Incremental Contingency Planning

Richard Dearden*, Nicolas Meuleau†, Sailesh Ramakrishnan†, David E. Smith & Rich Washington*
NASA Ames Research Center
Mail stop 269–2
Moffet Field, CA 94035–1000, USA
{dearden, nmeuleau, sailesh,de2smith, richw}@email.arc.nasa.gov

Abstract

There has been considerable work in AI on planning under uncertainty. However, this work generally assumes an extremely simple model of action that does not consider continuous time and resources. These assumptions are not reasonable for a Mars rover, which must cope with uncertainty about the duration of tasks, the energy required, the data storage necessary, and its current position and orientation.

In this paper, we outline an approach to generating contingency plans when the sources of uncertainty involve continuous quantities such as time and resources. The approach involves first constructing a "seed" plan, and then incrementally adding contingent branches to this plan in order to improve utility. The challenge is to figure out the best places to insert contingency branches. This requires an estimate of how much utility could be gained by building a contingent branch at any given place in the seed plan. Computing this utility exactly is intractable, but we outline an approximation method that back propagates utility distributions through a graph structure similar to that of a plan graph.

1 Introduction

For a Mars rover, daily operation is rife with uncertainty. There is inherent uncertainty about the duration of tasks, the energy required, the data storage necessary, position and orientation, and environmental factors such as soil characteristics, dust on the solar panels, ambient temperature, etc. For example, in driving from one location to another, the amount of time required depends on wheel slippage and sinkage, which varies depending on slope, terrain roughness, and soil characteristics. All of these factors also influence the amount of energy that is consumed. The amount of energy collected by the solar panels during a traverse depends on the length of the traverse, but also on the angle of the solar panels. This is dictated by the slope and roughness of the terrain.

Since rover operations are often highly constrained by time and energy constraints, plans that do not take this uncertainty into account often fail miserably. Based on the telemetry logs, we estimate that the Mars Pathfinder rover spent a substantial amount of its life doing nothing because of either plan failure or conservative action sequences constructed to avoid any possibility of plan failure. One way to attack this problem is to do on-board replanning when failures occur. While this capability is certainly desirable, there are several difficulties with exclusive reliance on this approach:

- Rovers have severely limited computational resources due to energy limitations and radiation hardening requirements. As a result, it is not always feasible to do timely or significant onboard replanning.
- Many actions are potentially risky and require preapproval by mission operations personnel. Because of the cost and difficulty of communication, the rover receives infrequent command uplinks (typically one per day). As a result, each daily plan must be constructed and checked for safety well in advance.
- Some contingencies require anticipation. For example, switching to a backup system may require that the backup system be warmed up in advance. For time critical operations there is insufficient time to perform these setup operations once the contingency has occurred, no matter how fast the planning can be done.

For these reasons, it is sometimes necessary to plan in advance for potential contingencies; that is, anticipate unexpected outcomes and events and plan for them in advance. In this paper we will be concerned with ground-based contingency planning for rovers. More precisely, the problem is to produce a (concurrent) plan with maximal expected utility, given the following domain information:

- A set of possible goals that may be achievable, each of which has a value or reward associated with it.
- A set of initial conditions, which may involve uncertainty about continuous quantities like temperature, energy available, solar flux, and position. This uncertainty is characterized by probability distributions over the possible values.
- A set of possible actions, each of which is characterized by:
  - a set of conditions that must be true before the action can be performed. (These may include metric temporal constraints and constraints on resource availability.)

* Research Institute for Advanced Computer Science (RIACS)
† QSS Group Inc.

– an uncertain duration characterized by a probability distribution.
– a set of certain and uncertain effects that describe the world following the action. Uncertain effects on continuous variables are characterized by probability distributions.

Contingency planning is already known to be quite hard both in theory [29] and in practice. However, there are some characteristics of this domain that make this planning problem different and even more difficult:

Time - actions take differing amounts of time and concurrency is often necessary.

Continuous outcomes - most of the uncertainty is associated with continuous quantities like time and energy. In other words, actions do not have a small number of discrete outcomes.

Problem size - a typical daily plan for a rover will involve on the order of a hundred actions.

As a result of these characteristics, it is not clear how to apply previous approaches to planning under uncertainty to this problem. In this paper, we outline a much different approach to this problem. At the top level, the approach involves 1) constructing a "seed" plan, and 2) incrementally adding contingent branches to this plan in order to improve utility. The challenge is to figure out the best places to insert contingency branches. In general, this requires an estimate of how much utility could be gained by building a contingent branch at any given place in the seed plan. Computing this utility exactly is intractable, but we outline an approximation method that involves back propagating utility distributions through a graph structure similar to that of a plan graph. In Section 2 we discuss *Just-in-Case Planning*, our incremental approach to contingency planning based on the *Just-in-Case Scheduling* work of Drummond, et al [16].We also argue that for planning, probability of failure is not a good heuristic for choosing branch points. In Section 3 we describe our plan graph method for estimating branch utility curves.

2 Just-In-Case Planning

In the classical approach to contingency planning, each time an action with uncertain outcomes is added to a plan, the planner attempts to establish the goals for each different outcome of the action. Unless there are only a few discrete sources of uncertainty in a domain, this approach is completely impractical. For more complex domains, it is critical that the planner focus on those contingencies that will make a large difference in the overall value of the plan. To do this, we build upon the Just-In-Case (JIC) scheduling technique [16], that was initially developed for contingency scheduling of automated observatories. The basic idea in the JIC approach is to take a seed schedule, look for the place where it is most likely to fail, and augment the schedule with a contingent branch at that point. The process is repeated until the resulting contingent schedule is sufficiently robust, or until available time is exhausted. This process is illustrated in Figure 1.

