Appears in the Proceedings of the Fourth European Conference on Planning
(ECP'97)

Contingency Selection in Plan Generation

Nilufer Onder[1] and Martha E. Pollack[1,2]

[1] Department of Computer Science
[2] Intelligent Systems Program
University of Pittsburgh, Pittsburgh PA 15260, USA

Abstract. A key question in conditional planning is: how many, and which of the possible execution failures should be planned for? One cannot, in general, plan for all the failures that can be anticipated: there are simply too many. But neither can one ignore all the possible failures, or one will fail to produce sufficiently flexible plans. We describe a planning system that attempts to identify the contingencies that contribute the most to a plan's overall value. Plan generation proceeds by extending the plan to include actions that will be taken in case the identified contingencies fail, iterating until either a given expected value threshold is reached or the planning time is exhausted. We provide details of the algorithm, discuss its implementation in the Mahinur system, and give initial results of experiments comparing it with the C-Buridan approach to conditional planning.

1 Introduction

Classical AI plan generation systems assume static environments and omniscient agents, and thus ignore the possibility that events may occur in unexpected ways—that contingencies might arise—during plan execution. A problem with classical planners is, of course, that things do not always go "according to plan." In contrast, universal planning systems and more recent MDP-based systems make no such assumption. They produce "plans" or "policies" that are functions from states to actions. However, the state space can be enormous, making it difficult or impossible to generate complete policies or truly universal plans.[3]

Conditional planners take the middle road. They allow for conditional actions with multiple possible outcomes and for sensing actions that allow agents to determine the current state[1, 3, 4, 5, 10, 11]. A key question in conditional planning is: how many, and which of the possible execution failures should be planned for?

In this paper, we describe Mahinur, a probabilistic partial-order planner that supports conditional planning with contingency selection. Mahinur implements an iterative refinement planning algorithm that identifies the contingencies that contribute the most to the plan's overall value, and gives priority to the contingencies whose failure would have the greatest negative impact. We concentrate on two aspects of the problem, namely, planning methods for an iterative conditional planner and a method for computing the negative impact of possible sources of failure. We present experimental data from our current implementation, and compare our system with other probabilistic conditional planners.

[3] Thus Dean et al.[2] describe an algorithm to construct an initial policy for a restricted set of states, and incrementally increase the set of states covered.

2 Planning with Contingency Selection

We start with a basic idea: a plan is composed of many steps that produce effects to support the goals and subgoals, but not all of the effects contribute equally to the overall success of the plan. Of course, if plan success is binary—plans either succeed or fail—then in some sense all the effects contribute equally, since the failure to achieve any one results in the failure of the whole plan. However, as has been noted in the literature on decision-theoretic planning, plan success is not binary. Some plans may do a better job than others of achieving the stated goals. Similarly, some contingencies that arise during plan execution may have more impact on plan outcome than other contingencies.

In this paper, we focus on the fact that the goals that a plan is intended to achieve may be decomposable into subgoals, each of which has some associated value. For example, consider a car in a service station and a plan involving two goals: installing the fallen front reflector and repairing the brakes. The value of achieving the former goal may be significantly less than the value of achieving the latter. Consequently, effects that support only the former goal (e.g., getting the reflector from the supplier) contribute less to the overall success of the plan than the effects that support only the latter (e.g., getting the calipers). Effects that support both goals (e.g., having placed a call to the supplier) will have the greatest importance. This suggests that it may be most important to have a back-up plan to handle the possibility of the supplier having already closed for the day; almost as important to have a back-up plan in case the supplier is out of calipers; and less important to have a back-up plan in case the supplier is out of reflectors. Of course, in reality the importance of having back-up plans will also depend on the likely difficulty of replanning "on-the-fly" for particular contingencies. But in this paper, we concentrate on simpler methods for calculating the impact of any contingency on plan success, assuming that all on-the-fly calculations are equally difficult. More sophisticated techniques that give up this assumption can later be folded into the process, and used within the conditional planning algorithm we present.

The high level specification of our iterative conditional planning algorithm is shown below:

1. **Skeletal plan**: `Construct a skeletal plan.`
2. **Plan refinement**: `While the plan is below the given expected value threshold:`
   - `2a. Select the contingency whose failure has the highest expected disutility.`
   - `2b. Extend the plan to include actions to cover the failure.`

The algorithm first finds a *skeletal plan*, i.e., a minimal plan with nonzero probability of success. During each iteration, a contingency whose failure will have a maximal *disutility* is selected. The plan is then extended to include actions to take in case the selected contingency fails. Iterations proceed until the expected value of the plan exceeds some specified threshold or the planning time is exhausted.

