithms operate directly on the state space rather than on its component propositions, and the actions are represented directly as probabilistic mappings from states to states—the algorithms do not manipulate symbolic action descriptions. (Koenig shows a translation from STRIPS-like symbolic operators to the transition-matrix representation, but the solution algorithm does not use the symbolic representation.)

A more important distinction is that these approaches build a *reaction strategy* rather than a plan. A reaction strategy is a policy that dictates the action the agent should take for each state in the state space. A plan, on the other hand, is a sequence of actions that the agent executes without regard to the state. The assumption behind the Markov decision process approach is that the agent will always *know* what state it is in while it is executing its strategy—in other words, that it will be provided with accurate and immediate information about the new world state every time it executes an action.

A plan embodies the opposite assumption—that the agent will get *no* additional information about the world at execution time—so it might as well plan what to do ahead of time. Recent extensions to BURIDAN [16, 15] take a middle ground: that information is available at execution time, but it has to be explicitly gathered, and is potentially inaccurate.

Symbolic planning approaches Farley [21] proposes a similar action representation, though he attaches probabilities directly to postconditions rather than to sets of postconditions. His planning algorithm is linear and "progressive": it starts from the initial state (assumed unique) and builds linear plan sequences, always adding steps to the end of the plan.

Mansell [37] proposes a strategy in which the planner attacks each possible initial world state in isolation (beginning with the most likely), and uses a deterministic hierarchical planning algorithm to build a plan for each. After these plans are built, the algorithm tries to merge the distinct plans. This approach is similar to the "robustification" approach proposed by Drummond and Bresina [17]. BURIDAN can be forced to operate in this mode (by allowing it to link to only a single initial state at a time), though the advantage of postponing the merging process to the end of the planning episode is not clear.

Preliminary work by Goldman and Boddy [24] attacks a similar problem: building plans that are likely to achieve the goal, where likely is defined in terms of a threshold. They develop an extended action and plan representation that incorporates observations and contingencies, so a comparison to C-BURIDAN [16, 15] is more apt. Their approach to planning is quite different from ours, however. They use a deterministic planner (based on CNLP [52]) and they manage uncertainty using an external probabilistic network model to assign probabilities to propositions with unknown truth values. Splitting the problem into a deterministic planner and an external mechanism for managing uncertainty is more similar to Mansell's approach than to ours.

7.2 Robotic motion planning

Robotics researchers have also considered the problem of planning with actions whose effects are uncertain. For example, Lozano-Perez, Mason and Taylor [36] introduced a backward chaining strategy (LMT) for motion planning given sensing and control uncertainty which has been extended by Erdmann [18] and others. An interesting connection between these approaches and ours is the analogy between the use of compliant motion and conditional effects for reducing uncertainty, but there are more differences than similarities. Most obvious is their emphasis on geometry. Second, they model sensing actions (but see Brost [4]) which are omitted from BURIDAN, though the extensions cited above address that deficiency. Third, their *preimage* notion of uncertainty bears more of a resemblance to a possible-worlds model of incomplete information than our probabilistic model. Fourth, their focus is on planning strategies that are guaranteed to succeed despite uncertainty (as are the Markov-process approaches above); in contrast, BURIDAN plans need only have probability of success that exceeds a user specified threshold. Donald's work [14] extends the basic LMT paradigm to handle incomplete knowledge of the world's geometry and to provide error detection and recovery.

7.3 Graphical decision models

Work on graphical probabilistic and decision models (see Howard [32], Pearl [44], or the overview in [11, Chapter 7]) also deals with decision making and planning problems, but has focused more on solving a given probabilistic or decision model whereas our algorithm interleaves the process of constructing and evaluating solutions. The problem modelled by an influence diagram involves choosing options from a fixed set of choices rather than constructing a course of action dynamically from a goal description.

Recent work, however, has recognized the importance of interleaving the model-construction and the model-solution problems, both in general [25] and as applied to the planning problem in particular [51]. Also see [3] for a survey of work in this area.

7.4 Probabilistic temporal reasoning

The representation for the NETWORK algorithm is similar to the network proposed by Dean and Kanazawa [13].

As we discussed in Section 5, a totally ordered plan can be formulated as a probabilistic network allowing assessment to be performed using standard propagation techniques [44]. Although our experiments with the NETWORK assessment algorithm showed that the Jensen clustering algorithm is probably inappropriate for problems of this type, other approaches might be more suitable. Dean and Kanazawa [13] advocate stochastic simulation techniques, but these lack convergence bounds and thus sacrifice soundness and completeness. Recent results [9] also suggest that an approximation algorithm may not be more effective than an exact method.

7.5 Action representation and plan evaluation

Our action representation comes from Hanks's work [29, 30, 28] on probabilistic projection. Chrisman [6] develops an action representation and projection rule for planning under uncertainty, and Martin and Allen [38] develop statistical techniques to gather probabilities like the ones our algorithm uses. None of this work directly addresses the problem of plan generation.

The QUERY algorithm is described in [30, 28]; Drummond [17] presents an alternative algorithm for a similar problem.

Haddawy and Hanks [26] motivate building a planner such as BURIDAN. They provide a framework for constructing a restricted class of utility functions for use by a decision-theoretic planner and show circumstances under which determining whether one plan dominates another reduces to establishing bounds on the probabilities of particular propositions at particular times, which is precisely what our plan assessment algorithms compute. Doyle and Wellman [57] discuss the general problem of modular specification of a planner's objectives in a decision theoretic framework. They exploit multiattribute utility theory to devise techniques for composing separate preference specifications.