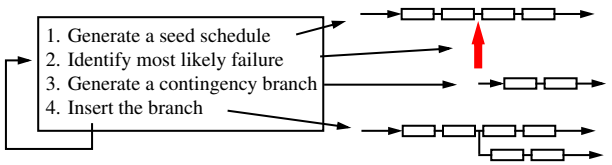

Figure 1: The JIC approach.

Conceptually, it seems straightforward to apply the JIC approach to planning problems. Using a conventional planner, we first generate a seed plan assuming the expected behavior of each activity; in other words, we reason as if every action uses the expected amount of time and resources. This is the same approach taken in JIC scheduling. As with JIC scheduling, we then choose a place to insert a contingency branch. Once again, using a conventional planner, we generate a plan for the contingency branch and add it to the existing plan.[1]

2.1 The JIC Branch Heuristic

For JIC planning, the tricky part is deciding where to insert contingency branches, and what the branch conditions should be. In Drummond et al.'s original implementation for automatic telescope scheduling, branches are added at the points *with the greatest probability of failure*. Given the distributions for time and resource usage this is relatively easy to calculate by statistical simulation of the plan. Unfortunately, the points most likely to fail are not necessarily good points for contingent branches. Consider the example in Figure 2 where we have a seed plan with two actions, A_1 and A_2, leading to a goal G that has positive value. Initially we have 20 units of some resource (say energy) and each of the actions consumes somewhere between 5 and 15 units of the resource. Clearly, this plan is most likely to fail after (or during) action A_2. However, if the plan fails after (or during) action A_2, there will not be any resources left. If all the alternative activities require some of this resource, then there is no point in putting a contingent branch after A_2.

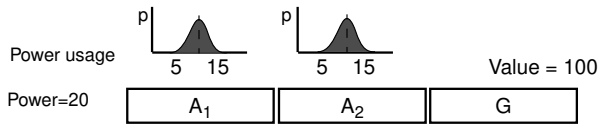

Figure 2: Example showing that the place where the plan is most likely to fail may not be the best branch point.

Fundamentally, the problem is that in order to select the best place to insert a branch, we need to know whether or not it is possible to accomplish anything useful at the points under consideration. More precisely, we need to know how

[1] Just as with JIC scheduling, this process is not guaranteed to converge to an optimal contingent plan. However, JIC will always monotonically improve a plan until a local optimum is reached.

much utility could be gained by inserting a branch at each given point. In order to do this, we need to know the *value function* of the mainline plan and of each possible branch. The value function gives the expected future reward (utility) at each step of a plan, as a function of the resource levels.

Computing the value function for a completed plan (such as the seed plan) is relatively straightforward. It may be done analytically if the resource consumptions for activities are simple distributions. However, more typically, Monte Carlo simulation is required [3, 45]. Similarly, it is easy to get an estimate of the probability distribution over resources at each step of a plan. A crucial piece of information is then *the value function of the best branch plan that can be added at each point in the existing plan*. If we had this information, we could easily determine the optimal branch point in the plan. We would just have to compare the relative gain in utility obtained by considering the best possible branch plan at each point and pick the branch point where this gain is maximal.

Our extended JIC algorithm finds the best branch point using the following approach. For each possible branch point in the current plan:

1. Calculate the value function (as a function of available resources) of the remaining plan as well as the probability distribution of resource availability at that point (using Monte Carlo simulation).
2. Estimate the value function of the best branch that can be added to the plan at this point, using the procedure described in Section 3. This procedure also partitions the value function for a branch according to the set of goals contributing to the expected value; we can thus determine the set of goals responsible for the branch's estimated value.
3. Calculate the net utility gain for adding the best branch plan, as described in Section 3.6. The best overall branch point is the one with the maximum net utility gain. The branch condition is the condition that defines the region where the value function of the branch is greater than that of the current plan.

We generate the contingency branch using the same planner as for the seed plan, setting the initial conditions equal to the branch condition, and providing the set of goals pursued by the optimal branch. Note that the steps for estimating the value of a branch do not actually construct the branch plan.

Unfortunately, there is no easy way to calculate the exact value function for the best possible branch plan at a given point; that would require actually doing the planning for the branch. Instead we must approximate this value function. In the next section, we present a procedure designed to estimate the value function of the best possible branch plan that could be generated at each point, without actually doing the planning.

3 Estimation of Branch Utility

A critical part of our algorithm is to compute an estimate of the value function of possible branch plans, at each candidate branch-point of the mainline plan. It is based on a representation of the planning problem as a plan graph [4]. Graphplan is a classical planning algorithm that first performs a reachability analysis by constructing a plan graph, and then performs goal regression within this graph to find a plan. Our approach retains only the first of these stages, the plan graph construction. We then perform back-propagation of utility tables in the graph to produce estimates of utility functions (instead of plans). This section provides an outline of this mechanism.

3.1 The Plan Graph

The plan graph is a sequential graph that alternates propositional (fluent) levels and action levels. Each propositional level contains the set of propositions that can be made true at that level, and a set of mutual exclusion (mutex) constraints between pairs of these propositions. A mutex between two fluents indicates that these propositions cannot both be true at the same time at this level of the graph. The first propositional level contains all the fluents that are true in the initial state of the problem (initial conditions). The action levels contain all the actions that can be applied given the previous propositional level. Each action has an arc from each fluent that it consumes and an arc to each fluent it produces.