This algorithm depends crucially on the notions of contingencies and their expected values. However, before defining those, we will describe the prior approaches taken to conditional planning.

3 Prior Approaches

The literature on conditional planning mirrors the broader literature on plan generation in having both total-order and partial-order representations for plans. Total-order conditional planners include the Plinth[5] and the Weaver[1] systems. Although these systems do not directly focus on the problem of intelligently selecting contingencies to plan for, they do in practice make such selections using heuristics. The developers of the Plinth system have conjectured that, although partial-order representations are frequently more efficient in unconditional planning, they are likely to be unwieldy for conditional planning. One of our research aims has been to explore this conjecture, and we have therefore focused on partial-order approaches to conditional planning.

The most well-known probabilistic partial-order planner is C-Buridan[3]. C-Buridan constructs conditional plans in a somewhat indirect fashion. In some situations, there will be more than one operator that can achieve a goal G. To increase the support for G, C-Buridan may introduce multiple steps into the plan, only to realize that those steps conflict with one another. To resolve the conflict, it has a special mechanism for introducing an observation action. The observation action will have two possible outcomes, and C-Buridan will split the plan into two branches, associating each branch arbitrarily with one of the conflicting outcomes. Consider the example from the C-Buridan paper[3]. It involves a part which must be processed. Two alternative actions, `SHIP` and `REJECT`, both have an effect `PROCESSED`; thus, C-Buridan will insert both a `SHIP` and a `REJECT` step into the plan. Both will have a precondition of `NOT-PROCESSED`, and an effect of `PROCESSED`, and hence C-Buridan detects a traditional threat in the plan. The threat cannot be resolved by step ordering, so C-Buridan inserts an observation step into the plan, and splits the plan into two contexts. The final plan is the correct one: `CHECK-PART; SHIP IF GOOD; REJECT IF BAD`.

Unfortunately, there are three drawbacks to C-Buridan's approach of forming conditional plans. First, and most consequential to our research, in the process of constructing plans, C-Buridan does not reason about whether particular branches of conditional plans are worth constructing. It is only *after* the branch is constructed that C-Buridan recognizes it as a branch and places it into one context of an observation action. For instance, in the example above, C-Buridan first inserts the `SHIP` and `REJECT` actions into the plan, to provide added support to the `PROCESSED` goal, and only then recognizes those actions as alternative contingencies. But what is needed is to reason ahead of time about the value of addressing these particular alternatives.

Second, as is, this approach can find only a restricted range of conditional plans, and will not find those in which the observation step should follow an initial step, to determine whether a second step should be taken. For instance, suppose that the probability of a `PAINT` action succeeding is 50%. One natural way to ensure that a part has at least a 75% chance of being painted is to include two `PAINT` actions in the plan. Assume, though, that a part should be painted once (say because it will result in an overly thick coat of paint). A reasonable conditional plan would be `PAINT; CHECK-IF-SUCCESSFUL; PAINT IF NOT SUCCESSFUL`. However, for this problem, C-Buridan would insert two `PAINT` steps, and then try to resolve their incompatibility by inserting an observation step *before* them (see Sect. 7 for one fix that we used in our implementation of C-Buridan).

The third problem involves the way in which C-Buridan links the outcomes of observation steps to later actions in the plan. The alternative outcomes of observation steps are labeled with arbitrary strings, called *observation labels*, that do not relate to propositions. The planner simply has to try every possible pairing of observation labels and subsequent actions into distinct contexts, resulting in needless expansion of the search space.

4 The Mahinur System

Our approach to conditional planning combines the Buridan system[8] for managing probabilistic actions with a CNLP-style[10] approach to handling contingencies.[4] Given the problems outlined in the previous section, we needed to keep several features of the CNLP system that C-Buridan had chosen to modify. We also added mechanisms for intelligent contingency selection.

Unlike the systems discussed above, CNLP does not include mechanisms for reasoning about probabilistic outcomes. But it does construct conditional plans to account for foreseen sources of uncertainty. That is, CNLP actions may have multiple possible outcomes. When the planner uses one of the outcomes of such an action to establish a precondition, it realizes that the goal cannot be achieved in every possible completion of the plan. To resolve this, it duplicates the goal, and marks it so that it can only be supported by actions that do not depend on the outcome that was just used. The actions that are inserted to support the new goal then constitute the "other" branch of the conditional plan.[5]

4.1 Actions and Contingencies

We build on the representation for probabilistic actions that was developed for Buridan[8, p. 247]. *Causal actions* are those that alter the state of the world. For instance, the `PAINT` action depicted both graphically and textually in Fig. 1 has the effects of having the part painted (`PA`) and blemishes removed (`~BL`) 95% of the time when the part has not been processed (`~PR`).

