"Classical" Planning under Uncertainty*

Steve Hanks
Department of Computer Science and Engineering
University of Washington, Box 352350
Seattle, WA 98195-2350
hanks@cs.washington.edu

Abstract

Research in the classical paradigm has produced effective representations and algorithms for building courses of actions that achieve an input goal. For the most part these systems have also assumed an omnipotent and omniscient agent: the agent knows with certainty the initial state of the world and what changes its actions will make, and is assured that no exogenous forces will change the world. These assumptions result in two features of classical planners: the output is an artifact that can be *proved* (by logical deduction) to solve the planning problem, and execution-time feedback is irrelevant.

While the classical certainty assumptions are justified in some problem domains and are reasonable approximations in others, an intelligent agent must in some cases cope with situations in which it has incomplete information, faulty sensors and effectors, and in which the world sometimes changes in ways the agent cannot predict perfectly. Our research is oriented toward relaxing the classical-planning certainty assumptions while keeping the essence of the classical representations and algorithms: symbolic state and operator descriptions and problem solving by backchaining on goals and action preconditions.

This paper describes a line of research oriented toward building planners in the classical style but with probabilistic semantics. The BURIDAN planner uses classical planning techniques to build straight-line plans (without feedback) that *probably* achieve a goal; the C-BURIDAN planner extends the representation and algorithm to handle sensing actions and conditional plans. The CL-BURIDAN planner adds the concept of a looping construct. In each case the ability to plan in uncertain domains is gained without sacrificing the essence of classical state-space operators or goal-directed backchaining algorithms.

*This paper reports on a number of related research projects conducted at the University of Washington over the past three years. The research was conducted by Denise Draper, Adam Carlson, Steve Hanks, Nick Kushmerick, and Dan Weld.

Introduction

The BURIDAN planners comprise a family of problem-solving algorithms descended from the classical planning paradigm (see Figure 1 for a summary). The systems start with a problem definition and solution techniques based on STRIPS and earlier planners, including an emphasis on symbolic operators and a backchaining (goal-pursuit) algorithm. They also adopt SNLP's algorithmic commitment to searching a space of *plans* as opposed to a space of possible world states.

The work on BURIDAN (Kushmerick, Hanks, & Weld 1995) restates the planning problem in probabilistic terms, which is further extended in C-BURIDAN (Draper, Hanks, & Weld 1994b; 1994a) to include information-gathering actions and conditionals. The CL-BURIDAN system adds looping constructs to the plan language and also introduces the idea of "plan blocks" which facilitate generating conditional and iterative plans and also aid in the process of assessing plan quality.

In this paper we describe the systems' semantics, representation, and algorithm, and describe new concepts involving block-structured plans and iteration. We end with a discussion of related and current work.

Semantics

The classical planning problem is defined formally in terms of the following components:

- *States* and *expressions*: a state is a complete description of the world (or some part thereof) at an instant in time. The only requirement of a state is that from the state it must be possible to determine (with certainty) whether any *expression* is true. An expression is generally represented as a boolean combination of *propositions*. That is, there is a function

  $$\texttt{true-in}(\textit{expression}, \textit{state}) \rightarrow \textit{boolean}$$

  defined over all states and all expressions.
- *Actions* describe state changes: an action is described fully by a function

  $$\texttt{exec}(\textit{action}, \textit{state}) \rightarrow \textit{state}$$

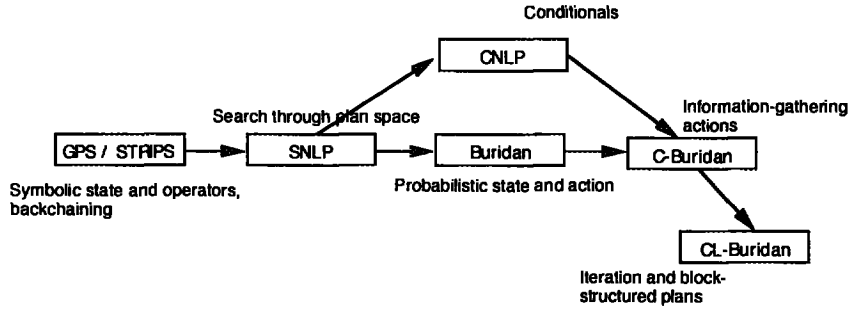

Figure 1: Chronology of probabilistic planners

that produces the successor state if *action* is executed in *state*. This function need *not* be defined for all action/state pairs: an action might not be executable in all states, and in that case the exec function is undefined. The exec function can easily be defined for sequences of actions as well:

$$\begin{aligned} \texttt{exec}(\langle\rangle, s) &= s \\ \texttt{exec}(\langle A_1, \ldots, A_n\rangle, s) &= \texttt{exec}(\langle A_2, \ldots, A_n\rangle, \texttt{exec}(A_1, s)) \end{aligned}$$

- *Input state* and *goal expression*. The former is the (unique) starting state and the latter is an expression that must hold true when execution completes.
- If I is the problem's initial state and G is the problem's goal expression, then a *solution* is *any* sequence of actions $\langle A_1, \ldots, A_n\rangle$ that satisfies $\texttt{true-in}(G, \texttt{exec}(\langle A_1, \ldots, A_n\rangle, I))$.