Figure 3 shows a part of the plan graph obtained in a simple example where the only continuous variable is energy. In this problem, the mainline plan (shown in bold) consists of two actions: A which takes the fluent p as precondition and produces q and r, and B which has q as precondition and g as effect. The fluent g represents a goal that provides a reward (utility) of 5. For actions A and B, the expected consumption is 10 Ah, and they can be started only if the current level of resource is at least 15 Ah. Three other actions, C, D, and E, are available in the domain, but they are not included in the mainline plan. The fluent g' represents a secondary goal with utility 1. Finally, both p and s are true and all the other fluents are false in the initial conditions. There are two points of the mainline plan that are candidate branch points: at the beginning of the plan, and between A and B. The latter is characterized by the following set of propositions: p, q, r and s (all other fluents being false). Our goal is to estimate the best utility gain we can get by branching at these points.

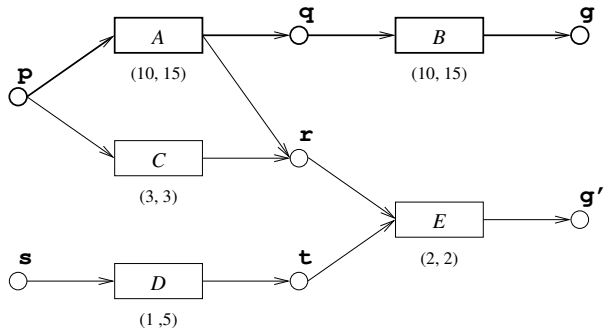

Figure 3: An example plan graph (partial). The two numbers below each action represent, first, its expected consumption, and second, the minimum energy required to start the action.

3.2 Utility Table Back-propagation

The basic principle of our algorithm is to back-propagate utility distribution tables in the plan graph back to the initial state. Each table is attached to a single (action or proposition) node and contains a piecewise constant function giving utility as a function of resource level e (e.g. energy). It represents an estimate of the expected reward we can get by performing this action, or by having this fluent true, as a function of current resource levels.

The process is initialized by creating utility tables for the goals. In our example, we start with a table for g and an expected return of 5 for positive resource levels (and 0 otherwise), indicating that we obtain a reward of 5 if we can get to g with some energy remaining.

We then back-propagate this table in the plan graph, until it has reached the initial conditions. First, a table is created for action B, based on the table in g. Its utility function is defined by:

$$V_B(e) = \left\{ \begin{array}{ll} 0 & \text{if } e < 15 \ ; \\ V_g(e-10) & \text{otherwise} \ ; \end{array} \right\} \quad (1)$$

where $V_B(e)$ and $V_g(e)$ are the (piecewise constant) utility estimates encoded by the tables in B and g respectively. The first line expresses the fact that we are not allowed to start B if the current energy is at or below 15 Ah. The second says that B consumes 10 Ah and leads to g, from where we can get the reward encoded by V_g. Next, the table attached to B is back-propagated to the previous propositional level. Since B has only one fluent, q, as precondition, the table in B is copied *as is* in q. Similarly, we back-propagate the table at q to A and to p. The resulting value function is

$$V_p(e) = V_A(e) = \left\{ \begin{array}{ll} 0 & \text{if } e < 25 \ ; \\ V_g(e-20) & \text{otherwise} \ ; \end{array} \right\} \quad (2)$$

3.3 Conjunctive Preconditions

The process described above—that is, regressing the goal g down to fluent p—is relatively simple because there is no action with multiple preconditions in the path. A more complex situation arises when we try to regress g' to initial conditions, since the tables must pass through action E, which has conjunctive preconditions.

To deal with this type of situation, we add to each utility table a *condition*, that is, a list of fluents such that the table is valid if and only if all the fluents are true. The fluents in the condition list of a table represent the set of goals that must be achieved, together with the fluent (or action) carrying the table, to get the utility encoded in the table. The condition is used to remember what subgoals are still to be achieved when we back-propagate a table.

In our example, we start by creating a utility table at g' with an empty condition and predicting a reward of 1 for all $e > 0$. This table is then back-propagated to action E, as shown in Figure 4. Since E has two fluents as preconditions, r and t, two copies of its table are created, one for each fluent node. The value functions encoded by these tables are both equal to the function of the table in E ($V_r(e) = V_t(e) = V_E(e)$). However, their conditions are different.

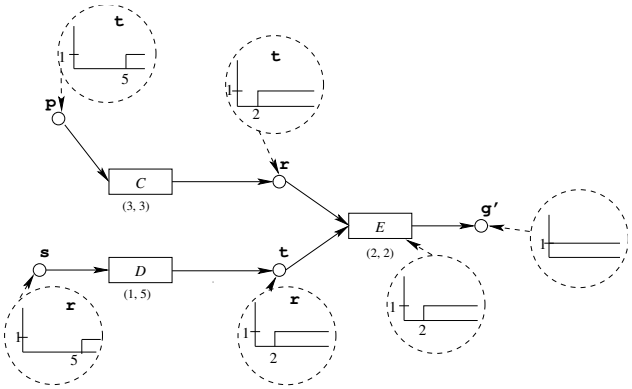

Figure 4: Utility table propagation for conjunctive preconditions

The condition of the table for r is $\{t\}$, while the condition of the table for t is $\{r\}$. Both tables indicate that a utility of 1 may be obtained if we can get s and t at the same time with a remaining energy of at least 2. However, they represent different orderings of the two sub-goals: the table attached to node r estimates the utility that can be gained by producing first t and then r, and the table in node t considers subgoals in the opposite order.

When a table reaches the beginning of the plan graph but still has unsatisfied fluents in its condition, a copy of it is posted at each node in the condition list that is not satisfied. For instance, the table in t with $\{r\}$ as condition is back-propagated through action D to the s node. This table predicts 0 reward if $e \leq 5$ (the energy requirement for D), and 1 otherwise. A copy of the table is then posted at the r node, and r is deleted from its condition list (which is now empty). It will in turn be back-propagated to p by applying the consumption of action C. This results in a table that predicts 0 reward if $e \leq 8$, and 1 otherwise.