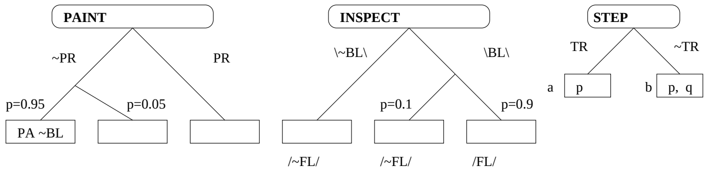
Definition 1. A **causal action** `N` is defined by a set of *causal consequences*:

$$\mathtt{N} : \{< t_1, \rho_{1,1}, e_{1,1} >, \ldots, < t_1, \rho_{1,l}, e_{1,l} >, \ldots,$$
$$< t_n, \rho_{n,1}, e_{n,1} >, \ldots, < t_n, \rho_{n,m}, e_{n,m} >\} \ .$$

For each i, j: t_i is an expression called the consequence's *trigger*, and $e_{i,j}$ is a set of literals called the *effects*. $\rho_{i,j}$ denotes the probability that the effects in $e_{i,j}$ will be produced given that the literals in t_i hold. The triggers are mutually exclusive and exhaustive. For each effect e of an action, $RESULT(e, st)$ is a new state that has the negated propositions in e deleted from, and the non-negated propositions added to st.

[4] C-Buridan also builds on Buridan; hence its name.

[5] With CNLP, the branches do not rejoin, whereas with C-Buridan they can. Methods for rejoining branches have not been explored in the context of CNLP style algorithms.



PAINT: { <{~PR},0.95,{PA,~BL}>, <{~PR},0.05,{}>,<{PR},1,{}> }

INSPECT: { <\~BL\,{},1,{},/~FL/>, <\BL\,{},0.1,{},/~FL/>, <\BL\{},0.9,{},/FL/> }

Fig. 1. Example actions.

Conditional plans also require *observation actions* so that the executing agent will know which contingencies have occurred, and hence, which actions to perform. Observational consequences are different from causal consequences in that they record two additional pieces of information: the *label* (marked with //) shows *which proposition's* value is being reported on, the *subject* (marked with \\) shows *what the sensor looks at*. The subject is needed when an aspect of the world is not directly observable and the sensor instead observes another (correlated) aspect of the world. For instance, a robot `INSPECT`ing a part might look at whether it is blemished (\\`BL`\\), to report whether it is flawed (/`FL`/). Note that the label and the subject are not arbitrary strings—they are truth-functional propositions.

Definition 2. An **observation action** `N` is a set of *observational consequences*:

$$\begin{aligned}\mathtt{N}: \{&< sbj_1, t_1, \rho_{1,1}, e_{1,1}, l_{1,1} >, \ldots, < sbj_1, t_1, \rho_{1,l}, e_{1,l}, l_{1,l} >, \ldots, \\ &< sbj_n, t_n, \rho_{n,1}, e_{n,1}, l_{n,1} >, \ldots, < sbj_n, t_n, \rho_{n,m}, e_{n,m}, l_{n,m} >\} \ .\end{aligned}$$

For each i, j: sbj_i is the *subject* of the sensor reading, t_i is the *trigger*, $e_{i,j}$ is the set of any *effects* the action has, and $l_{i,j}$ is the label that shows the *sensor report*. $\rho_{i,j}$ is the probability of obtaining the effects and the sensor report. The subjects and triggers are mutually exclusive and exhaustive. In our framework we wanted to keep the ability to represent faulty sensors as introduced by C-Buridan, by representing the relationship between the subject and the report. But, unlike a trigger, a subject proposition is not meant to be achieved by the planner.

An important issue in the generation of probabilistic plans is the identification of equivalent consequences. We define *contingencies* to be subsets of outcomes of probabilistic actions. For instance, if we paint a part, we may or may not succeed in having the part painted properly. Having the part painted is one contingent outcome of the paint action; not having it is another. A deterministic action is simply one with a single (contingent) outcome. The contingencies of a step are relative to the step's intended use. For instance, if the rightmost step in Fig. 1 is used to support `p`, then {a,b} is its single contingency, but if it is used to support `q`, then {a} and {b} are alternative contingencies.