7.6 Classical planning

Dealing with state-dependent effects is an essential requirement for any useful probabilistic planner. In this regard BURIDAN can be seen as generalizing the work on planning with deterministic conditional effects, *e.g.* in [47, 7, 50]. A deterministic form of confrontation is used in UCPOP [50]. Pednault's ADL language allowed for disjunctive effects and he used them to solve a simple symbolic version of the "Bomb in the Toilet" example [46] which we extended in Section 6. However, no implementations of ADL (*e.g.*, Pedestal [40] and UCPOP [50]) have implemented the functionality of disjunctive effects, which BURIDAN does.

8 Conclusions

BURIDAN represents a significant step in the development of practical algorithms for probabilistic planning. While much work remains to be done, BURIDAN provides a profitable basis for future study.

8.1 Implementation

BURIDAN is fully implemented in Common Lisp and has been tested on many examples including the ones presented in this paper. The implementation is robust (*e.g.* successfully searches tens of thousands of plans). Although the code has not been optimized for speed or search control, we feel that it is a solid foundation for future research. In addition, it would be excellent in an instructional setting. Send mail to `bug-buridan@cs.washington.edu` for instructions on acquiring BURIDAN source code via anonymous FTP.

8.2 Summary

In this paper we've reported on several significant advances:

1. We have extended the classical planning representation to handle uncertainty in the initial world state (via probability distributions over world states) and in the effects of actions (via mutually exclusive and exhaustive triggers paired with STRIPS-like effects).
2. We provided a precise probabilistic semantics for our representation. Execution of an action causes a transition from one state distribution to another.
3. We described BURIDAN, an implemented algorithm for probabilistic planning, and proved that it is both sound and complete.
4. We compared the efficiency of the FORWARD, QUERY, NETWORK and REVERSE probabilistic assessment algorithms both analytically and empirically. We characterized the strengths of each algorithm, and observe that none of the four is clearly dominant.
5. We noted that the fastest assessor does not necessarily lead to the fastest planner and explain why. We argued that the refine-assess architecture could be improved by allowing the plan assessor to provide more guidance to the plan refiner. As a simple example of this strategy, we demonstrated that considerable speedup is possible when the assessment algorithm returns action-ordering information in addition to its probability calculation.

8.3 Future work

We hope to extend BURIDAN in many directions. From a purely practical perspective, BURIDAN's functionality is limited by its propositional representation, so we plan to implement a lifted [53, 39] version using the codesignation constraint code developed for UCPOP

[50]. The major challenge of this endeavor is devising an efficient means for handling the disjunctive bindings that could result when a lifted trigger condition is supported by multiple causal links from different ground consequences.

Another extension would allow BURIDAN to handle probabilistic exogenous events and incorporate the model of sensing and information advanced in the UWL language [20]. We'd like to integrate Peot and Smith's [52] algorithm for generating conditional plans with this framework and to consider interleaved planning and execution [2, 43, 35] as well. Recent work on C-BURIDAN [16, 15] has addressed some of these issues.

We also hope to introduce an explicit temporal model (perhaps using ideas from ZENO [48, 49]) so we can represent deadline goals. This would allow us to consider integrating our probabilistic plan refinement algorithm with the utility model presented in [26].

On the algorithmic side we have just begun to explore methods of controlling the search for good plans. As Section 6 demonstrates, there are a number of important architectural issues which deserve exploration. We hope to develop a more sophisticated refine-assess interface so that the computational expense of plan assessment pays dividends by guiding subsequent refinements. We also wish to evaluate additional assessment methods (*e.g.* incremental assessment, stochastic simulation, *etc.*) and their relationship to plan refinement.

A Proof of Completeness

Theorem 2 (Completeness) *Let* $\Delta = \langle \tilde{\mathsf{s}}_I, \mathcal{G}, \tau, \Lambda \rangle$ *be a planning problem and let* $\langle \mathsf{A}_i \rangle_{i=1}^{N}$ *be an essential solution (*i.e.*, no proper subsequence is also solution) of* Δ*. Then there exists a sequence of nondeterministic choices such that* BURIDAN(Δ) *will return* $\langle \mathsf{A}_i \rangle_{i=1}^{N}$*.*

To finesse issues of search control, we use $\langle \mathsf{A}_i \rangle_{i=1}^{N}$ as an oracle to guide the construction of the partially ordered plan; McDermott [40] refers to this technique as a *clairvoyant algorithm.* Our implementation uses exhaustive search to ensure that every sequence of nondeterministic choices is eventually considered.

We first establish a useful lemma. Recall that plan data structures contain a set of subgoals: $\mathcal{S} = \{\ldots, \mathsf{p}@\mathsf{A}_i, \ldots\}$. We introduce one new piece of terminology to concisely refer to the result of executing action subsequences: let $\mathcal{SD}_j^k$ be the state distribution produced by executing $\langle \mathsf{A}_i \rangle_{i=j}^{k}$ in $\mathcal{SD}$. If $k < j$ then $\mathcal{SD}_j^k \equiv \mathcal{SD}$.

