From: AAAI Technical Report WS-98-02. Compilation copyright © 1998, AAAI (www.aaai.org). All rights reserved.

A Unifying Algorithm for Conditional, Probabilistic Planning

Nilufer Onder
Dept. of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260
nilufer@cs.pitt.edu

Martha E. Pollack
Dept. of Computer Science and
Intelligent Systems Program
University of Pittsburgh
Pittsburgh, PA 15260
pollack@cs.pitt.edu

John F. Horty
Institute for Advanced
Computer Studies
University of Maryland
College Park, MD 20742
horty@umiacs.umd.edu

Abstract

Several recent papers describe algorithms for generating conditional and/or probabilistic plans. In this paper, we synthesize this work, and present a unifying algorithm that incorporates and clarifies the main techniques that have been developed in the previous literature. Our algorithm decouples the search-control strategy for conditional and/or probabilistic planning from the underlying plan-refinement process. A similar decoupling has proven to be very useful in the analysis of classical planning algorithms, and we suspect it can be at least as useful here, where the search-control decisions are even more crucial. We describe an extension of conditional, probabilistic planning, to provide candidates for decision-theoretic assessment, and describe the reasoning about failed branches and side-effects that is needed for this purpose.

Introduction

Several recent papers describe algorithms for generating conditional and/or probabilistic plans. In this paper, we synthesize this work, and present a unifying algorithm that incorporates and clarifies the main techniques that have been developed in the previous literature. Our algorithm decouples the search-control strategy for conditional and/or probabilistic planning from the underlying plan-refinement process. A similar decoupling has proven to be very useful in the analysis of classical planning algorithms (Weld 1994), and we suspect it can be at least as useful here, where the search-control decisions are even more crucial.[1]

Our algorithm relies on three techniques for dealing with a plan with branching actions:

- *corrective repair*, introduced in the work on conditional planning, which involves reasoning about what to do if the desired outcome of the branching action does not occur,
- *preventive repair*, introduced in the work on probabilistic planning, which involves reasoning about how to help ensure that the desired outcome of the branching action will occur; and
- *replacement*, which is implemented by backtracking in the planning literature, and involves removing the branching action and replacing it with an alternative.

We use our algorithm to describe an extension of conditional, probabilistic planning—namely, providing candidates for decision-theoretic assessment—and we describe the reasoning about failed branches and side-effects that is needed for this purpose.

Background

When a planning agent does not have complete knowledge of the environment in which its plans will be executed, it may have to create a *conditional plan*, which includes *observation steps* to ascertain the unknown conditions. Using an example based on that of Peot and Smith (Peot & Smith 1992), we can imagine a planning problem in which the goal is to get go skiing, but the planning agent does not know whether the road leading from the highway to the skiway is open. The plan formed thus involves driving on the highway to the road in question, observing whether it is open, and if so, continuing on to the skiway and go skiing.

Conditional planning systems (Warren 1976; Goldman & Boddy 1994a; Peot & Smith 1992; Pryor & Collins 1996) generate plans that have *branching actions*, i.e., actions with (at least) two possible outcomes. One of these outcomes (the *desired outcome*) will be linked to a later step on the path to the goal, while the other(s) (the *undesired outcomes*) will not. We will also refer to an unlinked outcome as a *dangling edge*.[2] In the skiing example, the knowledge that the road is open is the desired

[1] We do not directly discuss search-control strategies in this paper, but see our preceding work (Onder & Pollack 1997).

[2] To simplify presentation, the authors of the systems under discussion have focused on observation actions with binary outcomes. We will follow this practice here.

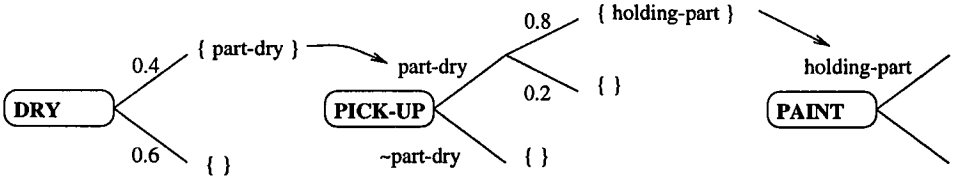

Figure 1: Plan for picking up a part.

outcome, while knowledge that it is closed is the undesired outcome. The plan is guaranteed to succeed if the desired outcomes of all its observation actions occur; there is no such guarantee otherwise.

