Pre-print from AI Magazine, 20(2), Summer 1999, pp 37–54.

An Overview of Planning Under Uncertainty

Jim Blythe
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292 USA
blythe@isi.edu

Abstract

The recent advances in computer speed and algorithms for probabilistic inference have led to a resurgence of work on planning under uncertainty. The aim is to design AI planners for environments where there may be incomplete or faulty information, where actions may not always have the same results and where there may be tradeoffs between the different possible outcomes of a plan. Addressing uncertainty in AI planning algorithms will greatly increase the range of potential applications but there is plenty of work to be done before we see practical decision-theoretic planning systems. This article outlines some of the challenges that need to be overcome and surveys some of the recent work in the area.

Introduction

AI planning algorithms are concerned with finding a course of action to be carried out by some agent to achieve its goals. In problems where actions can lead to a number of different possible outcomes, or where the benefits of executing a plan must be weighed against the costs, the framework of decision theory can be used to compare alternative plans.

For a long time the planning community has maintained an interest in decision-theoretic planning, with Feldman and Sproull's work being one of the earliest in this area (Feldman & Sproull 1977). After a period where attention was focussed on efficient algorithms for planning under quite restrictive assumptions, the past five years have seen a resurgence in work on planning under uncertainty and decision-theoretic planning. There are several reasons for this. The recent advances in Bayesian inference have made it feasible to compute (or approximate) the expected utility of a plan under conditions of uncertainty. The success of Markovian approaches in areas such as speech recognition and the closely-related reinforcement learning techniques have encouraged work in planning using Markov decision processes. Faster computers have made it feasible to build decision-theoretic planners, which in general have more complicated search spaces than their classical counterparts.

In this article I provide a brief overview of some of the recent work in decision-theoretic planning. I highlight some of the advances made and some technical problems that still lie ahead. In the next section I describe the problem addressed by decision-theoretic planning systems. The following two sections describe work based on classical planning algorithms and based on solving Markov decision processes respectively.

Goals of decision-theoretic planning

A planning problem in AI is usually specified as follows: Given a description of the current state of some system, a set of actions that can be performed on the system and a description of a goal set of states for the system, find a sequence of actions that can be performed to transform the system into one of the goal states. In the text-book example of the blocks world, the system consists of a set of named blocks, each of which may be on the ground or on top of another block. The possible actions move a block from one location to another and the goal is a particular configuration of blocks, for example a tower.

When planning programs are used to provide courses of action in real-world settings, such as medical treatments, high-level robot control or disaster relief, they must take into account the fact that actions may have several different outcomes, some of which may be more desirable than others. They must balance the potential of some plan achieving a goal state against the risk of producing an undesirable state and against the cost of performing the plan.

Decision theory (Luce & Raiffa 1957) provides an attractive framework for weighing the strengths and weaknesses of a particular course of action, with roots in probability theory and utility theory. Given a probability distribution over the possible outcomes of an action in any state, and a reasonable preference function over outcomes, we can define a utility function on outcomes such that whenever the agent would prefer one plan over another, the preferred plan has higher expected utility. The task of the planner then seems straightforward — to find the plan with the maximum expected utility (MEU).

Unfortunately, decision theory says nothing about the task of constructing a plan with high expected utility. Therefore AI planning and decision theory would appear to be complementary and there has been interest in merging the two approaches for a considerable time (Feldman & Sproull 1977). However there are hard problems to overcome. It is usually not feasible to search the entire space of plans to find the MEU plan. Indeed, computing the expected utility of a single plan can be prohibitively expensive because the number of possible outcomes can grow very large. In order to avoid specifying the value of each possible outcome we need to find compact specifications of utilities as well as actions, and consider the interaction between them. The recent increase in research in decision-theoretic planning follows the recent success of work in reasoning under uncertainty, and draws on compact representations such as those of belief nets (D'Ambrosio 1999).

Extending classical planners

Since the early systems GPS (Newell & Simon 1963) and STRIPS (Fikes & Nilsson 1971), most AI planners that have been designed make essentially the same assumptions about the world in which they operate. From the point of view of decision-theoretic planing, three assumptions of note are:

- the goal of the planner is a logical description of a world state,
- the actions taken by the planner are the only sources of change in the world, and
- each action can be described by the conditions under which it can be applied and its effects on the world. These effects are described by a set of facts which are either to be added to or deleted from the world state in which the action is applied.

The last is known as the "STRIPS assumption". I refer to planners that share these assumptions as the "classical planners". They include Prodigy (Veloso *et al.* 1995), UCPOP (Penberthy & Weld 1992), Graphplan (Blum & Furst 1997) and the hierarchical task network planner Nonlin (Tate 1977)[1].

[1]Although compound operators in hierarchical task network planners are more complex, the primitive operators (that exclusively fill complete plans) have this form.

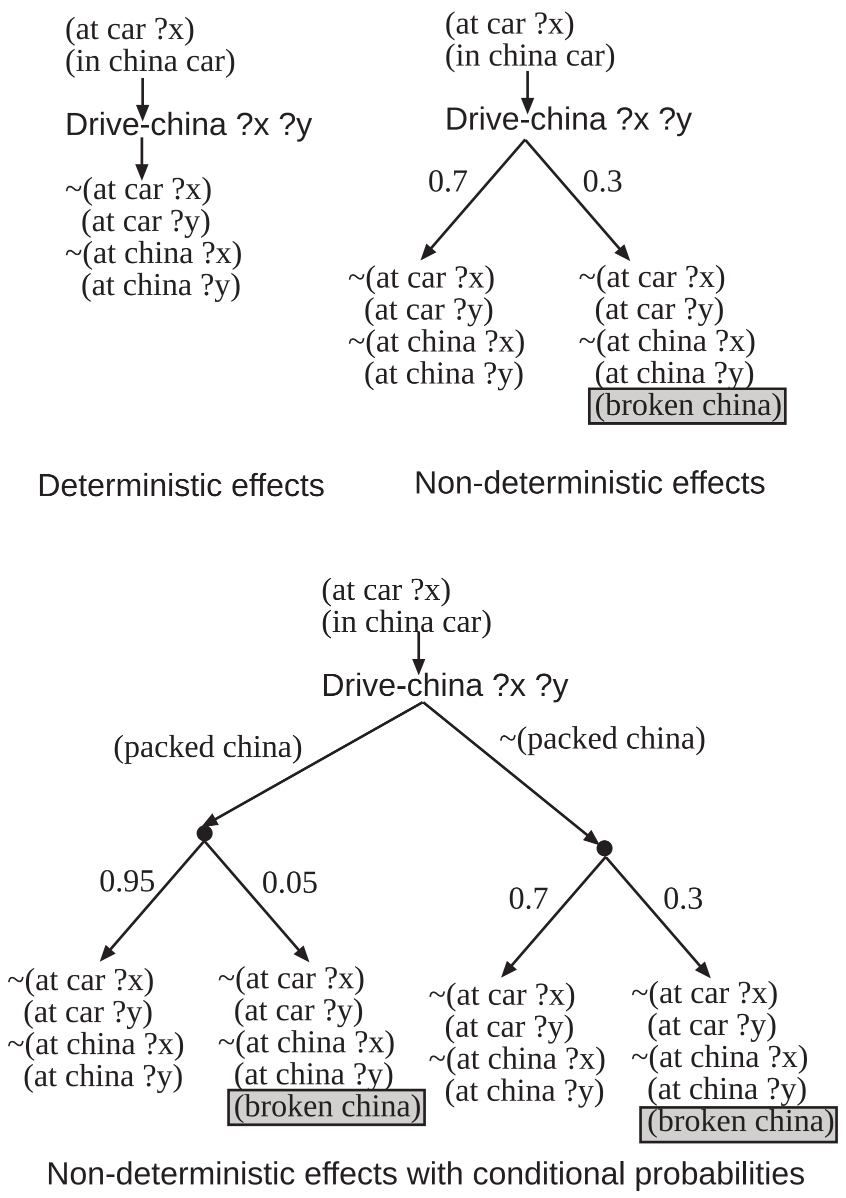

Figure 1: Three versions of the operator `drive-china`

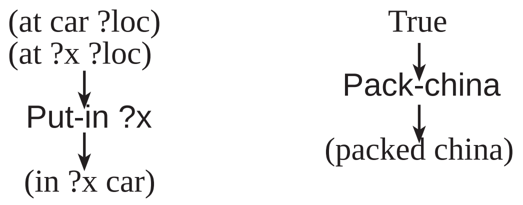

Figure 2: Other operators in the moving-house example.