Lemma 3 *Let* $\Delta = \langle \tilde{\mathsf{s}}_I, \mathcal{G}, \tau, \Lambda \rangle$ *be a planning problem and suppose that a call to* BURIDAN(Δ) *yields values* $\langle \mathcal{A}, \mathcal{O}, \mathcal{L}, \mathcal{S} \rangle$ *such that* $\langle \mathsf{A}_i \rangle_{i=2}^{N}$ *is a consistent topological sort of* $\mathcal{A}$ *(excluding the initial and goal actions). Let* $\mathcal{E}$ *be an expression composed of literals all of which are subgoals in* $\mathcal{S}$ *for the same action* A_m*, and let* A_1 *be some action not in* $\mathcal{A}$*. If there exists* $l < m$ *such that*

$$\mathrm{P}\Big[\mathcal{E} \,|\, \mathcal{SD}_1^l\Big] > \mathrm{P}\Big[\mathcal{E} \,|\, \mathcal{SD}_2^l\Big],$$

then REFINE *can make a sequence of nondeterministic choices that will add* A_1 *to* $\mathcal{A}$*.*

Proof: Our proof is by induction on m.

Base Case: $m = 2$.

In this case $l = 1$ and we assume that $\mathrm{P}\Big[\mathcal{E} \,|\, \mathrm{EXEC}(\mathsf{A}_1, \mathcal{SD}_1^1)\Big] > \mathrm{P}\Big[\mathcal{E} \,|\, \mathcal{SD}_2^1\Big]$, which is equivalent to $\mathrm{P}[\mathcal{E} \,|\, \mathrm{EXEC}(\mathsf{A}_1, \mathcal{SD})] > \mathrm{P}[\mathcal{E} \,|\, \mathcal{SD}]$.

By definition, the only way that the probability of $\mathcal{E}$ can be greater after executing A_1 is if doing so increases the probability associated with the states containing $\mathcal{E}$. But the only way that this could happen is if A_1 has an consequence containing p for some $\mathsf{p} \in \mathcal{E}$. But in that case REFINE Line 2.a could choose to add A_1 to the plan since $\mathsf{p}@\mathsf{A}_2 \in \mathcal{S}$.

Inductive Step: $m > 2$.

The inductive hypothesis guarantees that if there exists some $l < m - 1$ such that $\mathrm{P}\Big[\mathcal{E} \,|\, \mathcal{SD}_1^{l-1}\Big] > \mathrm{P}\Big[\mathcal{E} \,|\, \mathcal{SD}_2^{l-1}\Big]$ then A_1 can be added to the plan. We need to show that this holds for $l = m - 1$ as well.

Suppose that $\mathrm{P}\Big[\mathcal{E} \,|\, \mathcal{SD}_1^{m-1}\Big] > \mathrm{P}\Big[\mathcal{E} \,|\, \mathcal{SD}_2^{m-1}\Big]$. Three (exhaustive but non-exclusive) cases can explain this relationship:

1. The increase in the probability of $\mathcal{E}$ happens *before* A_{m-1} is executed—in other words, $\mathrm{P}\Big[\mathcal{E} \,|\, \mathcal{SD}_1^{m-2}\Big] > \mathrm{P}\Big[\mathcal{E} \,|\, \mathcal{SD}_2^{m-2}\Big]$. But in that case the inductive assumption directly indicates that A_1 could be added.

2. The increase in the probability of $\mathcal{E}$ occurs because including A_1 causes A_{m-1} to contribute additional probability mass to $\mathcal{E}$. Specifically, A_{m-1} contains a consequence $\langle \mathsf{t}_\iota^{m-1}, \rho_\iota^{m-1}, \mathsf{e}_\iota^{m-1} \rangle$ such that e_ι^{m-1} makes some proposition p in $\mathcal{E}$ true, and $\mathrm{P}\left[\mathsf{t}_\iota^{m-1} \,|\, \mathcal{SD}_1^{m-2}\right] > \mathrm{P}\left[\mathsf{t}_\iota^{m-1} \,|\, \mathcal{SD}_2^{m-2}\right]$. But then a nondeterministic choice in REFINE Line 2.a could choose this consequence to support p. So for every $\mathsf{q} \in \mathsf{t}_\iota^{m-1}$ a nondeterministic choice in REFINE Line 2.a could make $\mathsf{q}@\mathsf{A}_{m-1}$ a subgoal as well, the inductive assumption applies to t_ι^{m-1}, and A_1 could be added.

3. Finally, the increase in the probability of $\mathcal{E}$ might occur because including A_1 causes A_{m-1} to contribute *less* probability mass to an consequence that makes $\mathcal{E}$ *false*. Note that in this case $\mathcal{E}$ must have non-zero probability before A_{m-1} is executed, *i.e.* $\mathrm{P}\left[\mathcal{E} \,|\, \mathcal{SD}_2^{m-2}\right] > 0$. But if this is the case then for every proposition $\mathsf{p} \in \mathcal{E}$ there must be some action A_i with a consequence ι that contains p, and REFINE Line 2.b could add causal links $\mathsf{A}_{i,\iota} \xrightarrow{\mathsf{p}} \mathsf{A}_m$, for each such of them.

   It must also be the case that some consequence in A_{m-1} tends to make $\mathcal{E}$ false, and A_1 tends to make that consequence less likely. In other words, A_{m-1} must contain a consequence $\langle \mathsf{t}_\kappa^{m-1}, \rho_\kappa^{m-1}, \mathsf{e}_\kappa^{m-1} \rangle$ such that $\overline{\mathsf{p}} \in \mathsf{e}_\kappa^{m-1}$, where $\mathsf{p} \in \mathcal{E}$, and furthermore $\mathrm{P}\left[\mathsf{t}_\kappa^{m-1} \,|\, \mathcal{SD}_1^{m-2}\right] < \mathrm{P}\left[\mathsf{t}_\kappa^{m-1} \,|\, \mathcal{SD}_2^{m-2}\right]$.