Intuitively, one way to improve such a plan is to figure out what to do if some observation step has an undesired outcome. We will call this a *corrective repair*, since it involves figuring out actions that can be taken to correct the situation that results after the undesired outcome occurs. For the skiing example, one corrective repair might be to drive further down the highway and try a different road to the skiway. In practice, conditional planners implement corrective repairs by duplicating the goal state, and attempting to find a plan that will achieve the (duplicated) goal state without relying on the assumption that any of the observation actions in the original plan have their desired outcomes.

A different approach to the problem of uncertainty during planning is taken in probabilistic planners. Where conditional planners assume that agents have no information about the probability of alternative action outcomes, but will be able to observe their environments during plan execution, probabilistic planners such as Buridan (Kushmerick, Hanks, & Weld 1995) make just the opposite assumption: they assume that planning agents have knowledge of the probabilities that their actions will have particular outcomes, but that they will be unable to observe their environment. Typically, actions are modeled with a finite set of tuples $< t_i, p_{i,j}, e_{i,j} >$, where the t_i are a set of exhaustive and mutually exclusive *triggers*, and $p_{i,j}$ represents the probability that the action will have effect $e_{i,j}$ if t_i is true at the time of the action's execution. The triggers serve the role of preconditions in standard POCL planners. In the example plan fragment shown in Fig. 1, the `PICK-UP` step has been inserted to achieve the goal of holding the part, and there is thus a causal link from that to the goal. The trigger for `holding-part` is `part-dry`, and a `DRY` step has been inserted to probabilistically make that true.

As can be seen, this plan is not guaranteed to succeed. Consider the possible failures that may be related specifically to the `PICK-UP` step. If the part is not dry, the step will fail to achieve the desired outcome of holding the part. Intuitively, a planner might therefore help to prevent this undesired outcome from occurring, by increasing the probability that the part is dry. One way to do this is to add a second `DRY` step. We can call this a *preventive repair*, since it involves adding actions that help prevent the undesired outcome.

Even if one or more of the `DRY` steps have their desired outcomes and the part is dry at the time of the pick-up, there is still no guarantee that the pick-up will be successful. Nothing can be done to increase the probability that the pick-up will succeed if the part is dry, but of course, preventive repairs could be performed for the `PAINT` step that requires `holding-part`, e.g., by adding a second `PICK-UP` step.

It is only natural to combine the ideas of conditional and probabilistic planning, since often a planning agent both will have prior knowledge of the probabilities associated with the outcomes of its actions, and will be able to observe its environment during plan execution. [3]

The first combined conditional, probabilistic planning system was C-Buridan (Draper, Hanks, & Weld 1994). Interestingly, while C-Buridan uses preventive repair to increase the probability of success, it does not use corrective repair to generate conditional branches. Instead, its branches are formed in a somewhat indirect fashion, and result only from detecting a conflict between two actions that have both been introduced to support some condition. The Plinth conditional-planning system was also expanded to perform probabilistic reasoning (Goldman & Boddy 1994b). The focus of this project was on using a belief network to reason about correlated probabilities in the plan. Another system designed by Goldman and Boddy can create plans that achieves the goals in all the possible cases without using observation actions (conformant planning)(Goldman & Boddy 1996).