Here I describe work that relaxes one or more of these assumptions in a classical planner. First, we can relax the assumption that actions are deterministic by specifying a set of possible outcomes, rather than the single outcome in the deterministic case. Consider the planning problem of moving my kitchen china from an old apartment to a new one. The goal is to move the china unbroken, specified as `(at china new-apartment)` $\wedge$ $\neg$ `(broken china)`. Figure 1 shows three increasingly detailed descriptions of the `drive-china` action which can achieve this goal. The first is deterministic and could be used in a plan that achieves the goal with certainty: `(put-in china)`, `(drive-china old-apartment new-apartment)`, using the extra operators in Figure 2. The second `drive-china` operator has two alternative sets of effects, modeling two different world states that can arise if the operator is applied. The numbers on the arcs are the probabilities of each state of affairs. Under this model, the two-step plan will only achieve the goal with probability 0.7.

In the third formulation of `drive-china`, different sets of outcomes are possible based on the conditions that hold when the operator is applied and the arcs now show conditional probabilities. The two-step plan still has a probability of 0.7 of succeeding. The three-step plan `(pack china) (put-in china) (drive-china ...)` succeeds with probability 0.95.

Second, we can relax the assumption that goals are logical descriptions of world states, so that a plan either succeeds or fails. We can attach utilities to different world states representing degrees of desirability. For example suppose the state in which all my china is moved unbroken has a utility of 10, while one in which half of my china is moved and none is broken has a utility of 6, any state in which china is broken has a utility of -100 and any other state has a utility of 0. If the pack operator can only pack half of the china, the two-step plan has an expected utility of -23 while the three-step plan has an expected utility of 1. In this case the plan to pack some of the china is slightly better than doing nothing, which has a utility of 0, while the plan to simply move the china without packing it is much worse.

Challenges for classically-based decision-theoretic planning

In the next four sections I describe four different approaches for planning under uncertainty that can be viewed as extending a classical planning approach. These are SNLP-based planners, DRIPS, Weaver and Maxplan, although the last of these is arguably not a classical planner. Organizing the article around specific research programs makes it easier to keep in mind the specific assumptions behind each one, at the cost of making it harder to follow how they address some of the issues and challenges that must be faced in creating decision-theoretic planners. In this section I briefly describe some of those issues, as a way of forming a road-map for understanding the different approaches.

Action representation

As the china-moving example shows, in order to represent uncertain actions one has to represent several alternative outcomes and their probabilities, conditional on the state. Providing a richer representation of action inevitably makes the planning problem harder, and planners must use a representation that expresses the different outcomes as concisely as possible while still being fully expressive. The representation of the final **drive-china** operator is not very concise, for example, because the four possible outcomes are described explicitly, even though the two branches only differ in their probabilities and each outcome only differs in whether the china is broken. When different state variables are affected independently by an action it is possible for the number of outcomes to grow exponentially in the number of variables.

Plan utility

Several systems based on classical planning ignore outcome utility beyond the binary measure of goal achievement, although exceptions are DRIPS (Haddawy & Suwandi 1994) and Pyrrhus (Williamson & Hanks 1994). In planners based on Markov decision processes, discussed later in this article, a numeric reward function is specified for an action in a state. While this is adequate from a purely decision-theoretic point of view, a more structured representation of utility is needed to make planning tractable, one where the utility of a state is built up from smaller building blocks so that it is less tedious to specify and also the utility of abstract states can be estimated. This problem has been approached in both frameworks.

Some planners relax the binary measure of goal achievement to allow partial satisfaction of goals. In the china-moving example the full goal is to move all the china unbroken, but we might attach partial credit for moving some of it unbroken. Given a conjunctive set of goals, sometimes the utility from solving a subset can be specified as the sum of some utility for each goal. Care must be taken with this approach, for example in the china-moving example the top-level goals are `(at china new-apartment)` and `(not (broken china))`, but a planner that achieves the first goal without the second should not receive half the credit. Sometimes the utility of an outcome can be made more modular by summing the cost of each operator independently (Pérez & Carbonell 1994), with the same caution.

Haddawy and Hanks describe an approach that further assumes a deadline by which a goal should ideally be achieved, with a separate penalty function that offsets the utility for the goal when it is achieved after the deadline. They propose a representation that encompasses most of these approaches independently of any planner in (Haddawy & Hanks 1998a).

Plan evaluation

Whether outcome utility is multi-valued or simply binary, one must compute or estimate the expected utility of a plan. For the degenerate case, this is simply the probability of success. There are several different ways this can be approached as we shall see, and none completely dominates any other. For the problem of deciding which partial plan to expand, all that is required is a comparative value — deciding which plan has higher expected utility — which is sometimes simpler than finding the absolute value. For efficient planning, one must also consider the tradeoff between time spent estimating plan value and time spent expanding a plan whose value is poorer than estimated.

Observability and conditional planning

Most of these systems distinguish between observable and non-observable domain variables. A classical STRIPS plan essentially assumes no observability since the plan is a sequence of actions executed "blind". In normal Markov decision processes, the output is a policy, a mapping from state to action which assumes that every state variable can be observed without error, although some work that relaxes this assumption is described later. In the Buridan planner (Kushmerick, Hanks, & Weld 1995) actions can have sensing results as well as effects in the world, and noisy sensors can be modeled.

In an uncertain world, high-utility plans created in advance may need to have some actions that are conditional on future observations. Most of the planners discussed are capable of handling this, employing a variety of techniques. Note that planning agents must be able to make observations of their domain during execution in order to use conditional plans.

Tractability

Perhaps the largest question mark hanging over the planners described here is their tractability. Few have been applied in large, high-dimensional domains although exceptions are DRIPS (Haddawy, Doan, & Kahn

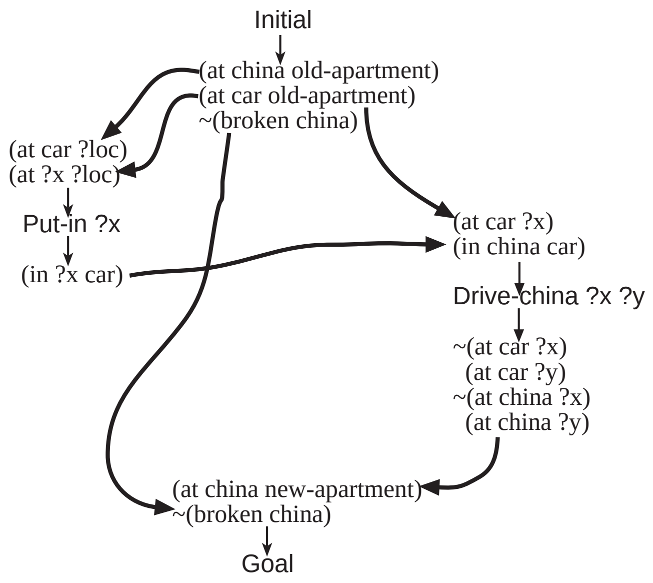

Figure 3: A plan to move china, as found by SNLP. The bold arrows represent causal links in the plan.

1996) and Weaver (Blythe 1998). In addition it is rare that the planners are directly compared with each other although MAXPLAN is an exception (Majercik & Littman 1998). This is partly because as we shall see the planners have concentrated on different aspects of the planning problem.

The tractability of the planners is affected by the kind of planning algorithm used: SNLP, Prodigy, refinement planning, HTN planning and compilation to satisfiability have all been used. It is also affected by the use of search control, which has been studied in several of the planners.

SNLP based planners

The classical planner SNLP maintains a plan as a partially-ordered set of actions, along with ordering constraints between steps, bindings for variables in steps and explicit data structures, called "causal links", that represent what the step is supposed to accomplish in the plan (Weld 1994). A causal link records the subgoal being achieved, the "consumer" step, that has this subgoal as a precondition, and the "provider" step, that has the subgoal as an effect. For example, the arrow between `drive-china` and `goal` in Figure 3 is a causal link for the subgoal `(at china new-apartment` with `drive-china` as provider and `goal` as consumer. `goal` is a dummy step added to the plan to represent the top-level goals. Similarly `initial` is a dummy step representing the initial state. The SNLP algorithm follows the loop shown in Figure 4. It repeatedly picks an unsupported condition in the plan, adds a causal link supporting it from some step that achieves the condition, and removes any threats to the link by forcing any step that negates the condition to take place either before or after both linked steps. By considering all possible choices for causal links and threat orderings, SNLP is complete, ie it will find a plan if one exists. For more details see (Weld 1994).

Buridan

Buridan (Kushmerick, Hanks, & Weld 1995) is a modified version of the SNLP planning algorithm that can create plans that meet a threshold probability of success when actions are non-deterministic, as are the last two actions in Figure 1. Buridan differs from SNLP by allowing more than one causal link for each condition in the plan. Under different execution scenarios, different actions may cause the condition to be true, so the

1. If the current plan has no unsupported conditions, return it.
2. Pick an unsupported condition *P* for some step *U* and add a causal link to the plan that supports it, achieving the condition with either an existing or a new step, *A*. (On backtracking, try all possible links.)
3. Resolve any *threats* to the new link. A threat comes from any step *T* in the plan that can take place between *A* and *U* and can negate *P*. Resolve the threat by ordering *T* either before *A* or after *U* if possible (on backtracking, try both orderings). If this is not possible, fail and backtrack.

