Focusing Attention in Anytime Decision-Theoretic Planning *

Peter Haddawy
Department of Electrical Engineering and Computer Science
University of Wisconsin-Milwaukee
PO Box 784
Milwaukee, WI 53201
haddawy@cs.uwm.edu

Abstract

Any anytime algorithm used for decision-making should have the property that it considers the most important aspects of the decision problem first. In this way, the algorithm can first eliminate disastrous decisions and recognize particularly advantageous decisions, considering finer details if time permits. We view planning as a decision-making process and discuss the design of anytime algorithms for decision-theoretic planning. In particular, we present an anytime decision-theoretic planner that uses abstraction to focus attention first on those aspects of a planning problem that have the highest impact on expected utility. We discuss control schemes for refining this behavior and methods for automatically creating good abstractions. We present an intelligent agent architecture for integrating our anytime planner with an execution module and describe the status of the implementation.

Introduction

In the framework of decision-theoretic planning, uncertainty in the state of the world and in the effects of actions are represented with probabilities and the planner's goals, as well as tradeoffs among them, are represented with a utility function over outcomes. Given this representation, the objective is to find an optimal or near optimal plan, where optimality is defined as maximizing expected utility (EU). In this paper we will be concerned with optimal decision-theoretic planning.

We define an anytime decision-theoretic planning algorithm as one that can return some plan or set of candidate plans at anytime during its execution and during its execution the expected utility of that plan or set of plans never decreases. We also impose the constraint that for some executions of the planner the EU of the plan or plans increases during some phase of the deliberation. This assures us that we can benefit from investing time in deliberation. But these properties are not sufficient to produce partial deliberation that we would consider to be rational. We would like an anytime planning algorithm to have the property that it considers the most important aspects of the planning problem before moving on to the details. We call this the *rational refinement* property.

A straightforward approach to achieving the anytime property in decision-theoretic planning is to limit the depth to which plans are projected. This is the approach taken by Dearden and Boutilier (Dearden & Boutilier 1994). But this myopic approach can result in the well-known horizon problem in which a disastrous or highly fortuitous outcome lies just beyond the projection horizon. So it does not have the rational refinement property.

An alternative recently popular approach to anytime decision-theoretic planning has been to plan over an ever-increasing subset of world states (Drummond & Bresina 1990; Dean *et al.* 1993; Thiebaux *et al.* 1994). In this approach, a planning problem is represented as a Markov decision process, in which we have a finite number of world states, states are completely observable, and actions are represented as probabilistic transitions between states. We will describe the work of Dean, et.al. (1993) since it is nicely formalized and representative of this general approach. Their algorithm starts by choosing an initial subset of states called the *envelope*. They generate an optimal policy for this subset of states and assign a default action to any state outside the envelope. The algorithm continues by alternating between adding states to the envelope and generating optimal policies. It can be interrupted at any time to obtain the current best policy. The performance of this approach is highly sensitive to the selection of which states to include in the envelope. One idea is to use some domain knowledge to choose an initial envelope and then at each iteration to add to the envelope those states outside it which are most likely to be entered. But this strategy can result in myopic reasoning in which the planner may not recognize early on that a plan has a high or low EU, since low probability states can be significant if their utility is

*Thanks to Jim Helwig, AnHai Doan, and Mark Kaplan for helpful discussions. This work was partially supported by NSF grant IRI-9509165 and by a Sun Microsystems AEG award.

sufficiently extreme. To avoid this problem, one might think of adding states to the envelope based on both the probability of entering them and on the utility associated with them. But consider a medical planning problem in which the death of the patient is a possibility. In every state the patient is either dead or alive. If mortality is significantly more important that other factors, then every state will have a relatively low or high utility. So choosing states on the basis of their utility is of little help. Rather, the planner should be focusing on the attribute describing morbidity instead of focusing on any particular state. Utility extremes will be characterized by *attributes* rather than *states* whenever a large degree of utility independence exists among the attributes over which utility is defined. Because of the complexity of specifying utility functions in multiattribute problems (Keeney & Raiffa 1976), this will typically be the case. So a more reasonable approach is to reason over a subset of the attributes and to expand the subset of attributes as time allows.