Two more recent systems that combine conditional and probabilistic planning are Weaver (Blythe & Veloso 1997) and Mahinur (Onder & Pollack 1997). Both these systems more closely follow the general model described above: they pro-

[3] The Just-In-Case algorithm (Drummond, Bresina, & Swanson 1994) involves creating an initial schedule and building contingent schedules for the points that are most likely to fail. We focus on planning algorithms in this paper.

```
PLAN (init, goal, T)
 plans ← { make-init-plan ( init, goal ) }

 while plan-time < T and plans is not empty do
   CHOOSE (and remove) a plan P from plans
   SELECT a flaw f from P.
   add all refinements of P to plans:
      plans ← new-step(P, f) ∪ step-reuse(P, f)
                                          if f is an open condition,
      plans ← demote(P, f) ∪ promote(P, f) ∪ confront(P, f) ∪ add-observe-link(P, f)
                                          if f is a threat.
      plans ← corrective-repair(P, f) ∪ preventive-repair (P, f)
                                          if f is a dangling-edge.
 return (plans)


preventive-repair (plan, f)
 open-conditions-of-plan ← triggers for the desired outcomes of the
     action in f.
 return (plan)

corrective-repair (plan, f)
 top-level-goals-of-plan ← top-level-goals-of-plan ∪ top-level-goals-of-plan labeled
    not to depend on the desired outcomes of the action in f.
 return (plan)
```

Figure 2: Conditional, probabilistic planning algorithm.

duce an initial plan, and then perform both preventive and corrective repairs to improve it.[4] Weaver was built on top of a bidirectional planner (Prodigy 4.0), and therefore uses a different set of basic plan generation operations than those described in this paper. The Weaver project focuses on efficiently reasoning about which actions to choose in order to most quickly improve the likelihood of success (Blythe 1995); both preventive and corrective repair are then considered for those actions. Unlike most of the other planners, it also includes explicit mechanisms for dealing with external events. Mahinur was built on top of a backward chaining planner (Buridan), and introduces utility functions to reason not just about a plan's probability of success, but also about its expected value. It also uses a different set of techniques than Weaver for selecting the branching actions to focus on, reasoning directly on the plan graph instead of using a separate Bayes net mechanism. It focuses on explicit mechanisms for selecting failures. This feature is complementary to the Weaver framework—Mahinur deals with which failure points to consider first, and given a failure point, Weaver's action selection mechanism can determine the best way to correct it.

In the next section, we will first describe a general algorithm that uses the basic conditional, probabilistic planning operations, and then describe a modified version that provides plans for assessment.

Algorithm

Based on the discussion above, we can now provide a clear algorithm for conditional, probabilistic planning (Fig. 2). The algorithm rests on the observation that this type of planning involves repairing plan flaws (closing an open precondition and resolving a threat) and repairing dangling edges (corrective repair and preventive repair). The input is a set of initial conditions, a set of goal conditions, and a time limit T. The output is a set of plans. The algorithm is a plan-space search, where, as usual, the nodes in the search space represent partial plans. We assume that actions are encoded using the probabilistic action representation described above. To achieve an open condition c, the planner will find an action that includes a branch $< t_i, p_{i,j}, e_{i,j} >$, such that one of the elements of $e_{i,j}$ unifies with c. The relevant trigger t_i will then become a new open condition. Note that a condition c remains "open" only so long as it has no incoming causal link; once an action α has been inserted to (probabilistically)

[4] In Mahinur, the mechanism for preventive repairs has not yet been implemented.

```
PLAN (init, goal, T)
  plans ← { make-init-plan ( init,goal) }
  qp-plans ← ∅

  while plan-time < T and plans is not empty do
    CHOOSE (and remove) a plan P from plans
    if P is quasi-complete then
      qp-plans ← qp-plans ∪ P
      SELECT a dangling edge f from P
      plans ← corrective-repair(P, f) ∪ preventive-repair (P, f)
    else
      SELECT a flaw f from P.
      add all refinements of P to plans:
         plans ← new-step(P, f) ∪ step-reuse(P, f) if f is an open condition,
         plans ← demote(P, f) ∪ promote(P, f) ∪ separate (P, f) ∪ confront(P, f)
                                                  if f is a threat.
return (qp-plans)
```

Figure 3: Modified conditional, probabilistic planning algorithm.

produce c, it is no longer open, even if α has only a small chance of actually achieving c.