The semantics for BURIDAN are only slightly different. A state still describes the world completely, but the agent's state of information generally takes the form of a *probability distribution* over states $\tilde{s}$. Instead of a single initial input state, BURIDAN takes as input a probability distribution over initial states.

BURIDAN still manipulates expressions, and the function true-in still determines (with certainty) whether an expression is true in a state. But we generally will need to compute the *probability* that an expression is true given a distribution over states. If e is an expression and s is a state, then

$$\mathrm{P}[e \,|\, \tilde{s}] = \sum_s \mathrm{P}[e \,|\, s]\mathrm{P}[\tilde{s} = s]$$

where $\mathrm{P}[e \,|\, s] = 1.0$ if $\texttt{true-in}(e, s) = \texttt{true}$ and 0.0 otherwise.

Actions are defined in terms of a *probabilistic* mapping from a state to successor states—instead of the exec function that selects a unique successor, we define actions in terms of conditional probabilities of the form $\mathrm{P}[s' \,|\, A, s]$ the probability that executing A in state s will result in a successor state s'. Since generally the input state will not be known with certainty, we further define the probability of a successor state relative to a distribution over input states:

$$\mathrm{P}[s' \,|\, A, \tilde{s}] = \sum_s \mathrm{P}[s' \,|\, A, s]\mathrm{P}[\tilde{s} = s]$$

This definition can likewise be extended and to sequences of actions:

$$\begin{aligned} \mathrm{P}[s' \,|\, \langle\rangle, \tilde{s}] &= \mathrm{P}[\tilde{s} = s'] \\ \mathrm{P}[s'' \,|\, \langle A_1 \ldots A_n\rangle, \tilde{s}] &= \sum_{s'} \mathrm{P}[s' \,|\, A_1, \tilde{s}]\mathrm{P}[s'' \,|\, \langle A_2, \ldots, A_n\rangle, \end{aligned}$$

Finally we define a solution plan a little differently, with respect to a *success threshold* which is an input parameter. Then we define a solution as follows, given a distribution $\tilde{s}$ over initial states, a goal expression G and a probability threshold τ,

- A *solution* is any sequence of actions $\langle A_1, \ldots, A_n\rangle$ such that $\mathrm{P}[G \,|\, \langle A_1, \ldots, A_n\rangle, \tilde{s}] \geq \tau$.

Extensions to BURIDAN differ on the definition of a plan: a BURIDAN plan is simply a sequence of actions, in C-BURIDAN actions can be executed conditionally depending on the results of prior sensing actions, and CL-BURIDAN adds the notion of a sequence of actions that can be executed *iteratively*.

System Descriptions

In the previous section we described the systems' semantics without regard to how they represented or processed information. Here we briefly describe the implementations. We will do so by discussing the C-BURIDAN system, which is a superset of BURIDAN.

We will demonstrate C-BURIDAN's operation using a short example adapted from (Draper, Hanks, & Weld 1994b). Our agent processes parts in a factory, and its goal is to process a part (make proposition PR true). The correct way to process a part depends on whether the part is flawed (FL). If the part is flawed, it should be discarded. If the part is not flawed ($\overline{\text{FL}}$), it should be shipped, which requires packing materials (PM). Shipping and discarding both make the part unavailable ($\overline{\text{AV}}$), so it is futile to try to both ship and discard it. There is a 25% chance that a given part will be flawed.

A sensor is available to determine whether or not FL is true, but it is not a perfect sensor. For one thing,

the sensor must be turned on (ON), otherwise it generates no useful report. If it is on, the sensor generates a report of *YES* if it thinks FL is true, and a report of *NO* if it thinks FL is false. The correspondence between FL and the report depends on whether or not the sensor is clean (CL): if so, the report almost always represents FL's state accurately. If not, the sensor often mis-reports. There is a 90% chance that CL is true initially.

Actions

This problem can be represented by the set of actions shown in Figure 2. Just as in the STRIPS representation, each is a mapping from preconditions to effects, but in this representation an action can have one of a number of mutually exclusive and exhaustive *consequences*. Each leaf of the tree for an action represents one possible consequence: its precondition is the conjunction of the propositions on the path to the leaf, and its effects (adds and deletes) is the list of propositions in the box at the leaf. The numeric parameters on some arcs (e.g. sense and turn-on) are probabilistic "noise" terms: turn-on is a noisy effector that makes ON true 99% of the time; sense is also noisy, but the probabilities depend on the state of the propositions ON, CL, and FL. The goal action is a "dummy" action which stores the planning problem's goal expression. It is not executed by the agent.