How can we apply the above observation to the design of anytime decision-theoretic planning algorithms? Notice that an optimal decision-theoretic planner must either explicitly or implicitly compare the expected utilities of all possible plans, which means that such planners tend to work by eliminating suboptimal plans, since one cannot prove that a plan is optimal without proving that the alternatives are suboptimal. For an anytime decision-theoretic planning algorithm to have the rational refinement property, it should function as if using an information filter. It should initially focus its attention on those attributes which have a high impact on expected utility in order to eliminate highly sub-optimal plans. As it runs it should consider less important details which distinguish between similarly ranked plans. We propose an approach that functions in just this way.

The DRIPS decision-theoretic refinement planning system (Haddawy & Suwandi 1994) searches through the space of possible plans by using an abstraction hierarchy. By using abstraction, the planner is capable of eliminating suboptimal classes of plans without explicitly examining all the plans in a class. DRIPS finds the optimal plan by building abstract plans, comparing them, and refining only those that might yield the optimal plan. It begins with a set of abstract plans at the highest abstraction level, and subsequently refines the plans from more general to more specific. After each refinement, it eliminates suboptimal abstract plans. While an abstract plan is a complete plan in the sense that it is a complete sequence of actions, the actions in the plan are only partially specified in the sense that some of the attribute values are incompletely specified. In an appropriately structured abstraction hierarchy, the abstract action specifications describe only those aspects of the actions that have a high impact on expected utility. As the planner descends the abstraction hierarchy, it considers more of the less important details. The planner can be stopped at any time to yield the set of abstract plans which have not yet been shown to be suboptimal.

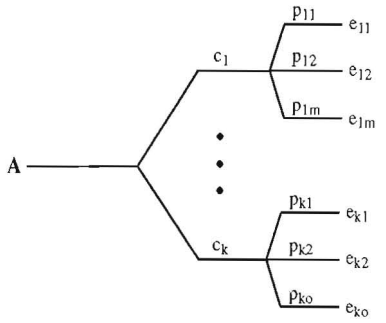

Figure 1: General form of an action description.

The rest of this paper is organized as follows. In the next section we describe the representation used by the planner. Then we present an overview of the abstraction theory used by the planner and present the planning algorithm. Finally, we discuss search control techniques for focusing the attention of the planner, as well as methods we are currently developing for generating appropriate abstraction hierarchies.

Representation

World Model Since we are interested in planning in temporal domains, we describe the world in terms of chronicles, where a chronicle is a complete specification of the world throughout time. We take time to be continuous and we describe chronicles by specifying the values of discrete and continuous attributes at various times, for example $\text{fuel}(t_0) = 10$. We express uncertainty concerning the state of the world with a set of probability distributions over chronicles. We express such a set by assigning probability intervals to attribute values at various times.

Action Model Actions are both conditional and probabilistic: under certain conditions an action will have a given effect with a given probability. An action is represented with a tree structure as shown in figure 1, where the c_i are a set of mutually exclusive and exhaustive conditions, the p_{ij} are probabilities which sum to one for fixed i, and the e_{ij} are effects.

We interpret the representation intuitively as saying that if one of the conditions holds at the beginning of the action then the effects on that branch are realized immediately following the action, with the specified probabilities. Each branch represents a mapping from the state before the action to a distribution over states after the action. This representation is similar to that used by Hanks (Hanks 1990). We assume that changes to the world are limited to those effects explicitly described in the agent's action descriptions, so we do not allow for exogenous events.

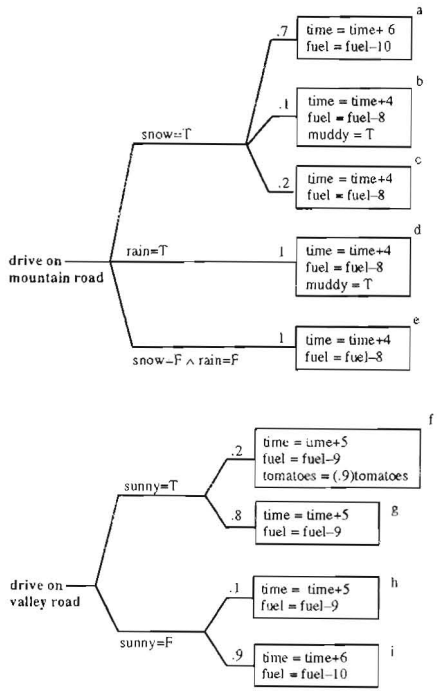

Figure 2: Example action descriptions.