We assume that preventive repair is achieved by reintroducing the triggers for desired effects into the set of open conditions, as done in Buridan; we assume corrective repair is achieved by adding new, labeled copies of the goal state as in CNLP. Consistent with the prior literature, we use SELECT in our algorithm to denote a non-deterministic choice that is *not* a backtrack point, and CHOOSE for a backtrack point. As usual, node selection, but not normal flaw selection, is subject to backtracking.

We will call a plan *quasi-complete* if it has no open conditions or unresolved threats, and modify the above algorithm to incorporate quasi-complete plans as shown in Fig. 3. This algorithm involves forming a quasi-complete plan, deciding which dangling edges in it to handle, and then repairing those edges by, intuitively, figuring out what to do if the chosen actions have undesired outcomes, and/or figuring out how to make the chosen action's desired outcome more likely.

There are two types of partial plans in the search space: normal plans and quasi-complete plans. Normal plans are refined in the usual way, by selecting either an open condition and establishing it, or by selecting a threat and resolving it. Quasi-complete plans are treated differently. First, the planning algorithm treats all quasi-complete plans as potential solutions to the planning problem. Therefore, whenever a quasi-complete plan is found, it is stored into the set of plans to be returned at the end of processing. Second, a quasi-complete plan may be subject to further refinement, to improve its probability of success. The procedures for corrective and preventive repairs are therefore invoked. The successor nodes generated by these procedures are normal nodes, and thus will be subject to normal refinement for at least one iteration of the main loop.

The output of this algorithm is a set of quasi-complete plans, which may be subsequently evaluated, as we describe in the next section.

Outcome Completion

Existing planners all attempt to find a plan that achieves a given goal with a probability exceeding some threshold. They implicitly assume that the cases in which the plan fails are all equivalent, and they explicitly assume that the cases in which it succeeds are all equivalent. Of course, neither of these assumptions is true in general. Some plan failures are worse than others. Moreover, even if we maintain the assumption that the achievement of a specific goal has a fixed value, different plans may have different "side-effects" that influence their value. A plan P that has a higher probability of achieving a goal G may nonetheless be less desirable than some other plan Q with lower probability of achieving G, because either the situation that will result should P fail will be bad, or because the side-effects associated with P's success are bad, or both.

This observation is the basis of the well-established field of decision theory. The classic example that illustrates the problem is Savage's omelet-making example (Savage 1972, pp. 13-14):

> "Your [spouse] has just broken five good eggs into a bowl when you come in and volunteer to finish making the omelet. A sixth

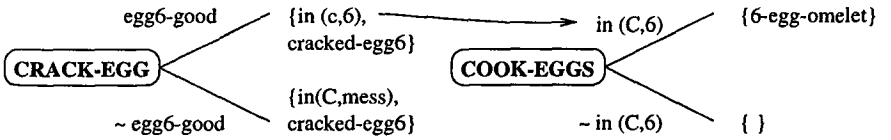

Figure 4: A quasi-complete plan for making an omelet.

	State	
Alternative	Egg Good	Egg Rotten
break into bowl	6-egg omelet	no omelet, 5 good eggs destroyed
break into saucer	6-egg omelet, saucer to wash	5-egg omelet, saucer to wash
throw away	5-egg omelet, 1 good egg destroyed	5-egg omelet

Table 1: Savage's Omelet Decision Problem (adapted from (Savage 1972)).

> egg, which for some reason must either be used for the omelet or wasted altogether, lies unbroken beside the bowl. You must decide what to do with this unbroken egg. Perhaps it is not too great an oversimplification to say that you must decide among three acts only, namely, to break it into the bowl containing the other five, to break it into a saucer for inspection, or to throw it away without inspection. Depending on the state of the egg, each of these three acts will have some consequence to you ..."