Figure 4: An outline of the SNLP algorithm

links combine to increase support for the condition. SNLP's termination criterion that all conditions have exactly one link is no longer appropriate. Instead, Buridan explicitly computes the probability of success of the plan and terminates if this is above the given threshold. Figure 5 shows a plan found by Buridan in the china-moving example if the third model of action is used.

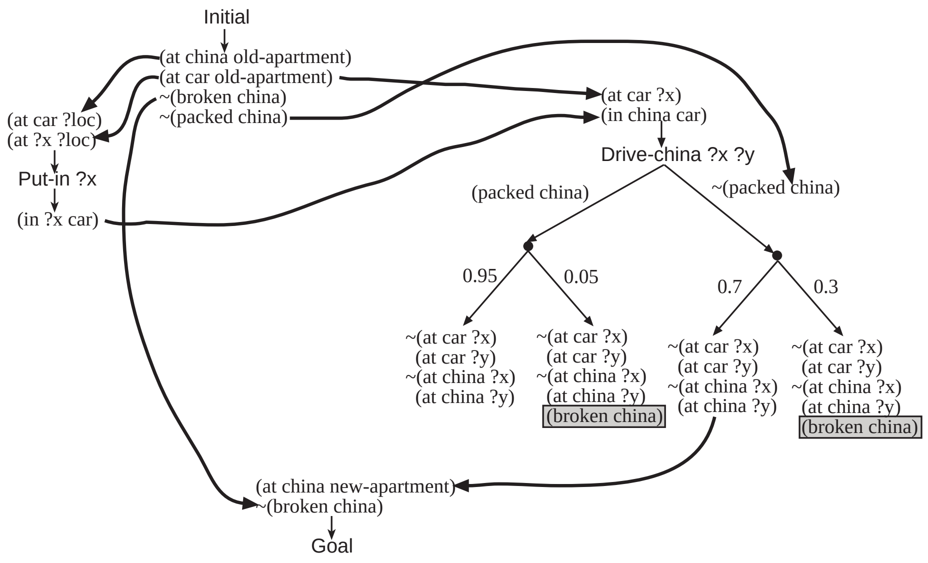

Figure 5: A plan to move china, found by Buridan

This plan succeeds with probability 0.7, because with probability 0.3 executing `drive-china` can lead to the china being broken. This step is therefore a threat to the link for `(broken china)` from `Initial` to *Goal*. In addition to SNLP's re-orderings to remove threats, Buridan can *confront* a threat by decreasing the probability of an outcome in which the threatened condition is negated. In this case, that is done by adding a link from the outcome with probability 0.95 that does not add `(broken china)`, and planning for its trigger, `(packed china)`. Buridan can then find the plan [`pack-china`, `put-in china`, `drive-china`], that succeeds with probability 0.95.

So far we haven't discussed how a planner finds the probability that a plan succeeds. There are a number of possible strategies, based on forward projection or on analyzing the dependencies among state descriptors. Empirical experiments show no clear winner (Kushmerick, Hanks, & Weld 1995).

Forward projection begins with the set of possible initial states and simulates executing the plan one

Step	State	Features	Parent	Prob
Pack-china	S^1_{pack}	packed	I	0.5
Pack-china	S^2_{pack}	¬ packed	I	0.5
Put-in-china	S^1_{put}	packed, in-car	S^1_{pack}	0.5
Put-in-china	S^2_{put}	¬ packed, in-car	S^2_{pack}	0.5
Drive-china	S^{11}_d	packed, ¬ broken	S^1_{put}	0.475
Drive-china	S^{12}_d	packed, broken	S^1_{put}	0.025
Drive-china	S^{21}_d	¬ packed, ¬ broken	S^2_{put}	0.35
Drive-china	S^{22}_d	¬ packed, broken	S^2_{put}	0.15

Figure 6: Using forward projection to evaluate the plan for the china-moving problem.

step at a time, maintaining the set of possible states and their probabilities after each step is completed. When the simulation is finished, summing the probability of each state in which the goals are satisfied gives the plan's probability of succeeding. The table at the top of Figure 6 shows the sets of states that are built as the three-step plan is evaluated by forward projection. To make the example more interesting we assume that the **pack-china** action succeeds with probability 0.5, and otherwise has no effect. It therefore leads to two states in the forward projection, which are propagated through the **put-in-china** step and lead to four possible states after the **drive-china** action. The probability of success, 0.825, is found by summing the probabilities of states S^{11}_d and S^{21}_d.

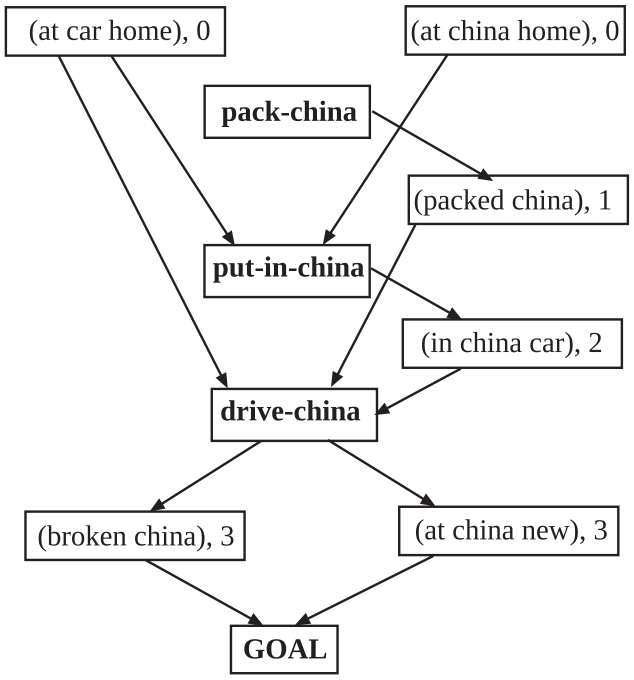

Figure 7: Using a belief net to evaluate the plan for the china-moving problem. The nodes with bold text represent actions in the plan, while the others represent fluent values at a given point in time.

In general, the number of states considered can clearly grow exponentially with the length of the plan. Many of the states created might differ only on features that are irrelevant to the plan. This observation

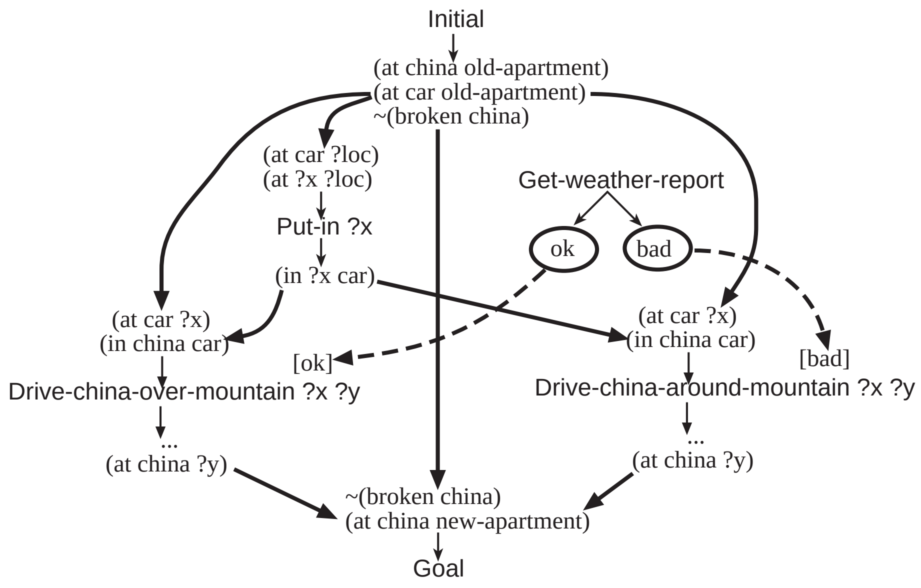

Figure 8: A conditional plan represented using context labels attached to actions.

suggests other plan evaluation strategies that exploit the structure of the plan. One such strategy is to evaluate the plan using a belief net (Pearl 1988). The STRIPS representation for action, in which the preconditions contain sufficient information to completely determine the outcomes of the action, is a special case of the Markov assumption and this is preserved in the action representation used here (Wellman 1990). One can use this observation to create a belief net that can be queried to determine the probability of success of the plan. There are many ways to do this — one example is shown in Figure 7. Its nodes are random variables representing either the value of state descriptors at a point in time or the outcomes of actions taken in the plan. The final action **goal** is included, with two possible values: **true** or **false**. The probability that this has value **true** is equal to the plan's probability of success.