To illustrate the action representation, consider the following example. We wish to deliver some tomatoes from a farm to a warehouse and we can take a mountain road or a valley road to make the delivery. If we take the mountain road, the length of time the drive takes, the fuel consumption, and whether the truck gets muddy depend on the weather conditions. The descriptions of these actions are shown in Figure 2. The conditions on the branches specify conditions that must be true just before the action is executed. The effects of the action are specified in terms of the duration of the action and the conditions that are changed by the action. For example, in the branch a $time = time+6$ indicates that the duration is 6 hours and $fuel = fuel - 10$ means that the fuel level after the action is 10 gallons less than before the action. Action effects can be specified as absolute or relative. In branch b we have the absolute effect $muddy{=}T$, and the relative effect $fuel = fuel - 8$. So branch b says that if it is snowing just prior to driving on the mountain road then there is a 10% chance that the drive will require 4 hours, and that the truck will consume 8 gallons of fuel and become muddy.

Decision-Theoretic Refinement Planning

Abstracting Actions

The DRIPS planner primarily uses two types of abstraction: interaction-abstraction and sequential abstrac-

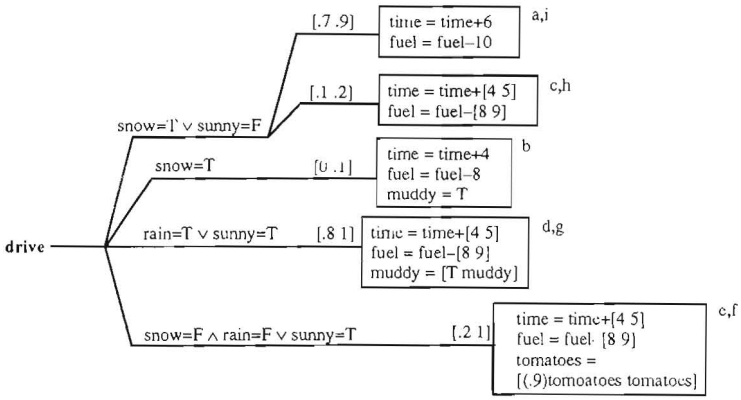

Figure 3: Example of inter-action abstraction. The letters labeling the branches indicate the pairings of the primitive action branches.

tion. The idea behind inter-action abstraction is to group together a set of analogous actions. The set is characterized by the features common to all the actions in the set. We then can plan with the abstract action and infer properties of a plan involving any of its instances. Formally, an inter-action abstraction of a set of actions $\{\mathsf{a}^1, \mathsf{a}^2, ...\mathsf{a}^n\}$ is an action that represents the disjunction of the actions in the set. The actions in the set are called the *instantiations* of the abstract action and are considered to be alternative ways of realizing the abstract action. Thus the a^i are assumed to be mutually exclusive.

To create an inter-abstraction of a set of actions $\{a_1, a_2, ..., a_n\}$ we do the following. Group the branches of the action descriptions into disjoint sets such that each set contains at most one branch from each action description. For each set s that contains fewer than n branches, add $n - |s|$ branches, each with the effect of one of the branches already in the set and with condition False and probability zero. The effect of an abstract branch is any sentence entailed by each of the effects of the branches in the set. The condition is the disjunction of the conditions on the branches in the set. The probability is specified as a range: the minimum of the probabilities of the branches in the set and the maximum of the probabilities of the branches in the set. Figure 3 shows an inter-action abstraction of the two action descriptions from Figure 2.