Similarly, the table in r obtained by regressing goal g' through action E can be back-propagated to the p node. This table predicts 0 reward if $e \leq 5$, (the consumption of C plus the consumption of E) and 1 otherwise. Since it has $\{t\}$ as condition, it is then sent to the t node and t is deleted from its condition. Finally, it is back-propagated through C to s, giving a table that predicts 0 reward if $e \leq 6$ and 1 otherwise. We thus have two tables with empty conditions originating from g': one at p and one at s. Both encode the same set of actions, $\{C, D, E\}$, but the table at p represents the ordering D, C, E, while the table at s represents the ordering C, D, E.[2] In this case, the table at p is strictly better than the table at s. This is because the high energy requirement for D makes it advantageous to do D first, before any of the energy is consumed by C.

This propagation process may lead to several tables attached to the same node, since there may be several ways

[2]When doing back-propagation, we only consider a few of the possible orderings, namely the ones where the conjuncts are established serially. Although it would be possible, we do not consider orderings that involve interleaving the actions for establishing the conjuncts.

to support a fluent. The total number of tables is limited by merging all the tables that have the same condition at some node: they are replaced by a single table that encodes the maximum of all their value functions.

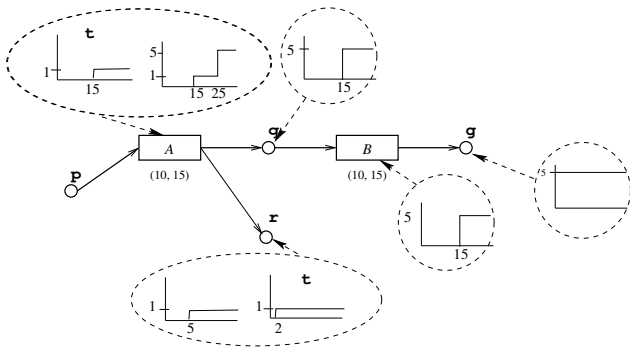

Figure 5: Utility table propagation for conjunctive effects

3.4 Conjunctive Effects

An interesting step in the back-propagation mechanism is illustrated by action A in Figure 5. Since this action has two effects, it will receive utility tables from both effects.

In general, each time a table is back-propagated to an action, we merge it with all other tables at the action that have exactly the same conditions. In our example, we have one unconditional table at q, and two tables at r, one with the condition t, and one that is unconditional. The table with condition t is back-propagated through A as usual. However, the two unconditional tables from q and r are merged by taking the maximum. The resulting merged table is then back-propagated through A. The value function for this table is defined by:

$$V_A(e) = \begin{cases} 0 & \text{if } e < 15 \ ; \\ \max \left\{ \begin{array}{l} V_q(e-10), \\ V_r(e-10) \end{array} \right\} & \text{otherwise .} \end{cases} \quad (3)$$

which in this case becomes:

$$V_A(e) = \begin{cases} 0 & \text{if } e < 15 \\ 1 & \text{if } 15 \leq e < 25 \\ 5 & \text{otherwise .} \end{cases} \quad (4)$$

Using the max operator to combine tables represents a rather pessimistic view, since it assumes that we can never obtain the reward from more than one table in any given plan. In our example, the reward in the table at q comes from the goal g, whereas, the reward in the tables at r come from g'. Intuitively, it would seem that if we had enough energy ($e \geq 30$) we should be able to achieve one goal, then the other. To deal with situations where several goals are reachable, we need to use a more complex operator that considers the sum of the rewards for two tables. To do this correctly, we need to augment the tables with some additional information: (i) the sum of the expected consumptions (C) of the actions performed to get the utility encoded by the table, and (ii) the goals that are responsible for each segment of the utility in the table. They both are a function of the resource level e and are piecewise constant like the utility function.

Using this information, in the case of action A, we have

$$V_A(e) = \begin{cases} 0 \text{ if } e < 15 \ ; \\ \max \left\{ \begin{array}{l} V_q(e-10) + \\ \quad V_r(e-10-C_q(e-10)), \\ V_r(e-10) + \\ \quad V_q(e-10-C_r(e-10)) \end{array} \right\} \\ \text{otherwise .} \end{cases} \quad (5)$$

The first of the two expressions in the maximum represents performing A, pursuing the goals beyond q, and then the goals beyond r. The second expression follows the same reasoning, but pursuing r then q. The information about the goals pursued is used to avoid counting the same goal twice. If the goals pursued in the two tables (for a given resource level) intersect, then we use a simple max rule as in (3).

Using equation (5) instead of (3) to evaluate the initial step of the mainline plan in our example, we would have identified the possibility of reaching both g and g' if there are sufficient initial resources. In particular, we get the table:

$$V_A(e) = \begin{cases} 0 & \text{if } e < 15 \\ 1 & \text{if } 15 \leq e < 25 \\ 5 & \text{if } 25 \leq e < 30 \\ 6 & \text{otherwise .} \end{cases} \quad (6)$$

Finally, note that the max operator in equations (3) and (5) represents the opportunity to make different choices with different levels of resource after executing A. It does *not* reflect the possibility to make choices during execution *as a function of the outcomes of previous actions*. In fact, the utility table back-propagation procedure works on a deterministic approximation of the problem: we suppose that each activity consumes exactly its expected consumption, with probability 1. Therefore, there is no uncertain outcome after executing A. Rather, we evaluate the utility that could be obtained for different initial resource levels, and thus different intermediate resource levels. The apparent choice due to the max in the previous equation represents the opportunity to do different things with different initial conditions. We are actually estimating the value of an *unconditional* plan *for each possible starting condition*.