All the actions except sense are non-sensing actions, and thus are part of BURIDAN too. Recall from the previous section that actions are defined in terms of conditional probabilities of the form $P[s' \mid A, s]$, but the representation specifies this probability in two phases, first describing the input state s then the *changes* the action will make to the state. Changes are described using a set of propositions: the ship action, for example has a single change set $\{\mathsf{PR}, \overline{\mathsf{AV}}\}$ indicating that it will make PR true, make AV false, and leave all other propositions unchanged. We assume a function $\mathtt{update}(s, c) \rightarrow s'$ where s and s' are states and c is a set of changes. The update function produces the state where all positive atoms in c are true in s', all negative atoms in c are false in s', and all atoms that do not appear in c have the same state in s' as they do in s.

Thus we can define the ship action as follows:

$$P[\mathtt{update}(s, \{\mathsf{PR}, \overline{\mathsf{AV}}\}) \mid s, \mathtt{ship}, \mathtt{true\text{-}in}(\overline{\mathsf{FL}} \wedge \mathsf{AV} \wedge \mathsf{PM}, s)] = 1.0$$

$$P[\mathtt{update}(s, \{\}) \mid s, \mathtt{ship}, \overline{\mathtt{true\text{-}in}(\overline{\mathsf{FL}} \wedge \mathsf{AV} \wedge \mathsf{PM}, s)}] = 1.0$$

Each consequence defines one possible outcome of the action, and the tree structure of our representation ensures that the consequences of every are mutually exclusive and exhaustive. The sense action is described by nine consequences:

$$P[\mathtt{update}(s, \{\overline{\mathsf{ON}}\}) \mid s, \mathtt{sense}, \mathtt{true\text{-}in}(\mathsf{ON} \wedge \mathsf{CL} \wedge \mathsf{FL}, s)] = 0.99$$

$$P[\mathtt{update}(s, \{\overline{\mathsf{ON}}\}) \mid s, \mathtt{sense}, \mathtt{true\text{-}in}(\mathsf{ON} \wedge \mathsf{CL} \wedge \mathsf{FL}, s)] = 0.01$$

$$\ldots$$

$$P[\mathtt{update}(s, \{\}) \mid s, \mathtt{sense}, \overline{\mathtt{true\text{-}in}(\overline{\mathsf{ON}}, s)}] = 1.0$$

The first two consequences of sense do not differ in the changes they make to the world (they have the same change set), but in the *information* they provide about the world.

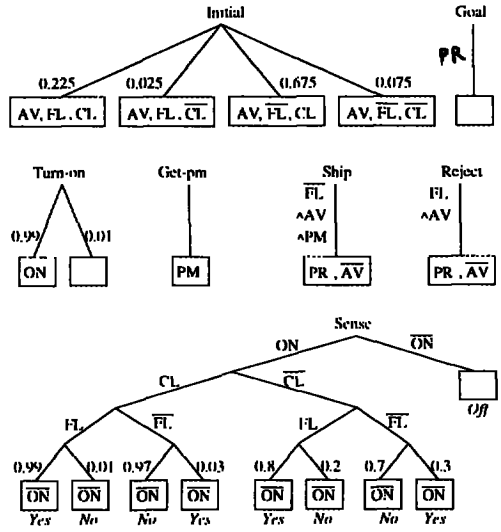

Figure 2: Sample C-BURIDAN actions

Sensing actions The sense action both makes changes to the world and provides information about the world state. Its effect consists of turning itself off, and the information is conveyed by one of three *messages*—*YES*, *NO*, or *OFF*—that it will generate when it is executed. A message is associated with each action consequence, and gives the agent information about which consequence actually occurred at execution time. If the agent executes sense and receives the message *OFF*, for example, it knows that the rightmost consequence occurred and as a result knows (for sure) that ON was false at the time sense was executed. The case with messages *YES* and *NO* is less clear, however, because four consequences could have generated either one. Suppose that the agent is certain that ON and CL were both true when it executed sense, and it receives a message of *YES*. In that case it is fairly certain that FL is true, since only with probability .03 would the sensor report *YES* if FL were false. So sense provides very good information about FL if it is executed when ON and CL are both true, less good information if ON is true but CL is false, and no information about FL is ON is false. See (Draper, Hanks, & Weld 1994a) for a precise account of the semantics of sensing actions.