A sequential abstraction is essentially a macro operator that specifies the end effects of a sequence of actions, as well as the initial conditions under which those effects are achieved, without specifying changes that occur as intermediate steps due to the individual actions within the sequence. Thus the information about the state of the world during the execution of the sequence of actions is abstracted away. We abstract an action sequence $a_1 a_2$ by pairing every branch of a_1 with every branch of a_2 and create an abstract branch for

each pairing. The condition on the abstract branch is the conjunction of the condition on the branch of the first action and the condition on branch of the second action, regressed through the effects of the first action. If the conjunction is inconsistent, the abstract branch is not included in the abstract action description. The probability on the abstract branch is the product of the probabilities on the paired branches and the effect is the composition of the effects.

We have implemented tools that automatically create inter-action abstractions (Finigan 1995) and sequential abstractions (Doan & Haddawy 1995), given a specification of what actions to combine and how to group their branches. For a general theory of action abstraction which includes intra-action and sequential abstraction see (Doan 1995).

The DRIPS Planner

A planning problem is described in terms of an initial state distribution, a set of action descriptions, and a utility function. The space of possible plans is described by an abstraction/decomposition network, supplied by the user. An abstract action has one or more sub-actions, which themselves may be abstractions or primitive actions. A decomposable action has a subplan that must be executed in sequence. The description of the abstract actions are created by inter-action abstraction and the descriptions of the decomposible actions are created by sequential abstraction. An example network is shown in Figure 4. A plan is simply a sequence of actions obtained from the net. The planning task is to find the sequence of actions for those represented in the network that maximizes expected utility relative to the given probability and utility models.

DRIPS finds the optimal plan by building abstract plans, comparing them, and refining only those that might yield the optimal plan. It begins with a set of abstract plans at the highest abstraction level, and subsequently refines the plans from more general to more specific. Since projecting abstract plans results in inferring probability intervals and attribute ranges, an abstract plan is assigned an expected utility interval, which includes the expected utilities of all possible instances of that abstract plan. An abstract plan can be eliminated if the upper bound of its expected utility interval is lower than the lower bound of the expected utility interval for any other plan. Eliminating an abstract plan eliminates all its possible instantiations. When abstract plans have overlapping expected utility intervals, the planner refines one of the plans by instantiating one of its actions. Successively instantiating abstract plans will narrow the range of expected utility and allow more plans to be pruned.

Given the abstraction/decomposition network, we evaluate plans at the abstract level, eliminate suboptimal plans, and refine remaining candidate plans further until only optimal plans remain. The algorithm works as follows.

1. Create a plan consisting of the single top-level action and put it into the set plans.
2. Until there is no abstract plan left in plans,
   - Choose an abstract plan P. Refine P by replacing an abstract action in P with all its instantiations, or its decomposition, creating a set of lower level plans $\{P_1, P_2, ..., P_n\}$.
   - Compute the expected utility of all newly created plans.
   - Remove P from plans and add $\{P_1, P_2, ..., P_n\}$.
   - Eliminate suboptimal plans in plans.
3. Return plans as the set of optimal plans.

If run to completion, DRIPS explores the entire space of possible plans. Since the planner only eliminates plans it can prove are suboptimal and if a plan is suboptimal, it will eventually be eliminated by the planner, DRIPS is guaranteed to find the optimal plan or plans if it is run to completion.

As the algorithm proceeds, it maintains the set of abstract plans that cannot be eliminated because their expected utility intervals overlap. So the algorithm can be stopped at any time to yield the current set of candidate plans. From this set any abstract plan and any instance of that abstract plan can be chosen. Several strategies for choosing a plan from this candidate set are possible. For example, one could take an optimistic strategy and choose an instance from the abstract plan with the highest upper bound on EU or one could take a conservative strategy and select an instance from the abstract plan with the highest lower bound on EU. The maximum degree to which we are sacrificing optimality by executing the chosen plan can be determined by taking the difference between the highest upper bound on EU and the lower bound on EU of the abstract plan of which the chosen plan is an instance.