3.5 Extracting Utility Estimates

Once the utility tables have been back-propagated to the fluents representing initial conditions of the problem, we extract the utility estimates for the candidate branch points from the graph. For the example in Figure 3, consider the point characterized by the initial set of fluents $\{p, q\}$. We build a single utility table for this branch point by merging all utility tables attached to p and q whose condition is included in the set $\{p, q\}$ (that is, whose condition is true in that state). This is all the tables that represent utility apparently reachable when p and q are true simultaneously. These tables are merged using the max operator of equation (3) or (5). The resulting table is the value function estimate that we need.

As shown in Figure 6, the calculation for the branch point at the beginning of the mainline plan uses two tables (all others are dominated):

- the table attached to p with empty condition and showing that a reward of 1 may be obtained if $8 \leq e < 25$, and a reward of 5 may be reached if $e \geq 25$;
- the table attached to s with $\{p\}$ as condition and showing a reward of 1 may be obtained if $e \geq 6$ (the sum of the consumptions of C, D and E).

The resulting table, which characterizes this branch point, shows that no reward can be obtained from here if $e < 6$, that a reward of 1 is available if $6 \leq e < 25$, and that a reward of 5 may be obtained if $e \geq 25$.

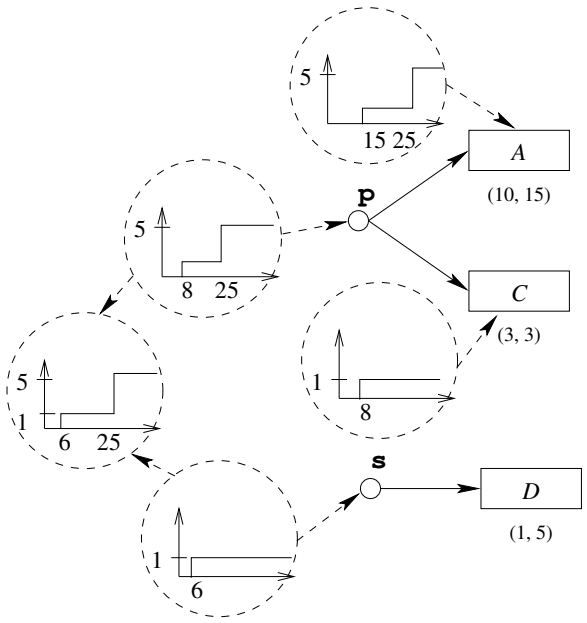

Figure 6: Extracting utility estimates (using the MAX operator)

3.6 Using Utility Estimates

Given the utility estimates at the various branch points, we can now use this information to select the branch point, the branch condition and the set of goals to pursue. For a particular branch point, we compute the gain in utility by taking the difference between the estimated utility curve for the branch, and the utility for the tail of the mainline plan. We must also attenuate this by the probability distribution over resources at the branch point. More precisely, the net utility gain for a branch point is given by:

$$\text{Gain} = \int_0^\infty P(r) \max\{0, V_b(r) - V_m(r)\} dr$$

where $P(r)$ is the probability distribution over the resources at the branch point, $V_b(r)$ is the estimated utility curve for the branch, and $V_m(r)$ is the utility curve for the tail of the mainline plan.

The branch condition is given by the places where the branch and mainline utility curves cross each other. Where the branch curve is higher, it is worthwhile to take the branch, otherwise not. The goals for the contingent branch are the goals that correspond to the portion of the branch utility estimate that is greater than the utility curve of the mainline plan.

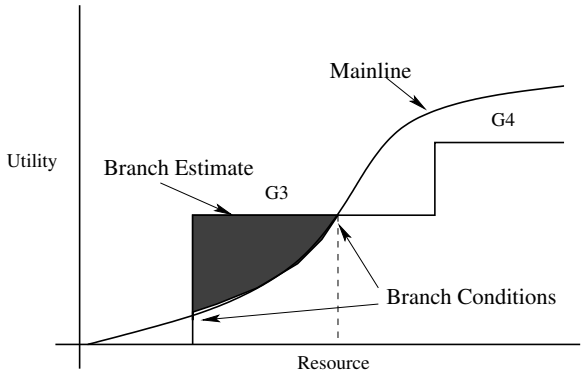

Figure 7: Selecting the branch point, branch condition and goals

For example, in Figure 7, we show the mainline utility curve and the branch estimate curve for a branch point. The shaded area represents the utility gain for the branch. The branch conditions are shown and the goal corresponding to the utility gain is G3.

4 Related Work

4.1 Contingency Planning

There has been considerable work in AI on planning under uncertainty. Table 1 classifies much of this work along the following two dimensions:

Representation of uncertainty: whether uncertainty is modeled strictly logically, using disjunctions, or is modeled numerically, with probabilities.

Observability assumptions: whether the uncertain outcomes of actions are not observable, partially observable, or fully observable.