The posterior probability of FL given a message of *YES* is computed using Bayes rule. Suppose the prior probability for FL is $P[\mathsf{FL} \mid \mathsf{CL}] = P[\mathsf{FL}] = f$ and CL is certainly true. Then

$$P[\mathsf{FL} \mid \mathtt{YES}] = P[\mathsf{FL} \mid \mathtt{YES}, \mathsf{CL}]$$

$$= \frac{P[YES|FL,CL]P[FL|CL]}{P[YES|FL,CL]P[FL|CL] + P[YES|\overline{FL},CL]P[\overline{FL}|CL]}$$
$$= \frac{(.99)f}{(.99)f + .03(1-f)}$$

Notice that $P[YES|FL,CL]$ and $P[YES|\overline{FL},CL]$ can be read directly from the action representation. For the example's prior of 0.25, for example, the posterior on FL is .92, so sense is a good sensor of FL provided that ON and CL are both true. If CL is false the posterior is only 0.47, and if ON is false the posterior is 0.25 the same as the prior, so senseing with a sensor that is off provides no information about FL.

It is worth summarizing the basic features of our state and action representation:

- Although the system is described semantically in terms of states, the system rather manipulates symbolic descriptions of sets of states (expressions) and changes to states (changes sets).
- Incomplete information is captured as a probability distribution over world states. Thus there is no difference between "uncertainty" and "incomplete information."
- Actions are probabilistic mappings from states to states (though once again the transition matrix is not stored explicitly).
- An action can change the agent's state of information in two ways, through the changes it makes to the world, and through the "messages" or "observations" it generates at execution time.
- The observations provide information about the execution-time world only indirectly: the agent reasons backward to determine what world state(s) could have generated that observation, and updates its state of belief accordingly using Bayes rule.
- There is no restriction on the outcomes an action can have; in particular actions are not forced to be either "sensors" or "effectors" and causal and information effects can be freely mixed within an action.
- Both causal and observational effects of an action can be arbitrarily state dependent and noisy.

Conditional Plans

Figure 3 shows a conditional plan built from these steps: it consists of turning on the sensor then sensing, then if the sensor reports *YES*, executing reject, or if the sensor reports *NO*, then executing ship. The get-pm step will be executed unconditionally, any time before ship. Steps are executed conditionally on the basis of sensor reports, which correspond to a particular state of belief in the state of the action's preconditions: reject will be executed if a report of *YES* is received because $P[FL|YES]$ is high.

The arcs in the diagram correspond to the three different kinds of links generated during the planning process. These links record dependency information about the steps in the plan.

The first is a ***causal link***, for example the one connecting connecting the single consequence of get-pm that makes PM true to the precondition of ship requiring it to be true. These links record a probabilistic dependency in the plan: PM is probably true (in this case certainly true) when ship is executed because get-pm makes it so. AV is true when reject is executed because it was initially true and nobody changed it. These links are identical to causal links in deterministic planners, except that multiple links can point to the same precondition (e.g. to PR).

The second link type, shown as dotted lines, represent ***threats***, just as in deterministic planners. Notice that links from the initial state to ship and to reject both require that AV be true, but both ship and reject make AV false, thus threatening the links asserting AV.

The final links are the wide ***conditioning*** links from sense to ship and from sense to reject. These links indicate that ship should be executed only when sense generates a message of *NO*, and that reject should be executed only when sense generates a message of *YES*. Since any execution will generate only one of these messages, the threat between ship and reject is effectively resolved by the conditioning link: reject will be executed only if the *NO* message is received, but in that case ship will not be executed, and therefore its links are unimportant.

We will not present the C-BURIDAN algorithm in detail, noting only that its plan construction algorithm is quite similar to other plan-space causal-link planners. The planner begins with the ***initial plan*** consisting of just the initial and goal steps and a list of ***active subgoals*** initialized to the propositions in the goal expression (the preconditions of goal's single consequence). The planner then enters an assess/refine loop. The planner *assesses* the current plan's probability of success, and if it is below the input threshold τ, the plan is nondeterministically refined. Refinements consist of linking from an existing step or creating a new step to increase the probability of some active subgoal, or resolving a threat by adding ordering constraints or using conditioning. Conditioning is achieved by partitioning the messages of a sensing action and using conditioning links to insure that the threatening step and the threatened step are not both executed.

Plan Blocks, Conditionals, and Loops

In the remainder of the paper we will briefly describe two pieces of ongoing and related work. The first introduces the idea of hierarchically nested "blocks" into the plan representation. Roughly speaking, the plan is partitioned into blocks (each a partially ordered sequence of actions), each of which serves a particular purpose in the plan, and which are mainly causally independent (in a sense we will describe below). Doing so improves performance in a number of ways. First it aids in the *assessment* process. Currently, each cycle of the planner generates a new refinement to a par-