Conditional planning

CNLP was the first SNLP-based planner to represent conditional plans (Peot & Smith 1992). Each step in the plan has a set of *context labels* associated with it that denote the conditions under which the step will be executed. For example, Figure 8 shows a conditional plan for moving the china in which the agent loads the china in the car and then listens to the weather report before deciding whether to take the route over the mountain or around the mountain. The operator `get-weather-report` has two possible outcomes, each producing one of the observation labels `ok` or `bad`. Each of the two driving operators is marked with one of the labels as its context. Thus they must be executed after `get-weather-report`, and `drive-china-over-mountain`, for example, will only be executed if the label `ok` was produced. If any subsequent actions had no context labels, they would be executed on both "branches" of the conditional plan. Contexts are not as direct a representation as putting an explicit branch in a plan, but they can easily represent plans that branch and then re-merge as well as partially-ordered branching plans.

C-Buridan refines the approach of CNLP by keeping observations distinct from effects and allowing different outcomes of a sensing action to have the same observation label, thus modeling partial observability (Draper, Hanks, & Weld 1994). In C-Buridan, new contexts are added in response to threats between actions that can be put in separate branches of a conditional plan. Obviously, the ability to create conditional plans can make the planning problem even less tractable. Most conditional planners do not require a plan to be produced for every contingency (although Cassandra is an exception (Pryor & Collins 1996)). Onder and Pollack have studied how to choose which contingencies to plan for, both to improve efficiency and based on the observation that some contingencies can safely be left for re-planning at execution time (Onder & Pollack 1997).

DRIPS: searching for optimal plans with a refinement planner

The planners described above search for plans that exceed a given minimum expected utility. In order to find the plan with maximum expected utility, however, one must somehow evaluate all possible plans to select the best one. In DRIPS, ranges of utility values are computed for partial plans, encompassing the best and worst expected utilities of all possible completions of the partial plan (Haddawy & Suwandi 1994). If the lowest value in some partial plan's range exceeds the highest value in another's the partial plan with the lower range can be dropped from consideration without expanding all the complete plans below it, an approach that can lead to significant savings. The dropped plan is said to be *dominated* by the other plan, an idea also explored by Wellman (Wellman 1990).

DRIPS accomplishes this by *skeletal refinement planning* based on an abstraction hierarchy of operators (Friedland & Iwasaki 1985). In this approach, a partial plan is a sequence of operators, one or more of which may be an abstraction of a number of ground operators. Planning proceeds by repeatedly choosing a more specific version of an operator in the plan until there are no abstract operators left. The planner begins with one of a set of skeletal plans containing highly abstract operators chosen from a library. Figure 9 shows some example ground and abstract operators from the china-moving example, extended with a richer model of the utilities of outcomes.

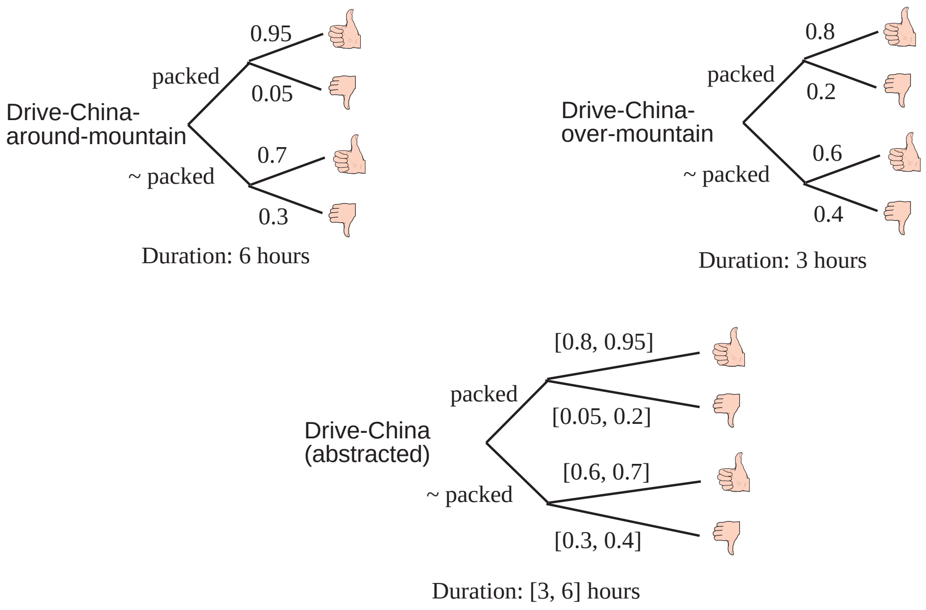

Figure 9: Ground and abstract operators from the china-moving example that could be used by the refinement planner DRIPS. The thumbs-up symbol denotes an outcome in which the china is moved successfully and the thumbs-down icon denotes an outcome in which the china is broken.

Figure 10 shows an abstraction/decomposition network that describes all possible specializations of a skeletal plan to move the china. Solid lines denote decomposition of an operator and dashed lines denote possible choices for an abstract operator. The skeletal plan encodes four potential plans: one can choose whether to pack the china and one can independently choose which route to take.

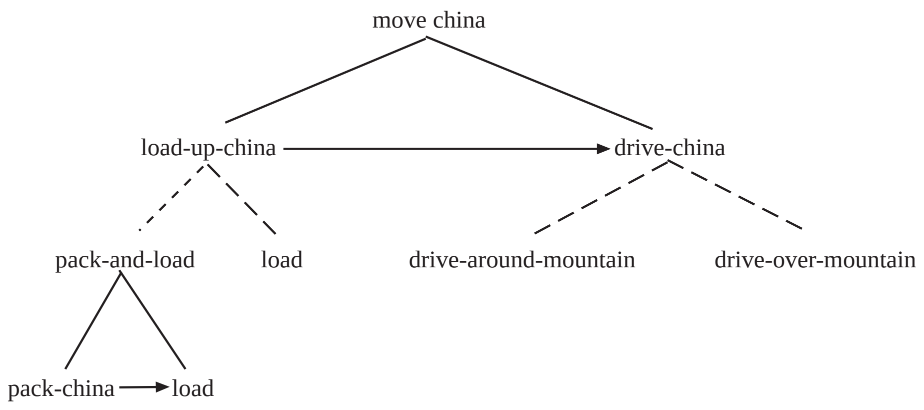

Figure 10: Ground and abstract operators from the china-moving example that could be used by the refinement planner DRIPS. Solid lines represent compound decompositions and dotted lines represent alternative decompositions.

Suppose DRIPS begins refining this plan by choosing between **pack-and-load** and **load** to refine the **load-up-china** operator. The planner will normally add both refinements to a queue of partial plans, but first it computes ranges of expected utility for each one. To do this, it explicitly considers each set of possible outcomes, or *chronicle*, for each plan, as shown in Table 1. Each plan has two possible chronicles, either the china gets broken or it does not. For each chronicle, DRIPS computes ranges for the duration, utility and probabilities based on the ranges or values found in the actions in the plan. A series of simple linear programming problems are solved to compute ranges for the expected utility of each abstract plan. In this case the ranges are [-17.6 0.7] for plan A and [-37 -23] for plan B. Therefore DRIPS will not add plan B to the list of plans to refine, since no refinement can do better than any refinement of plan A. DRIPS determined this without exploring the refinements of plan B, an ability that can lead to large computational savings in more complicated domains.

Plan A: pack and load china

Chronicle	Duration	Utility	prob
not broken	[3 6]	[3 6]	[0.8 0.95]
broken	[3 6]	-100	[0.05 0.2]

Plan B: load china, don't pack

Chronicle	Duration	Utility	prob
not broken	[3 6]	[5 10]	[0.6 0.7]
broken	[3 6]	-100	[0.3 0.4]

Table 1: A summary of the possible outcomes for each refinement of the skeletal plan found by refining the **load-china** abstract action.

The use of skeletal refinement planning in DRIPS trades planning power for simplicity and the ability to compute utility ranges for partial plans. In some complex domains it may be burdensome to represent the range of possible plans as skeletal plans that can be refined in this way. On the other hand, this form allows utility ranges to be computed for partial plans based on the ranges computed for abstract actions. It is not clear how to compute utility ranges for, say, partial Buridan plans, because they include no commitment for the action or actions used to achieve an open condition.

Some interesting work has been done on search control within the DRIPS framework. DRIPS has two choice-points: which abstract plan to specialize and which abstract action in a plan to specialize. The plan with current highest upper bound on expected utility must be expanded in order to provably find the optimal plan, so choosing such a plan to expand gives a search heuristic similar to A* (Haddawy, Doan, & Goodwin 1995; Goodwin & Simmons 1998). The representation of utilities in DRIPS allows for goal deadlines and goals of maintenance, following (Haddawy & Hanks 1998a). The system has been applied in challenging

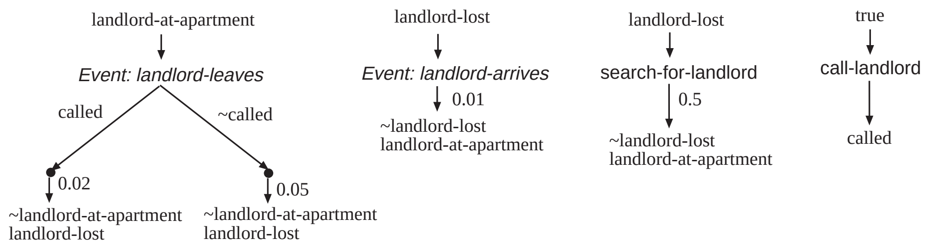

Figure 11: Two exogenous events modeling the landlord's whereabouts and operators to call and search for the landlord.

medical domains with favorable results (Haddawy, Doan, & Kahn 1996).