   But in this case, REFINE would recognize the κ consequence of A_{m-1} as a threat and Line 3.c could *confront* the threat. Confronting the threat means that the literals in the triggers of all *non-interfering consequences* of A_{m-1} could be adopted as a subgoal in $\mathcal{S}$ (Line 3.c.iii). Since Definition 1 states that an action's triggers are mutually exclusive and exhaustive, $\mathrm{P}\left[\mathsf{t}_\kappa^{m-1} \,|\, \mathcal{SD}_1^{m-2}\right] < \mathrm{P}\left[\mathsf{t}_\kappa^{m-1} \,|\, \mathcal{SD}_2^{m-2}\right]$ implies that the probability of at least one of A_{m-1}'s *non-interfering* triggers will have *greater* probability when A_1 is executed. But if so the inductive assumption is satisfied and A_1 could be added to the sequence. □

We are now ready to tackle the main theorem. Since the proof is somewhat complex, we sketch the high level concept before delving into the details. The proof method is induction and (unsuprisingly) the induction step is the crux. We demonstrate that a sequence of nondeterministic choices exists which returns an N step plan for a planning problem Δ by constructing a *modified* problem which can be solved in $N-1$ steps. Since the induction hypothesis states that BURIDAN can solve this easier problem, we need only show how the choices made for the modified problem lead to choices that solve Δ itself. Lemma 3 makes this (relatively) straightforward.

Proof (Completeness): Given a planning problem Δ and an essential solution $\langle \mathsf{A}_i \rangle_{i=1}^N$, we need to show two things. First, that REFINE can make a sequence of nondeterministic choices resulting in a plan consistent with $\langle \mathsf{A}_i \rangle_{i=1}^N$. Second, that the FORWARD assessor will recognize that plan as a solution. Our proof is by induction on N, the number of actions in the plan.

Base Case: $N = 0$.

If $N = 0$ then the goal is sufficiently likely without any actions being added: $\mathrm{P}[\mathcal{G} \,|\, \mathcal{SD}] \geq \tau$. A call to BURIDAN will create the null plan for Δ and immediately call FORWARD for assessment. Since there are no actions in $\mathcal{A} - \{\mathsf{A}_0, \mathsf{A}_G\}$, FORWARD Line 3 returns the probability of the single total order consistent with this plan, which by assumption exceeds the threshold. BURIDAN calls TOTAL-ORDER and returns the empty sequence.

Inductive Step: $N \geq 1$.

The inductive assumption ensures that clairvoyant BURIDAN correctly generates solutions of the form $\langle \mathsf{A}_i \rangle_{i=1}^{m}$ for $m < N$. We now show that BURIDAN finds a solution for N-action plans as well.

Let Δ' be the planning problem $\langle \mathrm{EXEC}(\mathsf{A}_1, \mathcal{SD}), \mathcal{G}, \tau, \Lambda \rangle$. By Definition 4, the length $N - 1$ action sequence $\langle \mathsf{A}_i \rangle_{i=2}^{N}$ is an essential solution to Δ'. Clairvoyant BURIDAN(Δ') will therefore generate a partially ordered plan, $\mathcal{P}' = \langle \mathcal{A}', \mathcal{O}', \mathcal{L}', \mathcal{S}' \rangle$, such that $\langle \mathsf{A}_i \rangle_{i=2}^{N}$ is a consistent topological sort. $\mathcal{P}'$ is very similar to the plan that we are seeking, but its initial action is doing double duty, providing probability mass for propositions that $\mathcal{SD}$ and A_1 provided collectively in the original solution.

Now consider the execution trace of all nondeterministic choices made by clairvoyant BURIDAN while constructing $\mathcal{P}'$ for Δ'. We can use this trace, with some modifications, to guide BURIDAN toward a solution to the original problem Δ.

Since the only difference between Δ' and Δ occurs in the initial state distributions $\mathcal{SD}$ and $\mathrm{EXEC}(\mathsf{A}_1, \mathcal{SD})$, we need to guide BURIDAN's choice only when it tries to create a link from the initial action, $\mathsf{A}_0{}'$—otherwise, plan refinement can proceed as it did when $\mathcal{P}'$ was generated. Recall that $\mathsf{A}_0{}'$ (the initial action of $\mathcal{P}'$) corresponds to the state distribution $\mathrm{EXEC}(\mathsf{A}_1, \mathcal{SD})$. If $\mathcal{P}'$ contains a link supporting p whose producing action is $\mathsf{A}_0{}'$, then there exists an consequence of A_0 or A_1 that contains p. In that case we instruct BURIDAN to choose such an consequence and create a link from it.

Note that this argument guarantees that BURIDAN will add actions $\mathsf{A}_2, \ldots, \mathsf{A}_N$ to the plan (along with ordering constraints on them), but it does not guarantee that BURIDAN will add A_1 to the plan: $\mathcal{P}'$ might not contain a link whose producer is $\mathsf{A}_0{}'$. But recall that no proper subsequence of $\langle \mathsf{A}_i \rangle_{i=1}^{N}$ is a solution, therefore

$$\mathrm{P}\Big[\mathcal{G} \,|\, \mathcal{SD}_1^N\Big] > \mathrm{P}\Big[\mathcal{G} \,|\, \mathcal{SD}_2^N\Big].$$

Since $\mathcal{G}$ is a conjunction of propositions that have been adopted as subgoals in $\mathcal{S}$, Lemma 3 guarantees that there is a sequence of nondeterministic choices REFINE can make that will add A_1.