4.2 Plan Representation

We define a planning problem in the usual way, as a set of initial conditions encoded as a start step (`START`: $\{< \emptyset, \rho_1, e_1 >, \ldots, < \emptyset, \rho_n, e_n >\}$), a goal (`GOAL`:

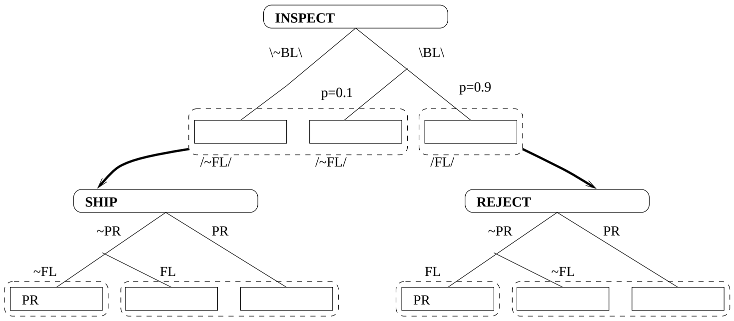


Fig. 2. Observation links.

$\{<\{g_1,\ldots,g_n\},1,\emptyset>\}$), and an action library. We assume that each top-level goal g_i has value $val(g_i)$. A plan in our framework has two sets of steps and links (causal and observational), along with binding and ordering constraints.

Definition 3. A **partially ordered plan** is a 6-tuple, $< T_c, T_o, O, L_c, L_o, S >$, where T_c and T_o are causal and observation steps, O is a set of temporal ordering constraints, L_c and L_o are sets of causal and observation links, and S is a set of open subgoals. It is important to note that both types of links emanate from contingencies rather than individual outcomes. For instance, in Fig. 2, `SHIP` will be executed if `INSPECT` reports /~FL/, and `REJECT` will be executed otherwise.

Definition 4. A **causal link** is a 5-tuple $< S_i, c, p, c_k, S_j >$. Contingency c of step S_i is the link's *producer*, contingency c_k of step S_j is the link's *consumer*, and p is the *supported proposition* ($S_i, S_j \in T_c \cup T_o$).

Definition 5. An **observation link** is a 4-tuple $< S_i, c, l, S_j >$. Contingency c is the collection of consequences of $S_i \in T_o$ that report l. An observation link means that $S_j \in T_c \cup T_o$ will be executed only when the sensor report provided by S_i is l.

Often the planner needs to reason about all the steps that are subsequent to a particular observation. We therefore define a *composite step* as a portion of the plan graph rooted at an observation action. Although composite steps are an important part of our overall framework we will omit the formal definitions for brevity in this paper (See [9] for the general definitions).

4.3 Expected Disutility

The importance of planning for the failure of any particular contingency can be seen to be a factor of two things: the probability that the contingency will fail, and the degree to which the failure will affect the overall success of the plan. To compute the former, one needs to know both the probability that the action generating the particular contingency will be executed, and the conditional probability of the contingency given the action's occurrence. Computing the latter is

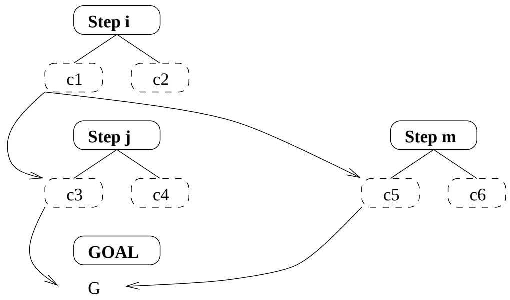


Fig. 3. Computing expected disutility.

more difficult. In this paper, however, we make two strong simplifying assumptions. First, we assume that the top-level goals for any planning problem can be decomposed into utility-independent[6] subgoals having fixed scalar values. Second, we assume that all failures are equally difficult to recover from at execution time. With these two assumptions, we can compute the *expected disutility* of a contingency's failure: it is the probability of its failure, multiplied by the sum of the values of all the top-level goals it supports. Planning for the failure of a contingency c means constructing a plan that does not depend on the effects produced by c, i.e., a plan that will succeed even if c fails.