Weaver: efficiently handling exogenous events

Weaver (Blythe 1998) is a probabilistic planner based on Prodigy (Veloso *et al.* 1995). Like C-Buridan, it has no representation of utilities. It has no explicit model of observability, but assumes the world is completely observable at plan execution time. However it has an explicit representation for uncertain exogenous events, in other words for events that take place outside the volition of the planning agent and that can be modeled as occurring with some conditional probability given the world state. External events often play a significant role in real domains, and should be considered by a decision-theoretic planner since (1) they can be influenced at least indirectly by altering the world state conditions they depend on and (2) contingent plans can be built based on their effects.

For example in the china-moving domain, suppose that in order to unload the china at the new apartment, the landlord must be waiting there with the key when the car arrives. Initially the landlord is waiting, but as time passes he may get restless and leave, in which case he may return later. A planning agent can reduce the probability that the landlord will leave by calling him when the car is close to the apartment. If he is not waiting when the car arrives, the agent can search for him. Figure 11 shows a set of exogenous events modeling the landlord's whereabouts over time, as well as the operators **call-landlord** and **search-for-landlord**. Assuming a discrete-time model, the **landlord-moves** events mean that if at any time point the landlord is at the apartment and has not been called, he is at the apartment at the next time point with probability 0.95, and his whereabouts are unknown with probability 0.05. If he has been called the probability of his leaving drops to 0.02. If his whereabouts are unknown, he may reappear at the apartment with probability 0.01.

In theory the use of exogenous events does not increase the representational power of the planner, since the effects of these events can be modeled by effects of the actions in the domain. For example, the drive-china actions could include effects describing the landlord's whereabouts, as could the pack-china operator if it takes a significant amount of time. In practice, modeling exogenous events in this way can be extremely inefficient, because their effects are duplicated in every operator in the domain, and conversely each operator may have to model many events.

Weaver evaluates plans efficiently when there are exogenous events using a three-step process. First a plan is built and partially evaluated ignoring exogenous events. In the china-moving domain, the same three-step plan that is found by Buridan will be found: **pack-china**, **put-in china**, **drive-china**. Second, the goal structure for this plan is used to determine the *relevant* exogenous events, which are those whose effects can alter the domain features required by the plan or can influence other relevant events. Third, the plan is improved by either adding steps to reduce the probability of an undesirable effect or by adding a conditional branch. The plan improvement operations are similar to those of C-Buridan (Draper, Hanks, & Weld 1994), but using the Prodigy planner (Veloso *et al.* 1995). The main difference is that the exogenous events that affect the plan evaluation are also considered as well as the actions in the plan (Blythe 1994). In the china-moving example, a possible improvement is to add a conditional branch to search for the landlord if he is not at the apartment when the car arrives.

The belief net fragment on the left in Figure 12 shows an evaluation structure for the plan before improvement, after the relevant events governing the movement of the landlord have been added. It is possible to make this evaluation more efficient by taking advantage of the fact that the exogenous events model a Markov process. Rather than explicitly modeling the events' effects at each time step, independent Markov chains are produced that describe parts of the world state's evolution over time — in this case, the landlord's whereabouts. The Markov chains are used to compute links in the belief net on the right in Figure 12. Such a belief net can be much more efficiently evaluated, especially when some actions have long durations compared with those of the exogenous events. Weaver uses a sound algorithm to create these streamlined belief nets automatically (Blythe 1996).

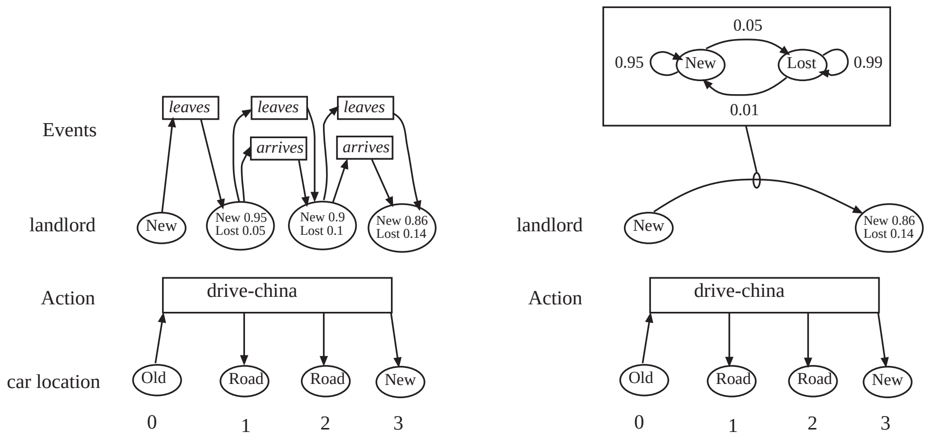

Figure 12: Two time-slice bayes nets that can be used to evaluate the plan under exogenous events. The numbers along the x-axis represent time points while nodes for fluents, actions and events are grouped along the y-axis. In the fragment on the left, exogenous events are represented directly while on the right, their effects are compiled into Markov chains that are used to compute probabilities over the time intervals of interest.

Search heuristics for planning under uncertainty have also been studied in Weaver. When steps are added to a plan to increase the number of chronicles in which it succeeds, a useful strategy is to seek steps that succeed in cases that are complementary to those in which the original plan succeeds. This can be done using a local analysis of the possible steps, producing search heuristics that improve performance both in Weaver and in Buridan (Blythe 1995). Weaver also makes use of analogy between different branches of a conditional plan (Blythe & Veloso 1997). These performance improvements allow it to be applied to problems requiring several rounds of improvement in plans some tens of steps in length (Blythe 1998). This is encouraging in the search for tractable planning under uncertainty, but leaves plenty of room for further improvement.

MAXPLAN: **probabilistic plan compilation**

One way to improve the efficiency of planning under uncertainty may come from inspiration from recently developed classical planning algorithms, such as SATPLAN (Kautz & Selman 1996) which compiles a planning problem into a logical satisfiability problem. After this step, standard algorithms can be used to solve the problem.

MAXPLAN (Majercik & Littman 1998) is a probabilistic planner that is similar in spirit to SATPLAN. Littman, Goldsmith and Mundhenk show that, under reasonable conditions, probabilistic planning is NP^{PP}-complete and introduce an NP^{PP}-complete decision problem E-MAJSAT (Littman, Goldsmith, & Mundhenk 1998). MAXPLAN compiles a probabilistic planning problem into an instance of E-MAJSAT and solves it. Just as Graphplan and SATPLAN are more efficient than classical planners in some domains, so MAXPLAN should have an advantage in some domains, as preliminary experiments indicate.

An instance of E-MAJSAT is a satisfiability problem, a Boolean formula whose variables are divided into two types, *choice* variables and *chance* variables. The chance variables are each assigned an independent probability of being true. The task is to find an assignment of truth values to the choice variables that maximizes the probability of the given Boolean formula being true. MAXPLAN transforms a planning problem into an instance of E-MAJSAT whose solution yields a plan with maximum probability of success, equal to the maximum probability of the E-MAJSAT problem. The number of clauses in the formula is polynomial in the size of the planning problem.

The resulting instance of E-MAJSAT is solved with the Davis-Putnam-Logemann-Loveland (DPLL) algorithm (Davis, Logemann, & Loveland 1962). This algorithm systematically finds the best assignment by constructing a binary tree in which each node represents a variable and each subtree represents a subproblem found by conditioning on the variable. DPLL either maximizes or computes the weighted average of the subtrees depending on whether the node is a choice or a chance variable. At each node, the algorithm eliminates all irrelevant variables (that only appear in satisfied clauses), forced variables (that appear alone in an unsatisfied clause) and "pure" variables (that appear either always negated or always not negated). The authors have experimented with different orderings on the variables.

Graphplan (Blum & Furst 1997) is Another classical planner that uses a plan compilation step, and is perhaps the best known. Probabilistic planning based on Graphplan has also been explored (Smith & Weld 1998; Weld, Anderson, & Smith 1998).

Pointers to other work

There is a growing literature on extending classical planning approaches to plan under uncertainty. I apologize to those whose work I omit here for reasons of space.

Goldman and Boddy (Goldman & Boddy 1994) explore the idea of relaxing the assumption that the outcomes of different actions in a plan are probabilistically independent, using a *knowledge-based model construction* approach (Wellman, Breese, & Goldman 1992). This assumption is made in all the planners discussed above, although the causal influences used by Goldman and Boddy bear similarities to the external events of Weaver.