The consequences, of course, depend on whether the egg happens to be good or rotten, as illustrated in Table 1. While decision theory provides a way of comparing these alternatives, what it does not provide is an account of where the alternatives come from in the first place. The planning algorithms we have been considering, however, can do just this. In particular, the quasi-complete plans that are returned by the second algorithm in the previous section can constitute the alternatives to which decision-theoretic reasoning should apply. However, for this approach to work, additional reasoning must be performed to compute the "completions" of the quasi-complete plans.

We illustrate the idea of completion computation with Savage's omelet example. Fig. 4 shows a quasi-complete plan that would be found by our algorithm, given the CRACK-EGG and COOK-EGGS operators depicted. (For clarity, we have omitted the initial state, and some preconditions/effects that are not immediately relevant.) This corresponds to the first alternative in Table 1: it is the obvious plan in which the agent cracks the egg directly into the bowl already containing five other eggs. However, this plan contains a dangling edge: the one that will occur after CRACK-EGG is performed, if the egg is bad. To assess this plan, we need to determine what the state of the world will be, should this outcome occur. This can be computed by means of action progression, as it is commonly understood in the planning literature. A linearization of the plan must be selected, and then the results of each action in sequence must be computed.

The particular plan shown in Fig. 4 cannot be refined by either preventive or corrective repair: there is no way to increase the probability that the egg will be good, and, if the egg is bad, once it has been cracked into the bowl with the other eggs, there is no way to recover from the mess created. However, another quasi-complete plan that would be formed by our algorithm corresponds to the second alternative in Savage's table: here the agent first cracks the egg into a saucer, then observes it, and, if it is good, dumps it into the bowl with the other five eggs.

Again, in order to evaluate this plan, it is necessary to know what the outcome will be should the branching action—this time, an observe action—have its undesired outcome. But it is also important here to note that even when the goal *is* achieved—i.e., when there are six eggs cracked into the bowl—the plan will have a negative side-effect, namely, that there will be an extra dirty saucer. What this suggests is that action progression needs to be performed not only for the states that follow dangling-edges, but also for the goal states themselves. This is a departure from standard planning algorithms, which assume that so long as the goal propositions are achieved, it does not matter what else is achieved as well.

The last of the alternatives in Savage's table is quite interesting, since it describes a plan to achieve a different goal than the original one: namely, the goal in which the omelet has only five eggs. We suggest that the new goal may be generated from an analysis of the previous plan, along the following lines. Assume that the planner begins with a goal of G. In the process of trying to form plans for G, it finds one with two branches. In the first branch,

outcome completion results in a final state including $< G, d >$, where d is some detractor, i.e. a side-effect with negative impact. In the second branch, outcome completion yields $< G', d >$, where d is the same detractor, and G' is a goal that is reasonably close to G (in this case, having 5 eggs instead of 6 in the bowl). This configuration of outcomes may suggest to the planner that it try and form a plan for the transformed goal G', seeking a solution that does not have the negative side-effect d. We suspect that other such goal-transformation principles can be derived from an analysis of outcome completion, but leave this to future research.

There have only been a few prior efforts at decision-theoretic planning. The best known and most well-developed of these is the DRIPS system (Haddawy & Suwandi 1994). DRIPS takes a very different approach to the problem. It is a hierarchical task-network (HTN) planner, which computes the expected utility of each plan it expands, and prunes those whose possible utility is dominated by other options. The mechanisms for handling uncertainty in an HTN framework are quite different from those in the causal-link framework: much of the reasoning that is encoded as preventive and corrective repair is done by the designer of the HTN task networks, rather than by the system during plan expansion.

DRIPS interleaves plan expansion and decision-theoretic assessment, which is another significant difference from the approach we are describing. Indeed, it is reasonable to ask why we have chosen not to do this interleaving as well. To some extent, our algorithm *does* rely on assessments that will occur during the planning process, in particular, in the implementation of the CHOOSE and SELECT decision. But in many circumstances, it may be reasonable to generate several alternatives—our quasi-complete plans—and store them for future use. Once the alternatives have been computed and stored, they can be recalled in subsequent situations, and decision-theoretic assessment can be directly applied, using the particular probability and utility functions that pertain to the current setting. Once you have figured out that you can first crack an egg into a saucer and only then dump it in with the other eggs, you can thereafter consider this alternative without "rediscovering" it. Sometimes you may decide to use it (say, because it's very important to you to that the other eggs not be ruined, since you're preparing a special dish for guests), while other times you may decide not to use it (say, because you haven't seen a rotten egg in three years, and expect that the probability that this particular egg is quite low). The generation and computation of the outcomes of quasi-complete plans can thus be seen as a means of populating a case-base for later re-use.