The DRIPS planner is implemented in CommonLisp and has been successfully applied to a number of problems. For example, we applied DRIPS to the medical decision problem of determining the optimal test/treat strategy for patients suspected of having lower extremity deep vein thrombosis (Haddawy, Doan, & Goodwin 1995; Doan, Haddawy, & Kahn 1995). This domain contained 6,206 possible strategies, involving different combinations of tests, conditional tests, and conditional treatments. DRIPS eliminated 5,551 (89%) of these strategies from consideration without examining them explicitly and successfully identified the optimal strategy[1].

Search Control Techniques

The extent to which DRIPS is able to eliminate suboptimal plans depends in large part on effectively con-

[1]The DRIPS code and the encoding of the described domain are available at http://www.cs.uwm.edu/faculty/haddawy.

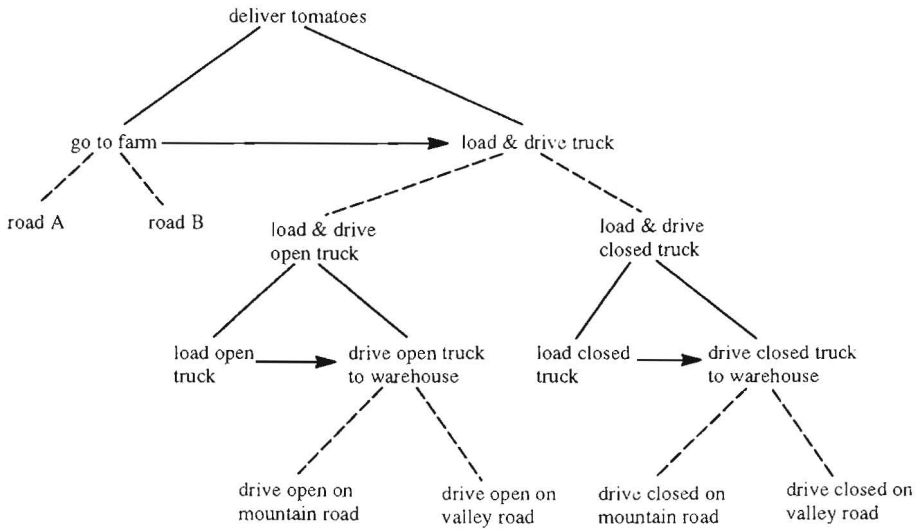

Figure 4: Abstraction/decomposition network. Abstraction relations are shown with dashed lines and decomposition relations are shown with solid lines.

trolling the search through the space of abstract plans. The algorithm above contains two non-deterministic choice points in its second step. The first choice is to select a plan from the set of abstract plans with overlapping expected utility. The second choice is to select an abstract action within the plan for expansion. Goodwin (1995) has shown that if we wish to find the optimal plan, the best strategy is to select the plan with the highest maximum EU. Selection of an action to expand is more difficult. Selecting actions that when expanded produce plans with greater reductions in the range of expected utility facilitates pruning and leads to more efficient planning. One strategy is to assign a fixed priority to each abstract action based how similar its instances are. The more similar the instances, the tighter the abstraction. One would like to delay expanding tight abstractions since they do not produce wide EU ranges. We will discuss how to automatically assign such priorities below. An alternative approach is to use sensitivity analysis at run time to select abstract actions whose expansion will produce the largest change in the EU range. In a recent paper (Haddawy, Doan, & Goodwin 1995) Goodwin presents techniques for performing this analysis and demonstrates their effectiveness on some large problems. The use of sensitivity analysis requires knowledge of how the narrowing of the range of an attribute effects the utility of a chronicle. Goodwin assumes that the user provides this function. The abstraction generation procedure we propose below could be used to generate such a function.

The best way to control search is largely dependent on how the anytime property of the planner is to be used. Consider an execution module which will execute the actions dictated by the planner. We can identify two extreme points along a spectrum describing the interaction between the planner and execution module. First, suppose the planner is given a fixed amount of time during which to plan and after this it must return the best plan, which will be executed in its entirety. This case is little different from simply finding the optimal plan and we can use the above search control techniques. The other possibility is that the planner will be used in a reactive fashion, deciding on a first action and while this is being executed, deciding on the next action. The execution module may also provide sensory feedback to the planner. In this case, we would like the planner to eliminate as many alternative first actions as possible, perhaps making little distinction between the later actions. To do this it is clear that we need to perform some refinement of the first action, but we are likely to also need to refine some of the later actions in order to get narrow enough EU ranges that we can eliminate some alternatives for the first action. For example, it would be possible for the planner to spend all its available time refining the first action down to the finest level of detail and for all the resulting plans to have overlapping EU intervals, in which case the planner would not be able to eliminate any initial actions. On the other hand, we do not want to make any distinctions later in the plan which do not help to resolve which initial action to choose. Within the context of choosing the first action we would like the rational refinement property to hold. For example,