The main idea in computing the expected value of a contingency c is to find out which top-level goal(s) will fail if the step fails to produce c. For example, suppose that contingency c_1 of step S_i establishes a proposition p for contingency c_3 of step S_j and for contingency c_5 of step S_m (Fig. 3). Suppose also that the trigger expressions for c_3 and c_5 contain other propositions which will become true with probability k_3 and k_5 just before their respective steps are executed. Then, the support provided along the path that has more probability of success is more valuable. We call *k_i the probability that c_i will support the goal.* If a contingency directly supports a goal, then k is 1. As an example, suppose that c supports top-level goal g_i in two ways: along one path with probability 0.7, and along another with probability 0.8. If c fails, the most that will be lost is the support for g_i with probability 0.8, and thus the expected value of c is $0.8 \times val(g_i)$. Therefore, while propagating the values to a contingency, we take the maximum support it provides for each top-level goal.

Definition 6. Expected value of a contingency: Suppose that contingency c has outgoing causal links to consumer contingencies $c_1, \ldots, c_n$ and possibly an outgoing observation link to step S. Assume that k_i is the probability that c_i supports goal g, and k_{n+1} is the probability that S supports g. Then, the expected value of c with respect to g is:

$$evg(c, g) = \begin{cases} val(g), \text{if } c \text{ directly supports } g, \\ max(k_1, \ldots, k_{n+1}) \times val(g), \text{otherwise.} \end{cases}$$

[6] The utility of one subgoal does not depend on the degree to which other goals are satisfied[6].

For z top-level goals, the expected value of c is, $EV(c) = \sum_{i=1}^{z} evg(c, g_i)$.

We next consider the computation of expected disutility of a contingency's failure. To begin, we need to compute the probability that a given state will occur after the execution of a sequence of actions.[7]

Definition 7. The **probability of a state after a (possibly null) sequence of steps is executed**:

$$P[st'|st, <>] = \{1, \text{if } st' = st; 0, \text{otherwise.}\}$$

$$P[st'|st, < S >] = \sum_{<t_i, \rho_{i,j}, e_{i,j}> \in S} \rho_{i,j} P[t_i|st] P[st'|RESULT(e_{i,j}, st)]$$

$$P[st'|st, < S_1, \ldots, S_n >] = \sum_{u} P[u|st, < S_1 >] P[st'|u, < S_2, \ldots, S_n >]$$

where S_1 is a causal step.[8] The action sequence is a total ordering consistent with the partially ordered plan.

The expected disutility of a contingency c is the product of the probability that the action generating c will be executed, the conditional probability of c's failure given the action's occurrence, and the expected value of c.

Definition 8. Expected disutility of the failure of a contingency: Let c be a contingency of step S_i. If the condition for c is $< \{t_1, \rho_1\}, \ldots, \{t_m, \rho_m\} >$, then the expected disutility of c is defined as

$$P[S_i \text{ is executed}] \times (1 - (\sum_{k=1}^{m} \rho_k P[t_k| < S_1, \ldots, S_{i-1} >])) \times EV(c) \ .$$

5 The Planning Algorithm

We can now see how the algorithm in Sect. 2 works. After forming a skeletal plan, it selects a contingency c whose failure has maximal expected disutility. Suppose that c has an outgoing causal link $< S_i, c, p, c_k, S_j >$ (Fig 4, left). Because the proposition supported by c is p, the planner selects an observation action, S_o, that reports the status of p and inserts it ordered to come after S_i. Two contingencies for S_o are formed: c_1 contains all of S_o's outcomes that produce the label $/p/$, and c_2 contains all the other outcomes. The causal link $< S_i, c, p, c_k, S_j >$ is removed and an observation link $< S_o, c_1, /p/, S_j >$ is inserted to denote that S_j will be executed when the sensor report is $/p/$. In addition, a causal link, $< S_i, c, sbj, c_1, S_o >$ is inserted (Fig. 4, right). This link prevents the insertion of an action that would perturb the correlation between the subject and label propositions. Conditional planning then proceeds in the CNLP style: the goal is duplicated and labeled so that it cannot receive support from the steps that depend on contingency c_1 of the observation step.

[7] The probability of an expression e with respect to a state st is defined as: $P[e|st] = \{1, \text{if } e \subseteq st; 0, \text{otherwise.}\}$.

[8] We will be omitting the formal definition with the observation steps for brevity. The main idea is to recursively calculate the result of a sequence of steps for each branch.