XFRM (McDermott 1992) produces plans using a rich language that can express do-while loops and include arbitrary LISP code. The system selects a plan from a user-defined library and applies transformations to improve its performance. Because of the rich plan representation it is hard to evaluate plans analytically, and instead XFRM relies on possible execution scenarios called *projections* to compare plans and find potential problems.

Cypress (Wilkins *et al.* 1995) is a framework for probabilistic planning built around three components: SIPE-2 is a hierarchical task network (HTN) planner, PRS-CL is a reactive plan execution system and GISTER-CL is an uncertain reasoning system. Cypress is broader than the planners described here in that it allows planning and execution to be done concurrently. The HTN planning style, where domain operators describe a problem decomposition rather than specify a set of subgoals, is successful in applications of classical planning. It typically offers a reduced search space at the cost of requiring more information to be specified as part of the domain knowledge.

Summary

Almost every approach that has been used to solve classical planning problems has been adapted for decision-theoretic planning. This section described decision-theoretic planners based on SNLP, skeletal refinement planning, Prodigy and compilation to satisfiability. This has necessarily been only a small sample of the work being done in this area, but a representative one.

Each described system concentrates on a different aspect of planning under uncertainty and while each has been successful, their relatively narrow focus has made meaningful comparison difficult. Future work should lead to broader systems that would allow us to compare alternative strategies directly.

Broadening the coverage of each of these planners raises a number of interesting research questions. For example, is it possible to exploit utility ranges to search for optimal plans in SNLP or Prodigy? How should plan compilation approaches deal with structured utility models? These approaches also have the potential to handle exogenous events gracefully although this has not yet been explored.

Solving Markov decision processes

In this section I review approaches to planning under uncertainty based on Markov decision processes (MDPs). My aim is to give the flavour of MDP approaches, the problems they engender and solutions being currently explored, and compare these approaches with those of the previous section. For a more thorough survey of MDP-based approaches, see (Boutilier, Dean, & Hanks 1998). After a brief description of MDP algorithms I show how at a broad level the two approaches are solving the same problem, and describe four approaches to exploiting structured representations of state, action and utility within the MDP framework.

Overview of Markov Decision Processes

This description of Markov decision processes follows (Littman 1996) and (Boutilier, Dean, & Hanks 1995). A Markov decision process M is a tuple $M =< S, \mathcal{A}, \Phi, R >$ where

- S is a finite set of states of the system.
- $\mathcal{A}$ is a finite set of actions.
- $\Phi : A \times S \rightarrow \Pi(S)$ is the *state transition function*, mapping an action and a state to a probability distribution over S for the possible resulting state. The probability of reaching state s' by performing action a in state s is written $\Phi(a, s, s')$.
- $R : S \times A \rightarrow \mathcal{R}$ is the *reward function*. $R(s, a)$ is the reward the system receives if it takes action a in state s.

A *policy* for an MDP is a mapping $\pi : S \rightarrow A$ that selects an action for each state. Given a policy, we can define its finite-horizon value function $V_n^\pi : S \rightarrow \mathcal{R}$, where $V_n^\pi(s)$ is the expected value of applying the policy π for n steps starting in state s. This is defined inductively with $V_0^\pi(s) = R(s, \pi(s))$ and

$$V_m^\pi(s) = R(s, \pi(s)) + \sum_{u \in S} \Phi(\pi(s), s, u) V_{m-1}^\pi(u)$$

Over an infinite horizon, a discounted model is frequently used to ensure policies have a bounded expected value. For some β chosen so that $\beta < 1$, the value of any reward from the transition after the next is discounted by a factor of β and the one after that by a factor of β^2, and so on. Thus if $V^{\pi}(s)$ is the discounted expected value in state s following policy π forever, we must have

$$V^{\pi}(s) = R(s, \pi(s)) + \beta \sum_{u \in S} \Phi(\pi(s), s, u) V^{\pi}(u)$$

This yields a set of linear equations in the values of $V^{\pi}()$.

A solution to an MDP is a policy that maximizes its expected value. For the discounted infinite-horizon case with any given discount factor β, there is a policy V^* that is optimal regardless of the starting state (Howard 1960), that satisfies the following equation:

$$V^*(s) = \max_a \{R(s, a) + \beta \sum_{u \in S} \Phi(a, s, u) V^*(u)\}$$

Two popular methods for solving this equation and finding an optimal policy for an MDP are *value iteration* and *policy iteration* (Puterman 1994).

In policy iteration, the current policy is repeatedly improved by finding some action in each state that has a higher value than the action chosen by the current policy for that state. The policy is initially chosen at random, and the process terminates when no improvement can be found. The algorithm is shown in Table 2. This process converges to an optimal policy (Puterman 1994).

```
Policy-Iteration($S, \mathcal{A}, \Phi, R, \beta$):
1. For each $s \in S$, $\pi(s) =$ RandomElement($\mathcal{A}$)
2. Compute $V^{\pi}(.)$
3. For each $s \in S$ {
4.     Find some action $a$ such that
           $R(s, a) + \beta \sum_{u \in S} \Phi(a, s, u) V^{\pi}(u) > V^{\pi}(s)$
5.     Set $\pi^{'}(s) = a$ if such an $a$ exists,
6    otherwise set $\pi^{'}(s) = \pi(s)$.
   }
7. If $\pi^{'}(s) \neq \pi(s)$ for some $s \in S$ goto 2.
8. Return $\pi$
```

Table 2: The policy iteration algorithm

In value iteration, optimal policies are produced for successively longer finite horizons, until they converge. It is relatively simple to find an optimal policy over n steps $\pi^*_n(.)$, with value function $V^*_n(.)$, using the recurrence relation:

$$\pi^*_n(s) = \text{argmax}_a \{R(s, a) + \beta \sum_{u \in S} \Phi(a, s, u) V^*_{n-1}(u)\}$$

with starting condition $V^*_0(s) = 0 \forall s \in S$, where V^*_m is derived from the policy π^*_m as described above. Table 3 shows the value iteration algorithm, which takes an MDP, a discount value β and a parameter ϵ and produces successive finite-horizon optimal policies, terminating when the maximum change in values between the current and previous value functions is below ϵ. It can also be shown that the algorithm converges to the optimal policy for the discounted infinite case in a number of steps that is polynomial in $|S|$, $|\mathcal{A}|$, log $\max_{s,a}|R(s, a)|$ and $1/(1 - \beta)$.

Planning under uncertainty with MDPs

Solving a Markov decision process is essentially the same problem as planning under uncertainty discussed in the first part of this article, with some minor differences. The standard algorithms for MDPs find a policy,

```
**Value-Iteration**($S, \mathcal{A}, \Phi, R, \beta, \epsilon$):
1. for each $s \in S$, $V_0(s) = 0$
2. $t = 0$
3. $t = t + 1$
4. for each $s \in S$ {
5.     for each $a \in A$
6.         $Q_t(s,a) = R(s,a) + \beta \sum_{u \in S} \Phi(a,s,u) V_{t-1}(u)$
7.     $\pi_t(s) = \text{argmax}_a Q_t(s,a)$
8.     $V_t(s) = Q_t(s, \pi_t(s))$
   }
9. if ( $\max_s |V_t(s) - V_{t-1}(s)| \geq \epsilon$) goto 3
10. return $\pi_t$
```

Table 3: The value iteration algorithm

which chooses an action for every possible state of the underlying system, while methods based on classical planners expect a set of possible initial states as input to the problem, and find a sequence of actions (possible branching) based on them. The difference is partly because the MDP solution methods emphasized the use of dynamic programming while AI planning methods emphasized the use of structured representations for the state space and for actions. However the improvements to policy and value iteration discussed below exploit structured representations.

Another difference is in the specification of goals. Markov decision processes attach a reward to an action in a state, while classical planning takes a logical description of a set of states as a goal. Some of the planners described in the last section attach utilities to goals, however, and again as the MDP algorithms described below make use of structure their objective functions become more goal-like.

The standard MDP algorithms seek a policy with maximum expected utility. Half of the planners of the last section do this while half seek a plan that passes a threshold probability of success. Again this is a minor difference, because given a thresholding planner one could produce an optimizing planner by repeatedly increasing the threshold until no plan is found. Conversely one could terminate policy iteration when a threshold value is reached.

Policy iteration and value iteration can find optimal policies in polynomial time in the size of the state space of the MDP. However, this state space is usually exponentially large in the inputs to a planning problem, which includes a set of literals whose cross product describes the state space. Attempts to build on these and other techniques for solving MDPs have concentrated on ways to gain leverage from the structure of the planning problem to reduce the computation time required.

Partitioning the state space and factored state representations.

Dean et al. used policy iteration in a restricted state space called an *envelope* (Dean *et al.* 1993). A subset of the states is selected, and each transition in the MDP that leaves the subset is replaced with a transition to a new state `OUT` with zero reward. No transitions leave the `OUT` state. They developed an algorithm that alternated between solving the restricted-space MDP with policy iteration and expanding the envelope by including the n most likely elements of the state space to be reached by the optimal policy that were not in the envelope. The algorithm converges to an optimal policy considerably more quickly than standard policy iteration on the whole state space, but as the authors point out (Dean *et al.* 1995), it makes some assumptions that limit its applicability, including a sparse MDP in which each state has only a small number of outward transitions. Tash and Russell extend the idea of an envelope with an initial estimate of distance-to-goal for each state and a model that takes the time of computation into account (Tash & Russell 1994).