At this point we have established that BURIDAN can add the right *actions* to $\mathcal{A}$, but we haven't yet guaranteed that it will add enough *ordering constraints* to $\mathcal{O}$. In particular we have not guaranteed that A_1 will be constrained to occur first in the plan. If BURIDAN fails to constrain A_1 to be the first action in the plan, then FORWARD will iterate over all total orders consistent with $\mathcal{O}$ and one of these might achieve $\mathcal{G}$ with probability less than τ, meaning that BURIDAN would fail to recognize the solution.

We can show that it is a contradiction to assume that *no* sequence of nondeterministic choices will cause A_1 to be ordered first in the plan. Let $m \geq 2$ be the smallest number such that executing A_1 before A_m achieves the goal with some probability $\geq \tau$, while executing A_1 immediately *after* A_m achieves the goal with some probability $< \tau$. If so there must be a sequence of nondeterministic choices made by REFINE at Lines 2.a and 3.c that create a causal link whose producer is A_m and which is threatened by A_1. But if that is the case, REFINE Line 3.a could demote A_1 by adding $\mathsf{A}_1 < \mathsf{A}_m$ to $\mathcal{O}$.

In summary, if $\langle \mathsf{A}_i \rangle_{i=1}^{N}$ is an essential solution to a planning problem, then a sequence of nondeterministic decisions can cause BURIDAN to add each of actions A_1 through A_N to the plan, along with all relevant ordering constraints. FORWARD will return `Min` $\geq \tau$, and clairvoyant TOTAL-ORDER will return $\langle \mathsf{A}_i \rangle_{i=1}^{N}$ which is a solution to Δ. □

B The REVERSE Assessment Algorithm

REVERSE uses the plan's causal links to evaluate a plan. The probability that a proposition holds when a particular action is executed can be estimated by traversing the link structure that provides causal support to the proposition. The idea is to traverse the links, constructing an *assessment expression*, a boolean combination of causal links, triggers, subgoals and ρ_ι^i terms. Starting from the trigger of goal's SUCCESS outcome, the assessment expression is incrementally transformed as follows:

- The assessment expression for the trigger of a consequence is the *conjunction* of the assessment expressions of the subgoals corresponding to the trigger's conjuncts, conjoined with the consequence's probabilistic term.
- The assessment expression for a subgoal is the *disjunction* of the assessment expressions for all the links supporting the subgoal. If a subgoal has no causal support then no transformation is made.
- The assessment expression for a link is the assessment expression of the trigger for the link's producing outcome, conjoined with a conjunction of the assessment expressions of the subgoals of the safety condition associated with confronted threats.

These transformations are applied repeatedly until the expression is a boolean combination of only subgoals without causal support and probabilistic terms for the consequences that constitute the plan's causal structure. This expression can then be evaluated directly. Table 7 precisely specifies the REVERSE algorithm.

REVERSE computes a lower bound on the probability of plan success. To understand this, note that main difference between REVERSE and the other algorithms is that whereas the others algorithms take into account *all* causal relationships inherent in the plan, REVERSE reasons about only those causal relationships explicitly represented in the plan's link structure. There are therefore two ways in which a probability computed using causal links might differ from the value returned by the exact algorithms:

- There might be a action that produces a proposition that is required by a subsequent action, yet REFINE has not installed a link between those two actions. In that case REVERSE may underestimate the proposition's probability.
- There might be a threat to an existing link that has not been resolved yet by the refinement algorithm. In that case REVERSE may overestimate the probability of the link's supported proposition.

We force REVERSE to produce a lower bound on probabilities by ignoring links that are threatened (see the PROD function in Table 7) and by leaving subgoals with no causal support untransformed (Line 2.b applies only if the subgoal has causal support). When a plan is refined so that all threats are resolved and all subgoals are supported in all possible ways, then REVERSE computes the same probability as the other assessment algorithms.

```
REVERSE($\mathcal{P}$)
  1. Initialize the assessment expression to $\mathsf{t}_\alpha^G$.
  2. Loop: Transform a term from the assessment expression as follows:
       a.  $\mathsf{t}_\iota^i \Rightarrow \rho_\iota^i \wedge \bigwedge_{\mathsf{p} \in \mathsf{t}_\iota^i} \mathsf{p@A}_i$
       b.  $\mathsf{p@A}_j \Rightarrow \bigvee_{\mathsf{A}_{i,\iota}\xrightarrow{\mathsf{p}}\mathsf{A}_j \in \mathrm{PROD}(\mathsf{p@A}_j)} \mathsf{A}_{i,\iota}\xrightarrow{\mathsf{p}}\mathsf{A}_j$    if $\mathrm{PROD}(\mathsf{p@A}_j) \neq \emptyset$
       c.  $\mathsf{A}_{i,\iota}\xrightarrow{\mathsf{p}}\mathsf{A}_j \Rightarrow \mathsf{t}_\iota^i \wedge \bigwedge_{\mathsf{s} \in \mathrm{SAFE}(\mathsf{A}_{i,\iota}\xrightarrow{\mathsf{p}}\mathsf{A}_j)} \mathsf{s@A}_j$
     until no further replacements are possible (i.e., the assessment expression consists only of
     literals with no causal support and terms of the form $\rho_\iota^i$).
  3. Convert the assessment expression to disjunctive normal form.
  4. Using the probabilistic axiom $\mathrm{P}[A \vee B] = \mathrm{P}[A] + \mathrm{P}[B] - \mathrm{P}[A \wedge B]$, compute the set of disjuncts of
     the DNF expression that must be conjoined to compute the probability of the expression as a whole.
  5. Compute the probability of each conjunction as follows:
       If the conjunction contains terms of the form $\rho_\iota^i$ and $\rho_\kappa^i$, or $\mathsf{p@A}_i$ and $\overline{\mathsf{p}}\mathsf{@A}_i$, then
         a. the probability of the expression is 0
       otherwise
         b. remove duplicate terms, substitute probabilities for the remaining terms (the value of
              each $\rho_\iota^i$, and 0 for each remaining $\mathsf{p@A}_i$), and
         c. multiply the results
  6. Add or subtract (as appropriate) the probabilities as computed by line 5 for each of the
     conjunctions generated by line 4.

PROD($\mathsf{p@A}_i$)
  returns the set of $\mathcal{P}$'s unthreatened causal links supporting $\mathsf{p@A}_i$.
SAFE($\mathsf{A}_{i,\iota}\xrightarrow{\mathsf{p}}\mathsf{A}_j$)
  returns the set of safety propositions corresponding to confronted threats against $\mathsf{A}_{i,\iota}\xrightarrow{\mathsf{p}}\mathsf{A}_j$ in $\mathcal{P}$.
```