There are a number of difficulties in attempting to apply existing work on planning under uncertainty to spacecraft or rovers. First of all, the work listed in Table 1 assumes a very simple model of action in which concurrent actions are not permitted, explicit time constraints are not allowed, and actions are considered to be instantaneous. None of these assumptions hold for typical spacecraft or rover operations. These characteristics are not as much of an obstacle for Partial-Order Planning frameworks such as SENSp [17], PUCCINI [20], WARPLAN-C [48], CNLP [37], Buridan [28], UDTPOP [36], C-Buridan [15], DTPOP [36], Mahinur [35] and Weaver [6]. In theory, these systems could represent plans with concurrent actions and complex temporal constraints. The requirements for a rich model of time and action are more problematic for planning techniques that are

	Disjunction	Probability
Non Observable	CGP [44] CMBP [12, 1] C-PLAN [11, 18] Fragplan [27]	Buridan [28] UDTPOP [36]
Partially Observable	SENSp [17] Cassandra [38] PUCCINI [20] SGP [49] QBF-Plan [40] GPT [7] MBP [2]	C-Buridan [15] DTPOP [36] C-MAXPLAN [30] ZANDER [30] Mahinur [35] POMDP [9]
Fully Observable	WARPLAN-C [48] CNLP [37]	JIC [16] Plinth [21] Weaver [6] PGP [5] MDP [9]

Table 1: A classification of planners that deal with uncertainty. Planners in the top row are often referred to as conformant planners, while those in the other two rows are often referred to as contingency planners or conditional planners.

based on the MDP or POMDP representations, satisfiability encodings, the graphplan representation, or state- space encodings. These techniques rely heavily on a discrete model of time and action. (See [43] for a more detailed discussion of this issue.) Although semi-Markov decision processes (SMDPs) [39] and temporal MDPs (TMDP) [10] can be used to represent actions with uncertain durations, they cannot model concurrent actions with complex temporal dependencies. The factorial MDP model has recently been developed to allow concurrent actions in an MDP framework. However, this model is limited to discrete time and state representations. Moreover, existing solution techniques are either too general to be efficient on real-world problems (e.g. Singh and Cohn [41]), or too domain-specific to be applicable to the rover problem (e.g. Meuleau et al. [31]).

A second, and equally serious, problem with existing contingency planning techniques is that they all assume that uncertain actions have a small number of discrete outcomes. To characterize where a rover could end up after a move operation, we have to list all the different possible discrete locations. We would need to do something similar to characterize energy usage. For most spacecraft and rover activities this kind of discrete representation is impractical since most of the uncertainty involves continuous quantities, such as the amount of time and energy an activity requires. Action outcomes are distributions over these continuous quantities. There is some recent work using models with continuous states and/or action outcomes in both the MDP [3, 32, 33, 42] and POMDP [46] literature, but this has not yet been applied to SMDPs and has primarily been applied to reinforcement learning rather than planning problems.

A third problem with conventional contingency planning technology is that it does not scale to larger problems. Part of the problem is that most of the algorithms attempt to account for all possible contingencies. In effect, they try to produce policies. For spacecraft and rover operations, this is not realistic or tractable—a daily plan can involve on the order of a hundred operations, many of which have uncertain outcomes that can impact downstream actions. The resulting plans must also be simple enough that they can be understood by mission operators, and it must be feasible to do detailed simulation and validation on them in a limited time period. This means that a planner can only afford to plan in advance for the "important" contingencies and must leave the rest to run-time replanning. Of the planning systems in table 1, only Just-In-Case (JIC) contingency scheduling [16] and Mahinur [35] exhibit a principled approach to choosing what contingencies to focus on.

4.2 Utility Estimation

A number of authors working in probabilistic planning have looked at the problem of estimating the utility of a plan. Haddawy et al. [23, 22] developed the DRIPS planner, which attempts to optimize the utility of the plan it returns. DRIPS differs from our approach in that it represents uncertainty about the utility of a plan as an interval, which allows it to determine when one plan dominates another, taking into account utility uncertainty rather than only expected utility. However, DRIPS plans in a very restricted domain represented as an abstraction/decomposition network that constrains the plans that can be created. Comparing abstractions of plans built using this network allows DRIPS to discard plans without having to estimate the utility of a partial plan, just as we do.

Another related system is Blythe's Weaver [6]. Weaver builds contingency plans in an incremental fashion very similar to our appraoch, although it only considers actions with discrete outcomes (and uncontrolled external actions). Weaver is concerned with adding contingencies to increase the probability of success of the plan, rather than the utility.[3] It translates the current plan into a Bayesian network to compute the probability of success of the plan, and identifies points where an action can lead to a failure as candidates for new contingent branches. Unlike our approach, it does not attempt to estimate the value of the contingent branch before adding it. As a result, it may add branches at points where it is not possible to measurably improve the overall probability of success of the plan.

4.3 Plan graph Heuristics

Haslum & Geffner [24], Nguyen et al. [34], and others have described ways of using a plan graph to derive distance heuristics for guiding planning search. Such heuristics are now used in a wide variety of different planning systems, including state space search planners such as HSP2 [8], FF [25], and AltAlt [34], Graphplan-based systems [26], and partial-order planning systems such as RePop [14] and VHPOP [50]. Recently, Do [13] has reported on the use of plan graphs for deriving more complex "cost functions". These

[3] Littman et al. [29] have shown that it is possible to translate problems involving utility into probabilistic planning problems, but it is not clear how practical it is to do this for realistic problems.

cost functions are used to guide a planner in selecting actions when both time and resource usage are important.

Our use of plan graphs is somewhat different than these previous efforts. In particular, we are using the plan graph to derive estimates of the utility that can be achieved from different possible states (rather than estimates of the "difficulty" of achieving a particular goal from the initial conditions). This allows us to make intelligent choices about the goals to be achieved in a contingent branch. Because we are dealing with uncertainty in resource availability, we must also propagate utility functions through the plan graph, rather than single numbers.

5 Discussion

5.1 Complexity

The problem of conditional planning is known to be very difficult ($\Pi_P^2 Hard$ [40], Σ_P^3 [47]). Here, it is combined with the problem of goal selection, which is intrinsically exponential in the size of the number of goals, since all subsets of goals must be considered. Our algorithm avoids part of the complexity of the problem by:

- incrementally adding contingency branches to the plan (rather than considering sets of possible branches)
- using a plan graph based heuristic to select branch points
- using this heuristic to select goal sets for planning.