we would like initial actions which commit us to disastrous courses of action to be eliminated as early as possible in the deliberation process.

Automatically Generating Abstractions

The DRIPS planning algorithm focuses its attention on the most important attributes first only if the abstraction hierarchy has been appropriately structured. We are currently exploring several approaches to automatically constructing useful abstraction hierarchies. In this section we discuss one approach. Throughout this discussion we will assume that utility is a function of only the final state of the world after execution of a plan and that action effects are conjunctions of propositions specifying assignments of particular values to attributes.

The world is described in terms of a set of attributes and our first task is to determine the impact that each attribute has on overall expected utility. We will end up with a set of weights assigned to the attributes. The basic idea is to start by assigning weights to the attributes that appear in the utility function and then to backward chain through the actions, assigning weights to the remaining attributes as they are encountered.

1. Let RA be the global set of relevant attributes, RA-old be the last set of relevant attributes, and RA-new be the new set of relevant attributes. Initialize RA and RA-old to be the set of all the attributes in the utility function, with weights proportional to their contribution to utility. For an additive utility function determining the weights is straightforward; for more complex utility functions this will be more difficult.
2. Identify the set of primitive actions such that all their effects appear in RA-old. The set of branches of each of these actions can be seen as specifying a correlation between the attributes in the conditions on the branches and the utility as specified by the effects. So for each action determine the correlation coefficients for the attributes in the conditions. Average these coefficients over all the actions[2] and add the attributes with their associated weights to RA-new.
3. RA = RA ∪ RA-new
   RA-old = RA-new
   RA-new = {}
4. If RA is changed, loop back to 2, otherwise exit.

This algorithm accounts for the fact that a condition is important only if it actually influences the value of an important attribute. We might have an important attribute that has the same value in all effects of a particular action. Clearly although the action is influential, the conditions of this action are not themselves important since they do not influence the value of this attribute.

We now have a set of relevant attributes RA with weights proportional to their impact on utility, which we can use to define a similarity measure on attribute sets. So given a set of actions, we can determine the best way to group their branches in order to produce an inter-action abstraction. This grouping problem is a three-dimensional clustering problem since we need to consider grouping conditions, effects, and probabilities. We can apply any one of a number of clustering algorithms. Although clustering is a computationally complex problem, we do not expect the number of branches of any action to exceed 100 so the complexity is not a serious concern.

We now show how to generate the hierarchy. Start with the set of primitive actions and produce the optimal abstraction for every pair of actions. Now using the attribute weights, compute a measure indicating how good each abstraction is, i.e. how similar the pairs of actions are. One possible measure is the sum of the distances of the paired branches. Using this measure, group together the most similar actions at the first level. Working with these abstractions, group together the most similar actions at the second level. Continue in this way till all actions have been abstracted together. The measure indicating the goodness of an abstraction can be stored with the abstract action and used as a priority value to control search as discussed in the previous section. Also the weights on the attributes can be used to generate a function indicating the effect of changing an attribute range on the utility of a chronicle, to be used by the sensitivity analysis control scheme.

The process of generating the abstraction hierarchy is complicated by the fact that an abstraction/decomposition network contains ordering information (decompositions) in addition to the abstraction information. This ordering information can be provided to the abstraction generation process. For example, suppose we wish to determine the optimal plan for testing and treating a disease. Any plan will consist of first testing and then treating. So we can specify that the set of test actions will precede the set of treat actions. Now within each set we can produce an abstraction hierarchy. The final abstraction/decomposition network will have a decomposition at the top level into test and treat. Each of these actions will be the top-level abstraction in a hierarchy or test and treat actions, respectively.