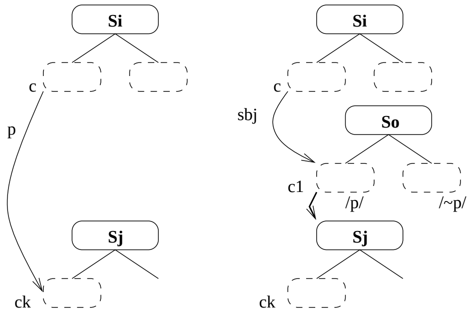


Fig. 4. Inserting a new observation action.

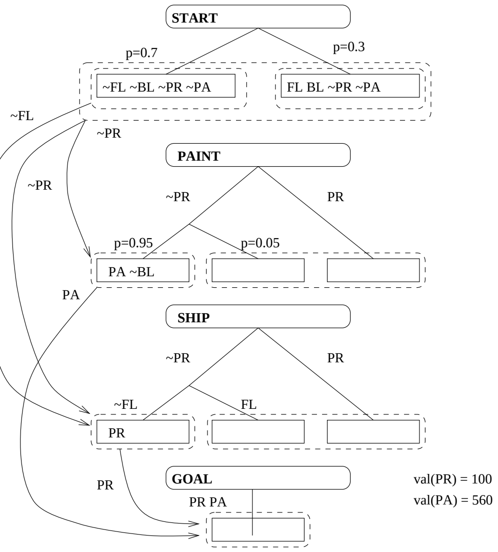


Fig. 5. The skeletal plan.

6 Example

The following example is based on the example presented in the C-Buridan paper. The goal is to process(PR) and paint(PA) parts. Initially, 30% of the parts are flawed(FL) and blemished(BL), the rest are in good condition. The "correct" way of processing a part is to reject it if it is flawed, and to ship it otherwise. We assign 100 as the value of processing a part, and 560 as the value of painting. The action library contains the SHIP, REJECT, INSPECT and PAINT actions.

The Mahinur planning system starts by constructing a skeletal plan, and nondeterministically chooses the SHIP action to process the part. Two contingencies of SHIP are constructed (shown in Fig. 5 in dashed boxes): the first one produces PR, and the second does not. A causal link that emanates from the first contingency is established for PR.

The triggers for the first contingency are adopted as subgoals (~PR,~FL), and both are supported by START. For START, two different sets of contingencies are constructed: one with respect to ~PR, and one with respect to ~FL. When PAINT is inserted to support the PA goal, and the ~PR trigger of the first contingency is supported by START, the skeletal plan is complete. The skeletal plan contains three contingencies whose disutilities of failure must be calculated (the contingency for ~PR of START has no alternatives). The expected disutilities for the first contingencies of START, PAINT, and SHIP are $0.3 \times 100 = 30$ (for ~FL), $0.05 \times 560 = 28$ (for PA), and $0.3 \times 100 = 30$ (for PR), respectively.

Mahinur chooses to handle the first contingency with the higher expected disutility, namely the first contingency of START. Because ~FL is the proposition supported by that contingency, an action that inspects the value of FL is inserted before SHIP (Fig. 6). The causal link supporting ~FL is removed and an observation link from the ~FL report is inserted. The INSPECT step looks at BL to report about FL. Therefore, the first contingency of START is linked to INSPECT to ensure that an action that alters the value of BL cannot be inserted between START and INSPECT (BL and FL are correlated). The new branch is then completed. For the duplicated goal, a new REJECT action is inserted to support PR, and the existing PAINT step is used to support PA yielding the plan shown in Fig. 6. If the success threshold has not been met, Mahinur will plan for additional contingencies. Note that existing conditional planners can solve this problem, but the plan generated will be the same regardless of the value assigned to the top-level goals. Within our framework, the planner is sensitive to the values assigned to each goal and is able to focus on different parts of the plan based on the disutility of contingencies.

7 Implementation and Experiments

Mahinur is implemented using C-LISP Common Lisp on a PC running Linux. We conducted preliminary experiments on two sets of problems. In the first set, we used synthetic problems that had an increasing number of goals, and thus an increasing number of potential failures (Problem 1 has one goal and one action that can fail, problem 2 has two goals and one action that can fail for each goal, and so on). For each problem, Table 1 shows the total run time for solving the problem and the amount of time used for disutility calculation as the expected utility threshold is increased and more iterations of conditional planning are performed.[9] As expected, runtime increases exponentially with the size of the problem; disutility calculation, however, is insignificant.