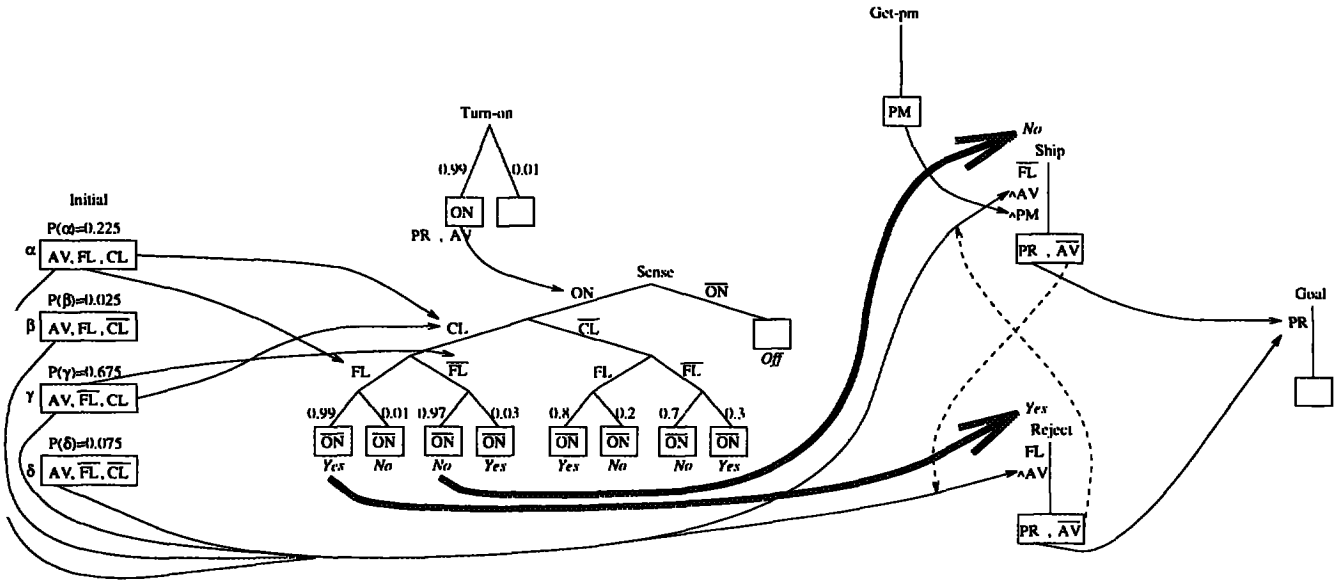

Figure 3: A conditional plan

tial plan, and its success probability must be computed from scratch. Intuitively most of this computational effort is unnecessary, since each new refinement changes only one small part of the plan—we should be able to cache assessment information and reuse that information for those parts of the plan that do not change. The fact that blocks record the causal structure of the plan means that assessment information for each block can be cached there, and the cache is invalidated only if the block itself is refined. Plan blocks also provide a natural implementation for conditionals and loops, as we will see below.

Loops, unlike blocks, actually enhance the expressive power of the plan language. Iteration is necessary for completeness—one can easily construct problems for which there is a solution plan that C-BURIDAN cannot generate, because there can be no *a priori* bound on the number of steps in the plan. Our iteration construct allows CL-BURIDAN to generate plans like "continue to pound the nail until at least two of your sensors tell you it is flush with the surface." This plan has bounded length (as measured by the number of steps in the plan itself), but we can provide no bound on the number of iterations that will actually occur when the plan is actually executed.

Plan blocks

We begin the discussion of plan blocks with an illustration: Figure 4 shows the plan from Figure 3 organized into a nested structure of blocks: the plan itself is a block B1 containing various "causally local" sub-blocks: B5 and B8 produce PR, and B2 provides information about FL. Each of these blocks can also contain sub-blocks: in B2 there is a block B3 to make ON true, then a block B4 that actually does the sensing.

A block contains either a single step (e.g. B3), or a partially ordered set of blocks (e.g. B1 or B2). In either case a block represents a mapping from an *entry condition* (a boolean combination of propositions) to an *exit condition* (a list of propositions that acts as an effects list). The entry condition is shown at the left of the block, the exit condition is shown at the right. The block represents a commitment to leave the world in a state in which its exit condition is true, provided it is executed in a state in which its entry condition is true. Of course since sensors and effectors are not always perfect, the relationship between a block's entry condition and exit condition will generally be probabilistic: we will be computing the efficacy of the block, which can be expressed as P[exit(B)|entry(B)]. The plan's success will be assessed as

$$\mathsf{P[exit(B1)|entry(B1)]\ P[entry(B1)]}$$

where P[entry(B1)] can be computed directly by examining the probability distribution over initial states.