Table 7: REVERSE plan assessment algorithm.

We have not fully investigated the computational complexity of REVERSE, but clearly the algorithm runs in time exponential in the number of disjuncts in the disjunctive normal form of the assessment expression: Line 5 computes the probability of each conjunction generated by Line 4, and the number of such conjunctions is exponential in the number of disjuncts.

Example: We now show how REVERSE assesses the plan shown in Figure 7. From Line 1 of Table 7, the initial assessment expression is simply t_α^G. This gets transformed by several applications of Lines 2.a and 2.b as follows:

$$\begin{aligned} \mathsf{t}_\alpha^G &\Rightarrow \rho_\alpha^G \wedge \mathsf{HB@A}_G \wedge \mathsf{BP@A}_G \wedge \mathsf{GC@A}_G \\ &\Rightarrow \rho_\alpha^G \wedge \mathsf{A}_{1,\alpha}\xrightarrow{\mathsf{HB}}\mathsf{A}_G \wedge \mathsf{A}_{2,\beta}\xrightarrow{\mathsf{BP}}\mathsf{A}_G \wedge \mathsf{A}_{0,\alpha}\xrightarrow{\mathsf{GC}}\mathsf{A}_G \end{aligned}$$

Each of these links is then expanded using Line 2.c. Expanding $\mathsf{A}_{1,\alpha}\xrightarrow{\mathsf{HB}}\mathsf{A}_G$ and $\mathsf{A}_{2,\beta}\xrightarrow{\mathsf{BP}}\mathsf{A}_G$ is straightforward: these links just expand into their producing outcomes. But $\mathsf{A}_{0,\alpha}\xrightarrow{\mathsf{GC}}\mathsf{A}_G$ was threatened by `paint` (A_2) and this threat was resolved by confrontation. So in addition to the producing outcome, $\mathsf{A}_{0,\alpha}\xrightarrow{\mathsf{GC}}\mathsf{A}_G$ expands into the safety proposition subgoal $\mathsf{s}_1@\mathsf{A}_G$, and we have:

$$\rho^G_\alpha \wedge \mathsf{A}_{1,\alpha}\xrightarrow{\mathsf{HB}}\mathsf{A}_G \wedge \mathsf{A}_{2,\beta}\xrightarrow{\mathsf{BP}}\mathsf{A}_G \wedge \mathsf{A}_{0,\alpha}\xrightarrow{\mathsf{GC}}\mathsf{A}_G \quad\Rightarrow\quad \rho^G_\alpha \wedge \mathsf{t}^1_\alpha \wedge \mathsf{t}^2_\beta \wedge (\mathsf{t}^0_\alpha \wedge \mathsf{s}_1@\mathsf{A}_G)$$

Now applying Lines 2.a, 2.b and 2.c, we have:

$$\begin{aligned}
\rho^G_\alpha \wedge \mathsf{t}^1_\alpha \wedge \mathsf{t}^2_\beta \wedge \mathsf{t}^0_\alpha \wedge \mathsf{s}_1@\mathsf{A}_G \quad &\Rightarrow\quad \rho^G_\alpha \wedge (\mathsf{GD}@\mathsf{A}_1 \wedge \rho^1_\alpha) \wedge (\overline{\mathsf{HB}}@\mathsf{A}_2 \wedge \rho^2_\beta) \wedge \rho^0_\alpha \wedge \mathsf{A}_{2,\beta}\xrightarrow{\mathsf{s}_1}\mathsf{A}_G \\
&\Rightarrow\quad \rho^G_\alpha \wedge (\mathsf{A}_{0,\alpha}\xrightarrow{\mathsf{GD}}\mathsf{A}_1 \wedge \rho^1_\alpha) \wedge (\mathsf{A}_{1,\alpha}\xrightarrow{\overline{\mathsf{HB}}}\mathsf{A}_2 \wedge \rho^2_\beta) \wedge \rho^0_\alpha \wedge \mathsf{t}^2_\beta \\
&\Rightarrow\quad \rho^G_\alpha \wedge (\mathsf{t}^0_\alpha \wedge \rho^1_\alpha) \wedge (\mathsf{t}^0_\alpha \wedge \rho^2_\beta) \wedge \rho^0_\alpha \wedge (\overline{\mathsf{HB}}@\mathsf{A}_2 \wedge \rho^2_\beta) \\
&\Rightarrow\quad \rho^G_\alpha \wedge (\rho^0_\alpha \wedge \rho^1_\alpha) \wedge (\rho^0_\alpha \wedge \rho^2_\beta) \wedge \rho^0_\alpha \wedge (\mathsf{A}_{1,\alpha}\xrightarrow{\overline{\mathsf{HB}}}\mathsf{A}_2 \wedge \rho^2_\beta) \\
&\Rightarrow\quad \rho^G_\alpha \wedge \rho^0_\alpha \wedge \rho^1_\alpha \wedge \rho^0_\alpha \wedge \rho^2_\beta \wedge \rho^0_\alpha \wedge (\mathsf{t}^0_\alpha \wedge \rho^2_\beta) \\
&\Rightarrow\quad \rho^G_\alpha \wedge \rho^0_\alpha \wedge \rho^1_\alpha \wedge \rho^0_\alpha \wedge \rho^2_\beta \wedge \rho^0_\alpha \wedge \rho^0_\alpha \wedge \rho^2_\beta
\end{aligned}$$

As this point, the termination condition of Line 2 is satisfied, so Line 3 transforms the expression into disjunctive normal form. Since it is already just a conjunction, no transformation is needed. For the same reason, Line 4 is trivial: we use Line 5 to assess the probability of the entire expression.

The expression contains no contradictions, so Line 5.a does not apply. Rather, Line 5.b first removes duplicates, substitutes numbers for the remaining terms, and multiples:

$$\begin{aligned}
\rho^G_\alpha \wedge \rho^0_\alpha \wedge \rho^1_\alpha \wedge \rho^0_\alpha \wedge \rho^2_\beta \wedge \rho^0_\alpha \wedge \rho^0_\alpha \wedge \rho^2_\beta \quad &\Rightarrow\quad \rho^G_\alpha \wedge \rho^0_\alpha \wedge \rho^1_\alpha \wedge \rho^2_\beta \\
&\Rightarrow\quad 1.0 \times 0.7 \times 0.95 \times 0.9 \\
&\Rightarrow\quad 0.5985.
\end{aligned}$$

This example illustrates that the probability computed by REVERSE is a lower bound on the exact probability that a plan achieves the goal: the other algorithms return 0.7335 when they assess this plan. In order for REVERSE to realize this exact probability, additional causal links would need to be added to the plan. For example, the link $\mathsf{A}_{0,\beta}\xrightarrow{\mathsf{GC}}\mathsf{A}_G$ represents a way that the goal GC might be achieved that REVERSE has not considered.

References

[1] J. Allen, J. Hendler, and A. Tate, editors. *Readings in Planning.* Morgan Kaufmann, San Mateo, CA, August 1990.

[2] J Ambros-Ingerson and S. Steel. Integrating planning, execution, and monitoring. In *Proc. 7th Nat. Conf. on Artificial Intelligence*, pages 735–740, 1988.

[3] J. Breese, R. Goldman, and M. Wellman, editors. *Notes from the Ninth National Conference on Artificial Intelligence (AAAI-91) Workshop on Knowledge-Based Construction of Probabilistic and Decision Models.* AAAI, July 1991.

[4] R. Brost. Automatic grasp planning in the presence of uncertainty. *International Journal of Robotics Research*, 7(1):3–17, February 1988.

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

[6] L. Chrisman. Abstract Probabilistic Modeling of Action. In *Proc. 1st Int. Conf. on A.I. Planning Systems*, 1992.

[7] G. Collins and L. Pryor. Achieving the functionality of filter conditions in a partial order planner. In *Proc. 10th Nat. Conf. on Artificial Intelligence*, August 1992.

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

[9] P. Dagum and M. Luby. Approximating Probabilistic Inference in Bayesian Networks is NP-Hard. *Artificial Intelligence*, 60(1):141–153, March 1993.

[10] T. Dean and M. Boddy. Reasoning about partially ordered events. *Artificial Intelligence*, 36(3):375–400, October 1988. Reprinted in [56].

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

[12] Thomas Dean, Leslie Kaelbling, Jak Kirman, and Ann Nicholson. Planning with deadlines in stochastic domains. In *Proc. 11th Nat. Conf. on Artificial Intelligence*, July 1993.

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

[14] B. Donald. A geometric approach to error detection and recovery for robot motion planning with uncertainty. *Artificial Intelligence*, 37:223–271, 1988 1988.

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

[16] D. Draper, S. Hanks, and D. Weld. Probabilistic planning with information gathering and contingent execution. In *Proceedings, AI Planning Systems*, 1994.

[17] M. Drummond and J. Bresina. Anytime Synthetic Projection: Maximizing the Probability of Goal Satisfaction. In *Proc. 8th Nat. Conf. on Artificial Intelligence*, 1990.

[18] M. Erdmann. On Motion Planning with Uncertainty. AI-TR-810, MIT AI LAB, August 1984.

[19] K. Erol, D. Nau, and V. Subrahmanian. When is planning decidable? In *Proc. 1st Int. Conf. on A.I. Planning Systems*, pages 222–227, June 1992.