The computation of the heuristic information is expensive, because of the complexity of propagating value tables through the plan graph. However, this propagation only needs to be done once for a particular problem, rather than once for each possible branch point being considered. To do this, we simply construct the plan graph for the entire problem, and propagate all value tables and conditions back to the intitial conditions. For a particular branch point, the relevant tables are those associated with the conditions that hold at the branch point. In other words, a branch point corresponds to a set of internal nodes and tables in the plan graph.

5.2 Status and Future Work

We are currently implementing the techniques described in this paper for a Mars Smart Lander Technology Demonstration Effort. To build seed plans and branch plans, we are using the EUROPA planning system [19] developed at NASA Ames Research Center.

The incremental contingency planning algorithm that we have described is not guaranteed to produce an optimal contingency plan, because it makes greedy choices concerning branch selection, branch condition, branch goals and branch plan. In theory, we could make it complete by allowing backtracking on all these choices. However, since a branch condition involves continuous quantities, this would be somewhat problematic.

Our approach of using a plan graph to estimate value functions makes a number of assumptions and simplifications:

- We propagate expected values, rather than distributions through the plan graph.
- We consider only a few of the many possible orderings for achieving subgoals in the plan graph. (Order does not matter unless there are time or resource constraints on the execution of individual actions.)
- We ignore negative interactions (mutual exclusion) between actions in the plan graph.
- We ignore shared substructure between subgoals in the plan graph

We are considering ways to relax some of these assumptions, but doing so increases the computational cost of the heuristic. It is not yet clear how these assumptions will affect the quality of the resulting contingency plans. It may turn out that we are better off using the weaker heuristic presented here, and doing more search over the space of contingency branches.

Finally, we are also considering a number of other difficult issues such as:

- permitting sensory actions to consume time and resources. (Here we made the MDP assumption that the sensory information was always free and available.)
- allowing sensory information to be noisy
- allowing the possibility of inserting setup actions for contingent branches prior to the actual branch point

It appears that all of these issues can be addressed in our plan graph methods for computing utility tables, but it is non-trivial to do so.

Acknowledgements Thanks to Dan Weld and several anonymous reviewers for discussions on this subject and comments on drafts of the paper. This research was supported by NASA Ames Research Center and the NASA Intelligent Systems program.

References

[1] P. Bertoli, A. Cimatti, and M. Roveri. Heuristic search + symbolic model checking = efficient conformant planning. In *Proc. of the 17th Intl. Joint Conf. on Artificial Intelligence*, 2001.

[2] P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Planning in mondeterministic domains under partial observability via symbolic model checking. In *Proc. of the 17th Intl. Joint Conf. on Artificial Intelligence*, 2001.

[3] D.P. Bertsekas and J.N. Tsitsiklis. *Neuro-dynamic Programming*. Athena, Belmont, MA, 1996.

[4] A. Blum and M. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997.

[5] A. Blum and J. Langford. Probabilistic planning in the Graphplan framework. In *Proc. of the 5th European Conf. on Planning*, 1999.

[6] J. Blythe. *Planning under uncertainty in dynamic domains*. PhD thesis, Carnegie Mellon University, 1998.

[7] B. Bonet and H. Geffner. Planning with incomplete information as heuristic search in belief space. In *Proc. of the 5th Intl. Conf. on Artificial Intelligence Planning and Scheduling*, pages 52–61, 2000.

[8] Blai Bonet and Héctor Geffner. Planning as heuristic search: New results. In Susanne Biundo and Maria Fox, editors, *Proc. of the 5th European Conf. on Planning*, pages 360–372. Springer-Verlag, 1999.

[9] C. Boutillier, T.L. Dean, and S. Hanks. Decision theoretic planning: structural assumptions and computational leverage. *Journal of AI Research*, 11:1–94, 1999.

[10] J.A. Boyan and M.L. Littman. Exact solutions to time-dependent MDPs. In *Advances in Neural Information Processing Systems 13*, pages 1–7. MIT Press, Cambridge, 2000.

[11] C. Castellini, E. Giunchiglia, and A. Tacchella. Improvements to sat-based conformant planning. In *Proc. of the 6th European Conf. on Planning*, 2001.

[12] A. Cimatti and M. Roveri. Conformant planning via symbolic model checking. *Journal of AI Research*, 11:305–338, 2000.

[13] Minh Binh Do and Subbarao Kambhampati. Planning graph-based heuristics for cost-sensitive temporal planning. In *Proc. of the 6th Intl. Conf. on Artificial Intelligence Planning and Scheduling*, pages 3–12, 2001.

[14] Minh Binh Do and Subbarao Kambhampati. SAPA: A domain-independent heuristic metric temporal planner. In *Proc. of the 6th European Conf. on Planning*, 2001.

[15] D. Draper, S. Hanks, and D. Weld. Probabilistic planning with information gathering and contingent execution. In *Proc. of the Second Intl. Conf. on Artificial Intelligence Planning and Scheduling*, pages 31–36, 1994.

[16] M. Drummond, J. Bresina, and K. Swanson. Just-In-Case scheduling. In *Proc. of the Twelfth National Conf. on Artificial Intelligence*, pages 1098–1104, 1994.

[17] O. Etzioni, S. Hanks, D. Weld, D. Draper, N. Lesh, and M. Williamson. An approach to planning with incomplete information. In *Proc. of the Third Intl. Conf. on Principles of Knowledge Representation and Reasoning*, pages 115–125, 1992.

[18] E. Ferraris and E. Giunchiglia. Planning as satisfiability in nondeterministic domains. In *Proc. of the 17th National Conf. on Artificial Intelligence*, 2000.

[19] Jeremy Frank and Ari Jónsson. Constraint-based attribute interval planning. *Constraints, Special Issue on Planning*, 2003. To appear.