The "causal locality" enforced by this structure stems from restrictions placed on links: links from any step outside a block to any step inside a block must come through the block's entry condition. For example, if ON needs to be true for the sense action in B4, then the link must either come from another step in B2 or from B2's entry condition. One consequence of this structure is that the interleaving of steps in the plan is restricted: although steps or sub-blocks within any block can be unordered with respect to each other, and although any two blocks in the same immediate containing block can be unordered with respect to each

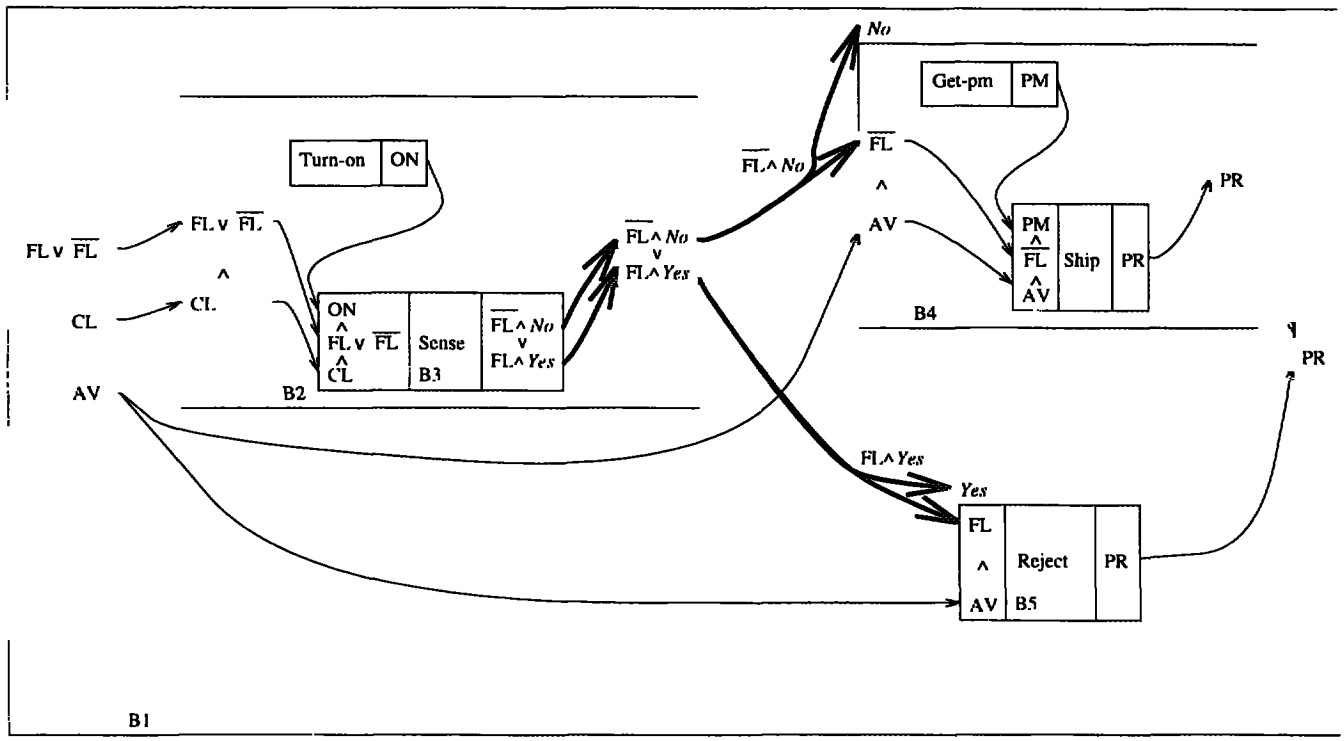

Figure 4: A block-structured plan

other, all the steps in a single block must be executed either before or after all the steps in every other disjoint block.

The details of producing block-structured plans do not differ significantly from the C-BURIDAN plan refinement operators. We defer the details to a longer paper, pointing out here only that

- Initializing the planner consists of building a single block with the goal as the exit condition and no entry conditions - we initially hope to build a plan that will work regardless of the initial world state
- Causal links can additionally connect a proposition in a block's exit condition to an existing subgoal. Links cannot cross block boundaries, however.
- An additional refinement operation involves moving a subgoal to the entry condition of the containing block. This is a generalization of the SNLP refinement that assumes that a subgoal will be true initially.
- Blocks can threaten each other, and the threats can be resolved by ordering or by conditioning (assigning disjoint execution contexts). Threats between blocks cannot be resolved by confrontation, however.

Blocks, Assessment, and Caching

We have found two computational benefits of imposing this additional structure on plans: ease in caching assessment results and ease in implementing and building plans with conditionals and loops.