[20] O. Etzioni, S. Hanks, D. Weld, D. Draper, N. Lesh, and M. Williamson. An Approach to Planning with Incomplete Information. In *Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning*, October 1992. Available via anonymous FTP from `~ftp/pub/ai/` at `cs.washington.edu`.

[21] Arthur M. Farley. A Probabilistic Model for Uncertain Problem Solving. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(4), July 1983.

[22] J. Feldman and R. Sproull. Decision theory and artificial intelligence II: The hungry monkey. *Cognitive Science*, 1:158–192, 1977.

[23] R. Fikes and N. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3/4), 1971.

[24] Robert P. Goldman and Mark S. Boddy. Epsilon-safe planning. forthcoming, 1994.

[25] Robert P. Goldman and John S. Breese. Integrating Model Construction and Evaluation. In *Proc. 8th Conf. on Uncertainty in Artifical Intelligence*, July 1992.

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

[27] K. Hammond. Explaining and repairing plans that fail. *Artificial Intelligence*, 45:173–228, 1990.

[28] S. Hanks. Practical temporal projection. In *Proc. 8th Nat. Conf. on Artificial Intelligence*, pages 158–163, August 1990.

[29] S. Hanks. Projecting plans about uncertain worlds. Ph.d. thesis, Yale University Computer Science Department, January 1990.

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

[31] Steve Hanks and Drew McDermott. Modeling a Dynamic and Uncertain World I: Symbolic and Probabilistic Reasoning about Change. *Artificial Intelligence*, 65(2), 1994.

[32] R. Howard and J. Matheson. Influence Diagrams. In *The Principles and Applications of Decision Analysis*. Strategic Decisions Group, 1984.

[33] S. Kambhampati. Characterizing multi-contributor causal structures for planning. In *Proc. 1st Int. Conf. on A.I. Planning Systems*, pages 116–125, June 1992.

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

[35] K. Krebsbach, D. Olawsky, and M. Gini. An empirical study of sensing and defaulting in planning. In *Proc. 1st Int. Conf. on A.I. Planning Systems*, pages 136–144, June 1992.

[36] T. Lozano-Perez, M. Mason, and R. Taylor. Automatic synthesis of fine motion strategies for robots. *International Journal of Robotics Research*, 3(1):3–24, Spring 1984.

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

[38] N. Martin and J. Allen. A Language for Planning with Statistics. In *Proc. 7th Conf. on Uncertainty in Artifical Intelligence*, 1991.

[39] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proc. 9th Nat. Conf. on Artificial Intelligence*, pages 634–639, July 1991. internet file at ftp.ai.mit.edu: /pub/users/dam/aaai91c.ps.

[40] D. McDermott. Regression planning. *International Journal of Intelligent Systems*, 6:357–416, 1991.

[41] R. Moore. A Formal Theory of Knowledge and Action. In J. Hobbs and R. Moore, editors, *Formal Theories of the Commonsense World*. Ablex, Norwood, NJ, 1985.

[42] John H. Munson. Robot Planning, Execution, and Monitoring in an Uncertain Environment. In *Proc. 2nd Int. Joint Conf. on Artificial Intelligence*, pages 338–349, August 1971.

[43] D. Olawsky and M. Gini. Deferred planning and sensor use. In *Proceedings, DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*. Morgan Kaufmann, 1990.

[44] J. Pearl. *Probablistic Reasoning in Intelligent Systems*. Morgan Kaufmann, San Mateo, CA, 1988.

[45] E. Pednault. *Toward a Mathematical Theory of Plan Synthesis.* PhD thesis, Stanford University, December 1986.

[46] E. Pednault. Synthesizing plans that contain actions with context-dependent effects. *Computational Intelligence*, 4(4):356–372, 1988.

[47] E.. Pednault. Generalizing nonlinear planning to handle complex goals and actions with context-dependent effects. In *Proceedings IJCAI-91*, pages 240–245, July 1991.

[48] J. S. Penberthy and Daniel S. Weld. A new approach to temporal planning (preliminary report). In *Proceedings of the AAAI 1993 Symposium on Foundations of Automatic Planning: The Classical Approach and Beyond*, pages 112–116, March 1993.

[49] J.S. Penberthy. *Planning with Continuous Change.* PhD thesis, University of Washington, 1993. Available as UW CSE Tech Report 93-12-01.

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

[51] M. Peot and John S. Breese. Model Construction in Planning. In *Notes from the Ninth National Conference on Artificial Intelligence (AAAI-91) Workshop on Knowledge-Based Construction of Probabilistic and Decision Models*, pages 95–100, July 1991.

[52] M. Peot and D. Smith. Conditional Nonlinear Planning. In *Proc. 1st Int. Conf. on A.I. Planning Systems*, pages 189–197, June 1992.

[53] J. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1), January 1965.

[54] R. Simmons. A theory of debugging plans and interpretations. In *Proc. 7th Nat. Conf. on Artificial Intelligence*, pages 94–99, August 1988.

[55] Sampath Srinivas and Jack Breese. IDEAL: Influence diagram evaluation and analysis in lisp; documentation and users guide. Technical Memo 23, Rockwell International Science Center, August 1990.

[56] D. Weld and J. de Kleer, editors. *Readings in Qualitative Reasoning about Physical Systems.* Morgan Kaufmann, San Mateo, CA, August 1989.

[57] M. Wellman and J. Doyle. Modular utility reprsentation for decision theoretic planning. In *Proc. 1st Int. Conf. on A.I. Planning Systems*, pages 236–242, June 1992.