Conclusion

In real-world environments, planners must deal with the fact that actions do not always have certain outcomes, and that the state of the world will not always be completely known. Good plans can nonetheless be formed if the agent has knowledge of the probabilities of action outcomes and/or can observe the world. Intuitively, if you don't know what the world will be like at some point in your plan, there are two things you can do: (i) you can take steps to increase the likelihood that it will be a certain way, and (ii) you can plan to observe the world, and then take corrective action if things are not the way you would like them to be. These basic ideas have been included, in different ways, in the prior literature on conditional and probabilistic planning. The focus of this paper has been to synthesize this prior work in a unifying algorithm that cleanly separates the control process from the plan refinement process.

Finally, we discussed the use of our planning algorithm as a source of alternatives for decision-theoretic assessment. We described the importance for this purpose of computing the failed states and side-effects in goal states, and we suggested that the analysis of the computed states can suggest new goals for consideration. We consider this last question, of how planning itself can lead to transformed goals, to be particularly interesting and important, and we plan to pursue it in our future research.

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 F49620-98-1-0436), and by NSF Grants (IRI-9619579, IRI-9258392, and IRI-9619562).

References

Blythe, J., and Veloso, M. 1997. Analogical replay for efficient conditional planning. In *Proc. 14th Nat. Conf. on Artificial Intelligence.*

Blythe, J. 1995. The footprint principle for heuristics for probabilistic planners. In *European Workshop on Planning.*

Draper, D.; Hanks, S.; and Weld, D. 1994. Probabilistic planning with information gathering and contingent execution. In *Proc. 2nd Int. Conf. on AI Planning Systems*, 31–36.

Drummond, M.; Bresina, J.; and Swanson, K. 1994. Just-in-case scheduling. In *Proc. 12th Nat. Conf. on Artificial Intelligence.*

Goldman, R. P., and Boddy, M. S. 1994a. Conditional linear planning. In *Proc. 2nd Int. Conf. on AI Planning Systems*, 80–85.

Goldman, R. P., and Boddy, M. S. 1994b. Epsilon-safe planning. In *Proc. 10th Conf. on Uncertainty in Artificial Intelligence*, 253–261.

Goldman, R. P., and Boddy, M. S. 1996. Expressive planning and explicit knowledge. In *Proc. 3rd Int. Conf. on AI Planning Systems.*

Haddawy, P., and Suwandi, M. 1994. Decision-theoretic refinement planning using inheritance abstraction. In *Proc. 2nd Int. Conf. on AI Planning Systems*, 266–271.

Kushmerick, N.; Hanks, S.; and Weld, D. S. 1995. An algorithm for probabilistic planning. *Artificial Intelligence* 76:239–286.

Onder, N., and Pollack, M. E. 1997. Contingency selection in plan generation. In *European Conf. on Planning.*

Peot, M. A., and Smith, D. E. 1992. Conditional nonlinear planning. In *Proc. 1st Int. Conf. on AI Planning Systems*, 189–197.

Pryor, L., and Collins, G. 1996. Planning for contingencies: A decision based approach. *Journal of Artificial Intelligence Research* 4:287–339.

Savage, L. J. 1972. *The Foundations of Statistics, 2nd ed.* New York: Dover Press.

Warren, D. 1976. Generating conditional plans and programs. In *Proceedings of the Summer Conf. on AI and Simulation of Behaviour.*

Weld, D. S. 1994. An introduction to least commitment planning. *AI Magazine* 15(4).