For the second set of experiments (Table 2), we implemented the C-Buridan algorithm on top of Buridan. We used the same support functions for conditional planning (context propagation, checking context compatibility) and the same ranking function as in Mahinur. To fix the problem we discussed in Sect. 3, namely, the problem of C-Buridan always inserting observation actions before both of the conflicting actions, we introduced a threat resolution mechanism that nondeterministically places only one of the conflicting actions into a context of an observation step. We implemented the Mahinur algorithm in two ways:

[9] The problem marked with a * terminated with a LISP out-of-memory error.

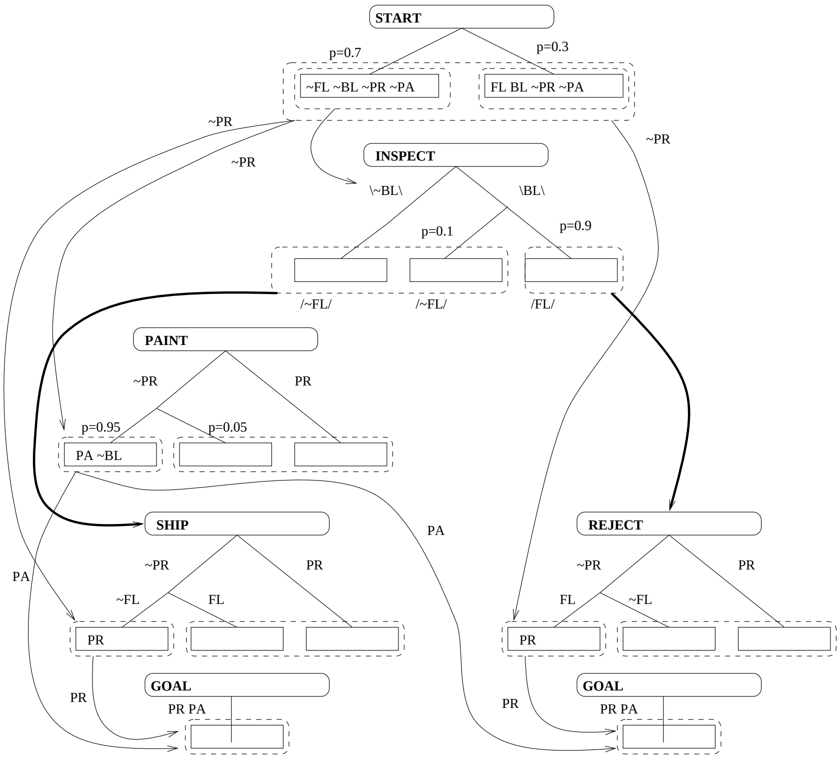


Fig. 6. The complete plan.

	Problem 1		Problem 2		Problem 3	
Iterations	Run	Disutility	Run	Disutility	Run	Disutility
0	0.01	0.01	0.03	0.01	0.08	0.01
1	0.14	0.01	0.56	0.01	3.92	0.02
2	0.42	0.01	11.28	0.02	335.21	0.03
3	1.67	0.02	229.22	0.03	4368.43	0.05
4	9.11	0.03	3558.87	0.04	*	-

Table 1. Run time and disutility calculation time in seconds (experiment 1).

Mahinur(a) forms contingencies but puts a plan in the search space for each outcome in a contingency; Mahinur(b) selects one of the outcomes randomly.[10]

To demonstrate the advantage of directly planning for contingencies, we used a domain which contains a single `PAINT` operator. Painting a part repeatedly increases the probability of success by 0.5^n, where n is the number of repetitions. It is an error to paint a part if it succeeded on the last try, thus the plan has to contain an observation action to check whether the part was painted, and repeat

[10] The experiments marked with a * terminated when the search limit of 20,000 nodes was reached.

Problem	C-Buridan	Mahinur (a)	Mahinur (b)
PAINT (t=0.75)	184	33	16
PAINT (t=0.875)	17,341	187	49
PAINT (t=0.9375)	*	1018	131
PAINT (t=0.9688)	*	5851	247
PAINT (t=0.9844)	*	*	512
SHIP-REJECT	*	250	40
TIREWORLD	*	278	47
TRANSPORT PACKAGE	1,190	52	40
TREATMENT	*	22	14

Table 2. Number of nodes visited (experiment 2).

the `PAINT` step only if it has not been successful. Comparing the second column of results (C-Buridan) to the first one (Mahinur(a)) in Table 2, we see that Mahinur expanded significantly fewer nodes as the probability threshold(t) is increased. This is because the planning process concentrates on directly planning for contingencies. In contrast, the problem quickly became hard for C-Buridan because of additional branching that results from overly frequent attempts to handle conflicts by conditional branching. We observed a similar explosion of the search space on the `SHIP-REJECT` problem, a standard textbook problem on conditional planning (`TIREWORLD`[12]), and examples from conditional planning literature (`TRANSPORT PACKAGE`[1] and `TREATMENT`[7]).