Abstractions and hierarchical approaches

While the envelope extension method ignores portions of the state space, other techniques have considered abstractions of the state space that try to group together sets of states that behave similarly under the chosen actions of the optimal policy. Boutilier and Dearden (Boutilier & Dearden 1994) assume a representation for actions that is similar to that of Buridan (Kushmerick, Hanks, & Weld 1994) described earlier and a state utility function that is described in terms of domain literals. They then pick a subset of the literals that accounts for the greatest variation in the state utility and use the action representation to find literals that can directly or indirectly affect the chosen set, using a technique similar to the one developed by Knoblock for building abstraction hierarchies for classical planners (Knoblock 1991). This subset of literals then forms the basis for an abstract MDP by projection of the original states. Since the state space size is exponential in the set of literals, this reduction can lead to considerable time savings over the original MDP. Boutilier and Dearden prove bounds on the difference in value of the abstract policy compared with an optimal policy in the original MDP.

Dean and Lin refine this idea by splitting the MDP into subsets and allowing a different abstraction of the states to be considered in each one (Dean & Lin 1995). This approach can provide a better approximation because typically different literals may be relevant in different parts of the state space. However there is an added cost to re-combining the separate pieces unless they happen to decompose very cleanly. Dean and Lin assume the partition of the state space is given by some external oracle.

Factored action and state representations

While the last approaches exploited structure in the state utility description it is also possible to exploit it in the action description. Boutilier et al. extend *modified policy iteration* to propose a technique called *structured policy iteration* that makes use of a structured action representation in the form of 2-stage Bayesian networks (Boutilier, Dearden, & Goldszmidt 1995). The representation of the policy and utility functions are also structured, using decision trees. In standard policy iteration, the value of the candidate policy is computed on each iteration by solving a system of $|S|$ linear equations (step 2 in Table 2), which is computationally prohibitive for large real-world planning problems. Modified policy iteration replaces this step with an iterative approximation of the value function V_π by a series of value functions $V^0, V^1, \ldots$ given by

$$V^i(s) = R(s) + \beta \sum_{u \in S} \Phi(\pi(s), s, u) V^{i-1}(u)$$

Stopping criteria are given in (Puterman 1994).

In structured policy iteration, the value function is again built in a series of approximations, but in each one it is represented as a decision tree over the domain literals. Similarly the policy is built up as a decision tree. On each iteration, new literals might be added to these trees as a result of examining the literals mentioned in the action specification Φ and utility function R. In this way the algorithm avoids explicitly enumerating the state space.

Using classical planning ideas to help approximate MDPs

Structured policy iteration makes use of a factored action representation that is similar to actions in classical planning. It is also possible to make use of causal links and the planning graphs used in Graphplan (Blum & Furst 1997). In (Boutilier, Brafman, & Geib 1997), Boutilier, Brafman and Geib decompose the reward function of the MDP into components and produce policies for them separately. They then make a causal link analysis of each policy to produce partially ordered action sets from them using UCPOP. The flexibility in the partial order makes it easier to merge the component policies into a policy for the original reward function, again much as an SNLP based planner such as UCPOP might merge subplans for individual goals. In (Boutilier, Brafman, & Geib 1998), a reachability analysis inspired by Graphplan is used to restrict the states considered for policy creation given an initial state.

Givan and Dean (Givan & Dean 1997) show that STRIPS-style goal regression computes an approximate minimized form of the finite state automaton corresponding to the problem. In (Dean & Givan 1997) the authors show how to use model minimization techniques to solve MDPs.

Partial observability

Similar work has been done with *partially-observable Markov decision processes* or POMDPs, in which the assumption of complete observability is relaxed. In a POMDP there is a set of observation labels $\mathcal{O}$ and a set of conditional probabilities $P(o|a, s), o \in \mathcal{O}, a \in \mathcal{A}, s \in S$, such that if the system makes a transition to state s with action a it receives the observation label o with probability $P(o|a, s)$. Cassandra et al. introduce the *witness algorithm* for solving POMDPs (Cassandra, Kaelbling, & Littman 1994). A standard technique for finding an optimal policy for a POMDP is to construct the MDP whose states are the belief states of the original POMDP, ie each state is a probability distribution over states in the POMDP, with beliefs maintained based on the observation labels using Bayes' rule. A form of value iteration can be performed in this space making use of the fact that each finite-horizon policy will be convex and piecewise-linear. The witness algorithm includes an improved technique for updating the basis of the convex value function on each iteration. Parr and Russell use a smooth approximation of the value function that can be updated with gradient descent (Parr & Russell 1995). Brafman introduces a grid-based method in (Brafman 1997). Although work on POMDPs is promising, it is still preliminary and can only be used to solve small POMDP problems (Brafman 1997).

Conclusions

After introducing decision-theoretic planning, this paper discussed four different approaches based on extending classical planning algorithms. The approaches all use different planning styles, and attack different aspects of the decision-theoretic planning problem. Approaches based on algorithms for solving Markov decision processes were briefly discussed, emphasizing extensions that make use of factored action representations, causal links, abstraction and other ideas from classical planning. There are interesting directions still to be explored that come from the interplay between the two approaches. Surprisingly, in improving MDP approaches little attention has been paid to the DRIPS style of planning, although it is the only classically-inspired planner mentioned here that aims to solve the same problem as an MDP: maximizing expected utility. Using dominance to eliminate portions of the state or action space may prove fruitful in MDP approaches. Similarly, few classical planning algorithms make use of local improvement search or dynamic programming, although some algorithms for SAT-compilation planning perform local improvements.

A blind spot shared by both approaches is the reliance on complete, detailed domain models. In many real domains, specifying full probability distributions for all actions and the initial state would be at best tedious, and typically impossible. There has been relatively little work in this area, but see a recent workshop on interactive decision-theoretic systems (Haddawy & Hanks 1998b).

It should also be noted that most of the systems described here either concentrate on parts of the decision-theoretic planning task or represent individual techniques that will need to be used in combination to achieve significant results. Scaling up in this way is one of the largest challenges ahead. The techniques developed so far have been shown to be practical in some domains (*e.g* (Blythe 1998; Haddawy, Doan, & Kahn 1996)) but many simple domains still lie beyond our current capabilities. Still, the wealth of ideas being proposed show that this is an exciting time to be working in decision-theoretic planning.

References

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

Blythe, J., and Veloso, M. 1997. Using analogy in conditional planners. In *Proc. Fourteenth National Conference on Artificial Intelligence*. AAAI Press.

Blythe, J. 1994. Planning with external events. In de Mantaras, R. L., and Poole, D., eds., *Proc. Tenth Conference on Uncertainty in Artificial Intelligence*, 94–101. Seattle, WA: Morgan Kaufmann. Available as http://www.cs.cmu.edu/j̃blythe/papers/uai94.ps.

Blythe, J. 1995. The footprint principle for heuristics for probabilistic planners. In Ghallab, M., and Milani, A., eds., *New Directions in AI Planning*, 173–185. Assissi, Italy: IOS Press.

Blythe, J. 1996. Decompositions of markov chains for reasoning about external change in planners. In Drabble, B., ed., *Proc. Third International Conference on Artificial Intelligence Planning Systems*, 27–34. University of Edinburgh: AAAI Press.

Blythe, J. 1998. *Planning Under Uncertainty in Dynamic Domains*. Ph.D. Dissertation, Carnegie Mellon University Computer Science Department.

Blythe, J. 1999. Decision-theoretic planning. *AI Magazine* 20(2).

Boutilier, C., and Dearden, R. 1994. Using abstractions for decision-theoretic planning with time constraints. In *Proc. Twelfth National Conference on Artificial Intelligence*, 1016–1022. AAAI Press.

Boutilier, C.; Brafman, R. I.; and Geib, C. 1997. Prioritized goal decomposition of markov decision processes: Toward a synthesis of classical and decision theoretic planning. In *Proc. 15th International Joint Conference on Artificial Intelligence*. Nagoya, Japan: Morgan Kaufmann.

Boutilier, C.; Brafman, R.; and Geib, C. 1998. Structured reachability analysis for markov decision processes. In *Proc. Fourtheenth Conference on Uncertainty in Artificial Intelligence*. Madison, Wisconsin: Morgan Kaufmann.

Boutilier, C.; Dean, T.; and Hanks, S. 1995. Planning under uncertainty: structural assumptions and computational leverage. In Ghallab, M., and Milani, A., eds., *New Directions in AI Planning*, 157–172. Assissi, Italy: IOS Press.