Plan assessment Computation directed toward determining the probability that the plan will succeed can be performed in a "divide and conquer" manner, with the probability for each block computed in terms only of the blocks it contains. More important, assessment information about the block can be cached, and remains valid unless the structure of that block changes. Put another way, assessment is process of computing the probability that block B will achieve its exit condition,

$$P[B] = P[\text{exit}(B) \mid \text{entry}(B)]P[\text{entry}(B)]$$

where $P[\text{exit}(B1) \mid \text{entry}(B1)]$ depends only on the structure of B itself and does not change as other parts of the plan are refined. The value of $P[\text{exit}(B1) \mid \text{entry}(B1)]$ can therefore be stored with B1 itself, and assessment is just a matter of combining the assessment values of blocks at the same level of nesting. The cache for a block is invalidated only when the structure of that block is refined, so only a very small part of the plan's assessment probability need be computed at each refinement cycle. This addresses a serious computational problem with BURIDAN and C-BURIDAN in that the expensive assessment computation had to be performed anew every time any part of the plan was changed.

Conditionals and Loops

Both conditionals and loops are constructed by restricting certain steps—those in the loop body or those in one branch of the conditional—to a particular *execution context.* In the C-BURIDAN algorithm a step is added to the plan with an unrestricted context, then its context is restricted if and when a threat is detected. This process can be quite time consuming if there are many steps in the branch or in the loop: each time a step is added the threat must be detected, all options to resolve it must be considered, and finally it is put in the correct context. CL-BURIDAN instead assigns each *block* a context, which is inherited by all the steps it contains. In this scheme the decision to create a conditional or loop construct is made once only, then steps are inserted into the construct's body without having to go through the algorithm's threat-resolution procedure each time.

Since iterative constructs are not present in C-BURIDAN, we will describe them in more detail. We should be clear at the outset what we mean by a loop, because the term has different interpretations. What we *don't* mean is iterating over a collection of objects (e.g. turn off every light in the house). CL-BURIDAN is a propositional planner, thus has no easy way of reasoning about collections in the first place. We implement DO-WHILE or DO-UNTIL loops, which take the form "repeat S until *it is believed* that P is true" where S is some sequence of actions. It is important to note that loop termination is based on a state of *belief* rather than the actual truth value of P. Ultimately the agent can act only on its states of belief, and its state of belief in any proposition P must be computed from the execution-time messages it has received. A better characterization of a loop would then be "repeat S until you receive messages $M_1, M_2, \ldots, M_n$ from your sensors," where receiving those messages induces a state of belief in which P is sufficiently likely to be true.

Loops are desirable for formal as well as computational reasons. Without loops, C-BURIDAN is formally incomplete: there are planning problems it cannot solve because there is no *a priori* bound on the number of steps in a solution plan. Consider the following simple example: there is a machine that (with probability 1) makes a part, but only if the machine is turned ON. The action which turns the machine ON sometimes fails, but a sensing action is available which perfectly reports on whether or not the machine is ON. The planner's task is to make a part.

Given any probability threshold $\tau < 1.0$, C-BURIDAN can construct a solution plan by inserting some fixed and bounded number of turn-on actions. But if $\tau = 1.0$, there is no solution plan of fixed and bounded length, even if a perfect sensor is available. On the other hand, the looping plan "repeatedly execute turn-on then sense-on, terminating when the sensor reports *on*, then use the machine to make the part" works with probability 1: when the loop terminates the machine is certainly ON, therefore the machine will certainly work. We cannot predict ahead of times how many iterations will actually be required to complete the task, however.

The fact that C-BURIDAN cannot represent iterated action sequences is also unfortunate computationally because it can lead to plans with many steps: if the turn-on operator is unreliable, it may have to be *attempted* many times, and each instance must explicitly be in the plan. Since the time required for both plan refinement and assessment grows with the number of steps in the plan, performance degrades.

Our method for structuring plans into nested blocks provides a natural representation for representing looping constructs as well. A loop is represented as a single block that is marked as "iterated," and in addition we specify a "termination criterion" that determines what messages from sensing actions in the block should cause an exit from the loop block. The meaning of the block's entry and exit conditions remain the same: the former is what the loop requires before execution starts, and the latter is the world state guaranteed by the loop when it terminates. Figure 5 shows how the loop in the example above would be represented. The fact that it is a loop is transparent to the rest of the plan—it participates in the plan's causal link structure just like any other plan block.

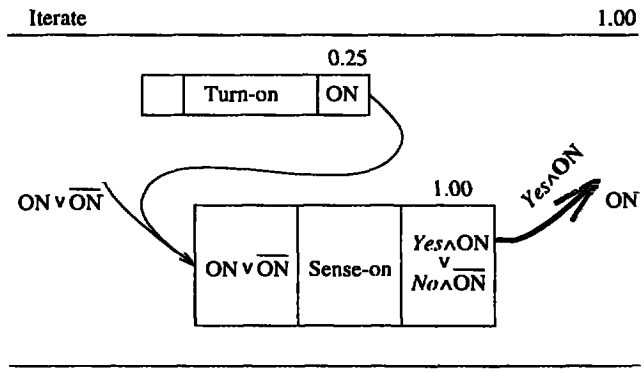

Figure 5: An iterative block. Each component block is annotated with its success probability.