Comparing the last column to the second column displays a second advantage of forming contingencies. The search space is decreased by heuristically selecting one outcome in a contingency rather than generate a plan for each outcome.

It is important to note that the disutility calculation we use is heuristic: we consider only the intended use of each action (as shown by the causal links) rather than the exact contribution to the plan, and we consider one total ordering for the disutility calculation. In $EV(c)$, we summarize the contribution of c to the achievement of each goal. One alternative to our method of calculation is to evaluate the exact probability of a goal failing given that c is not achieved. We will empirically compare the heuristic and exact approaches in upcoming work.

8 Summary and Related Work

We presented an approach to partial-order conditional planning that directly reasons about the value of planning for alternative contingencies. Other relevant work includes the recent decision-theoretic planning literature, e.g., the Pyrrhus system[13] which prunes the search space by using domain-specific heuristic knowledge, and the DRIPS system[7] which uses an abstraction hierarchy.

Our approach combines the Buridan mechanism for managing probabilistic actions with a CNLP-style of conditional branch formation and adds techniques for identifying contingencies and computing their effect on overall plan success. In this paper, we concentrated on constructing conditional branches as a method of increasing the probability of success of a plan. Our next step is to incorporate methods for reducing the probability of failure by adding more supporting actions. Our conjecture is that by concentrating on the "right" contingency using the disutility calculation, the planner will be more flexible to produce better plans quickly. Experimental assessment with a much larger "real-world" domain and extensions to utilize richer utility functions will be forthcoming.

Acknowledgments. This work has been supported by a scholarship from the Scientific and Technical Research Council of Turkey, by the Air Force Office of Scientific Research (Contract F49620-96-1-0403), by an National Science Foundation Young Investigator's Award IRI-9258392, and by National Science Foundation Award IRI-9619579.

References

1. Jim Blythe and Manuela Veloso. Analogical replay for efficient conditional planning. In *Proc. 14th Nat. Conf. on Artificial Intelligence*, pages 668–773, 1997.
2. T. Dean, L. P. Kaelbling, J. Kirman, and A. Nicholson. Planning under time constraints in stochastic domains. *Artificial Intelligence*, 76:35–74, 1995.
3. Denise Draper, Steve Hanks, and Daniel Weld. Probabilistic planning with information gathering and contingent execution. In *Proc. 2nd Int. Conf. on AI Planning Systems*, pages 31–36, 1994.
4. Oren Etzioni, Steve Hanks, Daniel Weld, Denise Draper, Neal Lesh, and Mike Williamson. An approach to planning with incomplete information. In *Proc. 3rd Int. Conf. on Principles of Knowledge Repr. and Reasoning*, pages 115–125, 1992.
5. Robert P. Goldman and Mark S. Boddy. Epsilon-safe planning. In *Proc. 10th Conf. on Uncertainty in Artificial Intelligence*, pages 253–261, 1994.
6. Peter Haddawy and Steve Hanks. Utility models for goal-directed decision-theoretic planners. Technical Report 93-06-04, Department of Computer Science and Engineering, University of Washington, 1993.
7. Peter Haddawy and Meliani Suwandi. Decision-theoretic refinement planning using inheritance abstraction. In *Proc. 2nd Int. Conf. on AI Planning Systems*, pages 266–271, 1994.
8. Nicholas Kushmerick, Steve Hanks, and Daniel S. Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, 76:239–286, 1995.
9. Nilufer Onder and Martha E. Pollack. Contingency selection in plan generation. In *AAAI Fall Symp. on Plan Execution: Problems and Issues*, pages 102–108, 1996.
10. Mark A. Peot and David E. Smith. Conditional nonlinear planning. In *Proc. 1st Int. Conf. on AI Planning Systems*, pages 189–197, 1992.
11. Louise Pryor and Gregg Collins. Planning for contingencies: A decision based approach. *Journal of Artificial Intelligence Research*, 4:287–339, 1996.
12. Stuart J. Russell and Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, Englewood Cliffs, NJ, 1994.
13. Mike Williamson and Steve Hanks. Optimal planning with a goal-directed utility model. In *Proc. 2nd Int. Conf. on AI Planning Systems*, pages 176–181, 1994.