Boutilier, C.; Dean, T.; and Hanks, S. 1998. Planning under uncertainty: structural assumptions and computational leverage. *Journal of Artificial Intelligence Research* in press.

Boutilier, C.; Dearden, R.; and Goldszmidt, M. 1995. Exploiting structure in policy construction. In *Proc. 14th International Joint Conference on Artificial Intelligence*, 1104–1111. Montréal, Quebec: Morgan Kaufmann.

Brafman, R. 1997. A heuristic variable grid solution method of pomdps. In *Proc. Fourteenth National Conference on Artificial Intelligence*, 727–733. AAAI Press.

Cassandra, A. R.; Kaelbling, L. P.; and Littman, M. L. 1994. Acting optimally in partially observable stochastic domains. In *Proc. Twelfth National Conference on Artificial Intelligence*, 1023–1028. AAAI Press.

D'Ambrosio, B. 1999. Inference in bayesian networks. *AI Magazine* 20(2).

Davis, M.; Logemann, G.; and Loveland, D. 1962. A machine program for theorem proving. *Communications of the ACM* 5:394–397.

Dean, T., and Givan, R. 1997. Model minimization in markov decision processes. In *Proc. Fourteenth National Conference on Artificial Intelligence*, 106–111. AAAI Press.

Dean, T., and Lin, S.-H. 1995. Decomposition techniques for planning in stochastic domains. In *Proc. 14th International Joint Conference on Artificial Intelligence*, 1121 – 1127. Montréal, Quebec: Morgan Kaufmann.

Dean, T.; Kaelbling, L. P.; Kirman, J.; and Nicholson, A. 1993. Planning with deadlines in stochastic domains. In *National Conference on Artificial Intelligence*, National Conference on Artificial Intelligence.

Dean, T.; Kaelbling, L.; Kirman, J.; and Nicholson, A. 1995. Planning under time constraints in stochastic domains. *Artificial Intelligence* 76(1-2):35–74.

Draper, D.; Hanks, S.; and Weld, D. 1994. Probabilistic planning with information gathering and contingent execution. In Hammond, K., ed., *Proc. Second International Conference on Artificial Intelligence Planning Systems*, 31–37. University of Chicago, Illinois: AAAI Press.

Feldman, J. A., and Sproull, R. F. 1977. Decision theory and artificial intelligence ii: The hungy monkey. *Cognitive Science* 1:158–192.

Fikes, R. E., and Nilsson, N. J. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

Friedland, P. E., and Iwasaki, Y. 1985. The concept and implementation of skeletal plans. *Journal of Automated Reasoning* 1(2):161–208.

Givan, R., and Dean, T. 1997. Model minimization, regression, and propositional strips planning. In *Proc. 15th International Joint Conference on Artificial Intelligence.* Nagoya, Japan: Morgan Kaufmann.

Goldman, R. P., and Boddy, M. S. 1994. Epsilon-safe planning. In de Mantaras, R. L., and Poole, D., eds., *Proc. Tenth Conference on Uncertainty in Artificial Intelligence*, 253–261. Seattle, WA: Morgan Kaufmann.

Goodwin, R., and Simmons, R. 1998. Search control of plan generation in decision-theoretic planning. In Reid Simmons, M. V., and Smith, S., eds., *Proc. Fourth International Conference on Artificial Intelligence Planning Systems*, 94–101. Carnegie Mellon University, Pittsburgh, PA: AAAI Press.

Haddawy, P., and Hanks, S. 1998a. Utility models for goal-directed decision-theoretic planners. *Computational Intelligence* 14(3):392–429.

Haddawy, P., and Hanks, S., eds. 1998b. *Working notes of the AAAI Spring Symposium on Interactive and Mixed-Initiative Decision-Theoretic Systems*. Stanford: AAAI Press.

Haddawy, P., and Suwandi, M. 1994. Decision-theoretic refinement planning using inheritance abstraction. In Hammond, K., ed., *Proc. Second International Conference on Artificial Intelligence Planning Systems*. University of Chicago, Illinois: AAAI Press.

Haddawy, P.; Doan, A.; and Goodwin, R. 1995. Efficient decision-theoretic planning: Techniques and empirical analysis. In Besnard, P., and Hanks, S., eds., *Proc. Eleventh Conference on Uncertainty in Artificial Intelligence*, 229–326. Montreal, Quebec: Morgan Kaufmann.

Haddawy, P.; Doan, A.; and Kahn, C. E. 1996. Decision-theoretic refinement planning in medical decision making: Management of acute deep venous thrombosis. *Medical Decision Making* 16(4):315–325.

Howard, R. A. 1960. *Dynamic Programming and Markov Processes*. MIT Press.

Kautz, H. A., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proc. Thirteenth National Conference on Artificial Intelligence*. AAAI Press.

Knoblock, C. A. 1991. *Automatically Generating Abstractions for Problem Solving*. Ph.D. Dissertation, Carnegie Mellon University.

Kushmerick, N.; Hanks, S.; and Weld, D. 1994. An algorithm for probabilistic least-commitment planning. In *Proc. Twelfth National Conference on Artificial Intelligence*, 1073–1078. AAAI Press.

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

Littman, M. L.; Goldsmith, J.; and Mundhenk, M. 1998. The computational complexity of probabilistic planning. *Journal of Artificial Intelligence Research* 9:1–36.

Littman, M. L. 1996. *Algorithms for Sequential Decision Making*. Ph.D. Dissertation, Department of Computer Science, Brown University.

Luce, R. D., and Raiffa, H. 1957. *Games and Decisions: Introduction and Critical Survey*. Wiley.

Majercik, S. M., and Littman, M. L. 1998. Maxplan: A new approach to probabilistic planning. In Reid Simmons, M. V., and Smith, S., eds., *Proc. Fourth International Conference on Artificial Intelligence Planning Systems*, 86–93. Carnegie Mellon University, Pittsburgh, PA: AAAI Press.

McDermott, D. 1992. Transformational planning of reactive behavior. Technical Report YALEU/CSD/RR/941, Yale University.

Newell, A., and Simon, H. A. 1963. Gps: a program that simulates human thought. In Feigenbaum, E. A., and Feldman, J., eds., *Computers and Thought*, 279–293. New York: McGraw-Hill.

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

Parr, R., and Russell, S. 1995. Approximating optimal policies for partially observable stochastic domains. In *Proc. 14th International Joint Conference on Artificial Intelligence*. Montréal, Quebec: Morgan Kaufmann.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann.

Penberthy, J. S., and Weld, D. S. 1992. Ucpop: A sound, complete, partial order planner for adl. In *Third International Conference on Principles of Knowledge Representation and Reasoning*, 103–114.

Peot, M. A., and Smith, D. E. 1992. Conditional nonlinear planning. In Hendler, J., ed., *Proc. First International Conference on Artificial Intelligence Planning Systems*, 189–197. College Park, Maryland: Morgan Kaufmann.

Pérez, M. A., and Carbonell, J. 1994. Control knowledge to improve plan quality. In Hammond, K., ed., *Proc. Second International Conference on Artificial Intelligence Planning Systems*, 323–328. University of Chicago, Illinois: AAAI Press.

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

Puterman, M. 1994. *Markov Decision Processes : Discrete Stochastic Dynamic Programming*. John Wiley & Sons.

Smith, D. E., and Weld, D. 1998. Conformant graphplan. In *Proc. Fifteenth National Conference on Artificial Intelligence*, 889–896. AAAI Press.

Tash, J. K., and Russell, S. 1994. Control strategies for a stochastic planner. In *Proc. Twelfth National Conference on Artificial Intelligence*, 1079–1085. AAAI Press.

Tate, A. 1977. Generating project networks. In *International Joint Conference on Artificial Intelligence.*

Veloso, M.; Carbonell, J.; Pérez, A.; Borrajo, D.; Fink, E.; and Blythe, J. 1995. Integrating planning and learning: The prodigy architecture. *Journal of Experimental and Theoretical AI* 7:81–120.

Weld, D.; Anderson, C.; and Smith, D. E. 1998. Extending graphplan to handle uncertainty and sensing actions. In *Proc. Fifteenth National Conference on Artificial Intelligence*, 897–904. AAAI Press.

Weld, D. 1994. A gentle introduction to least-commitment planning. *AI Magazine*.

Wellman, M. P.; Breese, J. S.; and Goldman, R. P. 1992. From knowledge bases to decision models. *Knowledge Engineering Review* 7(1):35–53.

Wellman, M. P. 1990. *Formulation of Tradeoffs in Planning Under Uncertainty*. Pitman.

Wilkins, D. E.; Myers, K. L.; Lowrance, J. D.; and Wesley, L. P. 1995. Planning and reacting in uncertain and dynamic environments. *Journal of Experimental and Theoretical AI* to appear.

Williamson, M., and Hanks, S. 1994. Optimal planning with a a goal-directed utility model. In Hammond, K., ed., *Proc. Second International Conference on Artificial Intelligence Planning Systems*, 176–181. University of Chicago, Illinois: AAAI Press.