The process of plan refinement does not change significantly with the addition of loop blocks: when supporting an active subgoal G the planner has the additional option of adding a looping block to the plan that has G as its exit condition, and threats are resolved in exactly the same way whether or not the threated or threatening block is a loop.

The assessment problem for looping blocks is more complicated, however. The main technical problem is determining what sequence of messages from the sensing actions in the loop will guarantee that the loop's exit condition is true with a particular probability.

The answer to this question will depend on the characteristics of the sensor of course. In certain cases like the example, a single message suffices: a single report of *YES* from the perfect sensor guarantees that the exit condition ON is true with probability 1. In the case of a sensor with state-independent noise, the termination condition is based on the *ratio* of positive messages to negative messages. The general problem—particularly situations in which the behavior of the sensor changes while the loop is being executed—is more complicated. At worst we have to reason about the entire *history* of messages generated by all sensing actions during execution of the loop. We are currently developing a taxonomy of sensor types and the corresponding termination criteria they imply.

Summary and Future Work

Our work on the BURIDAN planners has provided a decision-theoretic semantics to the planning process while maintaining the "classical" commitments to a goal-oriented problem solver, a representation based on symbolic state and operators, and a backchaining solution algorithm. Our goal is to extend significantly the range of problems to which classical planners can be applied, while at the same time exploiting the field's progress in terms of effective representations, algorithms, and control strategies.

Our immediate goals for the BURIDAN planners include exploring search-control issues, comparing our approach with other approaches to stochastic optimization, and developing a theory of planning and execution that includes both probabilistic inference in the planning process and execution-time reaction strategies.

Search control is a major problem for the BURIDAN planners, since the number of possible choices faced by the planner is staggering—much worse even than the number of choices faced by a deterministic planner (see (Kushmerick, Hanks, & Weld 1995) for some discussion). In fact the probabilistic planning problem is undecidable in the worst case—even when the state space is finite—so it is harder than deterministic planning both in theory and in practice.

Our current work on blocks represents a strong commitment to a particular form of search control in that the planner commits to subgoal decompositions (by deciding which subgoals should be assigned to separate blocks) very early in the planning process. It is our belief that search-control policies in the form of decomposition decisions can be naturally specified by a human user or generated automatically from a domain description, and that making a good decomposition decision will have a very significant effect on the algorithm's performance. We are currently working on an adaptation of the *operator graph* framework (Smith & Peot 1993) for deriving decomposition policies directly from a domain description.

The connection between the *plans* that CL-BURIDAN generates and the *policies* generated by related stochastic decision-making algorithms is intriguing. The problem solved by BURIDAN-like planners is very similar to the problem solved by Partially Observable Markov Decision Process algorithms (Cassandra, Kaebling, & Littman 1994), and with the incorporation of loops and conditionals, CL-BURIDAN plans and POMDP policies are also quite similar. A recent paper (Boutilier, Dean, & Hanks 1995) explores the similarities and differences between the two approaches to problem solving, and we will continue to try to synthesize representational and algorithmic techniques from the two fields.

Integrating planning and execution is also a priority: while an agent should be able to reason about and plan for uncertain world states ahead of time, some provision is also needed for putting off those decisions until execution time, which also implies the need to monitor the process of the plan and react appropriately when it looks like it is likely to fail. We are working on an approach to planning, monitoring, and executing that combines our probabilistic planner with a monitoring scheme based on the *action net* framework (Darwiche & Goldszmidt 1994) and the RAP (Firby 1987) execution system.

References

Boutilier, C.; Dean, T.; and Hanks, S. 1995. Planning under uncertainty: Structural assumptions and computational leverage. In *Proc. 2nd European Planning Workshop.*

Cassandra, A. R.; Kaebling, L. P.; and Littman, M. L. 1994. Acting optimally in partially observable stochastic domains. In *Proc. 12th Nat. Conf. on A.I.*, 1023-1028.

Darwiche, A., and Goldszmidt, M. 1994. Action networks: A framework for reasoning about actions and change under uncertainty. In *Proc. 10th Conf. on Uncertainty in Artifical Intelligence*, 136-144.

Draper, D.; Hanks, S.; and Weld, D. 1994a. A probabilistic model of action for least-commitment planning with information gathering. In *Proc. 10th Conf. on Uncertainty in Artifical Intelligence.*

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

Firby, R. J. 1987. An investigation into reactive planning in complex domains. In *Proc. 6th Nat. Conf. on A.I.*, 202-206.

Kushmerick, N.; Hanks, S.; and Weld, D. 1995. An Algorithm for Probabilistic Planning. *Artificial Intelligence* 76:239–286.

Smith, D., and Peot, M. 1993. Postponing threats in partial-order planning. In *Proc. 11th Nat. Conf. on A.I.*, 500–506.