[20] K. Golden. Leap before you look: information gathering in the PUCCINI planner. In *Proc. of the Fourth Intl. Conf. on Artificial Intelligence Planning and Scheduling*, pages 70–77, 1998.

[21] R. Goldman and M. Boddy. Conditional linear planning. In *Proc. of the Second Intl. Conf. on Artificial Intelligence Planning and Scheduling*, pages 80–85, 1994.

[22] Richard Goodwin. Using loops in decision-theoretic refinement planners. In B. Drabble, editor, *Proc. of the Third Intl. Conf. on Artificial Intelligence Planning and Scheduling*, pages 118–124. AAAI Press, 1996.

[23] Peter Haddawy, AnHai Doan, and Richard Goodwin. Efficient decision-theoretic planning: Techniques and empirical analysis. In *Proc. of the Eleventh Conf. on Uncertainty in Artificial Intelligence*, pages 229–236, Montreal, 1995.

[24] Parik Haslum and Héctor Geffner. Admissible heuristics for optimal planning. In *Proc. of the 5th Intl. Conf. on Artificial Intelligence Planning and Scheduling*, pages 140–149. AAAI Press, 2000.

[25] Jórg Hoffmann and Bernhard Nebel. The FF planning system: fast plan generation through heuristic search. *Journal of AI Research*, 14:253–302, 2001.

[26] Subbarao Kambhampati and Romeo Sanchez Nigenda. Distance-based goal-ordering heuristics for Graphplan. In *Proc. of the 5th Intl. Conf. on Artificial Intelligence Planning and Scheduling*, pages 315–322. AAAI Press, 2000.

[27] J. Kurien, P. Nayak, and D. Smith. Fragment-based conformant planning. In *Proc. of the 6th Intl. Conf. on Artificial Intelligence Planning and Scheduling*, 2002.

[28] N. Kushmerick, S. Hanks, and D. Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, 76(1-2):239–286, 1995.

[29] M. Littman, J. Goldsmith, and M. Mundhenk. The computational complexity of probabilistic planning. *Journal of AI Research*, 9:1–36, 1998.

[30] S. Majercik and M. Littman. Contingent planning under uncertainty via stochastic satisfiability. In *Proc. of the 16th National Conf. on Artificial Intelligence*, 1999.

[31] N. Meuleau, M. Hauskrecht, K. Kim, L. Peshkin, L. Kaelbling, T. Dean, and C. Boutilier. Solving very large weakly coupled Markov decision processes. In *Proc. of the Fifteenth National Conf. on Artificial Intelligence*, pages 165–172, 1998.

[32] R. Munos. Variable resolution discretization for high-accuracy solutions of optimal control problems. In *Proc. of the 16th Intl. Joint Conf. on Artificial Intelligence*, 1999.

[33] R. Munos. A study of reinforcement learning in the continuous case by the means of viscosity solutions. *Machine Learning*, 40:265–299, 2000.

[34] XuanLong Nguyen, Subbarao Kambhampati, and Romeo Nigenda. Planning graph as the basis for deriving heuristics for plan synthesis by state space and CSP search. *Artificial Intelligence*, 135(1–2):73–123, 2002.

[35] N. Onder and M. Pollack. Conditional, probabilistic planning: A unifying algorithm and effective search control mechanisms. In *Proc. of the 16th National Conf. on Artificial Intelligence*, pages 577–584, 1999.

[36] M. Peot. *Decision-Theoretic Planning*. PhD thesis, Dept. of Engineering-Economic Systems, Stanford University, 1998.

[37] M. Peot and D. Smith. Conditional nonlinear planning. In *Proc. of the First Intl. Conf. on Artificial Intelligence Planning and Scheduling*, pages 189–197, 1992.

[38] L. Pryor and G. Collins. Planning for contingencies: a decision-based approach. *Journal of AI Research*, 4:287–339, 1996.

[39] M.L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, New York, NY, 1994.

[40] J. Rintanen. Constructing conditional plans by a theorem prover. *Journal of AI Research*, 10:323–352, 1999.

[41] S.P. Singh and D. Cohn. How to dynamically merge Markov decision processes. In *Advances in Neural Information Processing Systems 11*. MIT Press, Cambridge, 1998.

[42] W. Smart and L. Kaelbling. Practical reinforcement learning in continuous spaces. In *Proc. of the 17th Intl. Conf. on Machine Learning*, 2000.

[43] D. Smith, J. Frank, and A. Jónsson. Bridging the gap between planning and scheduling. *The Knowledge Engineering Review*, 15(1), 2000.

[44] D. Smith and D. Weld. Conformant graphplan. In *Proc. of the Fifteenth National Conf. on Artificial Intelligence*, pages 889–896, 1998.

[45] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[46] S. Thrun. Monte Carlo POMDPs. In *Advances in Neural Information Processing Systems 12*, pages 1064–1070. MIT Press, Cambridge, 1999.

[47] Hudson Turner. Polynomial-length planning spans the polynomial hierarchy. In *Proc. of Eighth European Conf. on Logics in Artificial Intelligence (JELIA'02)*, page to appear in, 2002.

[48] D. Warren. Generating conditional plans and programs. In *Proc. of the Summer Conf. on AI and Simulation of Behavior.*, 1976.

[49] D. Weld, C. Anderson, and D. Smith. Extending Graphplan to handle uncertainty and sensing actions. In *Proc. of the Fifteenth National Conf. on Artificial Intelligence*, pages 897–904, 1998.

[50] Háakan Younes and Reid Simmons. On the role of ground actions in refinement planning. In *Proc. of the 6th Intl. Conf. on Artificial Intelligence Planning and Scheduling*, pages 54–61, 2001.