Notice that the effectiveness of the abstraction hierarchy generated by the proposed procedure is sensitive to the both the primitive action descriptions and the utility function. Although the action descriptions are not likely to change across problem instances, the utility function is. Fortunately, empirical results (Haddawy, Doan, & Goodwin 1995) have shown that if the changes in the utility function are not drastic, e.g., a

[2]The justification for averaging the coefficients is that we assume each action has an equal chance of being included in a plan.

change leaves the relative importance of the attributes unchanged, then the hierarchy will still be effective.

Intelligent Agent Architecture

In order to test the usefulness of DRIPS for realtime robot control, we are incorporating it into an intelligent agent architecture that includes the planner, an executor, and a controller.

The controller coordinates the planning and execution modules. It makes decisions regarding the tasks the modules should perform, as well as how much time to allocate for each task and what data to transfer between the modules. Typical commands to the planner include:

- send current set of candidate plans
- continue planning
- wait
- update set of candidate plans
- load new abstraction/decomposition network
- update world model

Typical commands to the executor include:

- obtain sensor information
- continue execution
- change execution
- wait

At this time the controller allows the planner to plan for a given amount of time, selects a next action to perform, and tells the planner to continue planning while the executor attempts to complete the action.

The DRIPS planner described earlier has been modified in two primary ways in order to support interleaving of planning with execution: interruptability of the algorithm, and updating of the world model. The main loop in DRIPS consists of choosing a plan to refine, refining that plan and then eliminating plans known to be sub-optimal. Our initial changes allow for communication to occur with the controller at the top of this loop. Additional changes could be made to the planner which would allow for the planner to be interrupted at any point.

After a sensing action is performed, the planner's world model must be updated to reflect the information gained about the world. This is done by manipulating the projection trees for the current set of candidate plans. After a sensing action is performed, the only relevant candidate plans (and hence projection trees) will be those that include the sensing action at the appropriate point in the plan. Those projection trees are updated by a simple two-step process. First we eliminate those branches emanating from the sensing action that represent outcomes other than the outcome that was realized. Then we renormalize the probabilities of the remaining branches so that the probabilities sum to one. This process is equivalent to conditioning the distribution over the branches on the outcome of the sensing action.

The executor provides the interface between the controller and the real world. The details are highly dependent on the implementation environment. We are currently interfacing to the TRUCKWORLD simulator (Hanks, Pollack, & Cohen 1993). TRUCKWORLD is a multiagent test bed designed to test theories of reactive execution. It simulates a truck with sensors driving around a world consisting of roads and locations in which events occur dependent on the characteristics of the world and on chance. Because TRUCKWORLD takes commands at a very low level, the primitive actions in the planner's hierarchy represent sequences of TRUCKWORLD actions. The executor translates from DRIPS primitive actions to TRUCKWORLD actions and coordinates the execution of each part of the sequence, as well as attempting to determine whether or not the DRIPS action was successfully executed. The executor also relays sensory information to the controller. The controller is responsible for processing that information.

An effort has been made to make the three modules as independent as possible to ensure optimal use of time. So each runs as a separate process, communicating via bi-directional streams. All code is written in Allegro CommonLisp.

Related Work

Our proposed approach is similar to that of Boutilier and Dearden (1994). They work within a Markov decision process framework and cluster together states which differ in irrelevant attributes into abstract states. They then generate a policy over the reduced state space. The clustering of the state space is based on classifying the attributes as either relevant or irrelevant, rather than assigning weights. They decide on relevance by starting with a set of directly relevant attributes and backward chaining through the action descriptions. For a given abstraction of the state space, they can quantify the degree of approximation of the policy generated. Their planning algorithm has the contract anytime property: given a time limit, it can produce an abstract state space such that it can return the optimal policy over that space in the time required. For some problems this is sufficient but for problems in which the time limit is not known a priori we need an algorithm which continuously refines its current best answer, as does the DRIPS algorithm. Their approach is also limited by the assumptions of a finite state space and of complete observability.

Current and Future Research

We are currently in the process of implementing two different techniques for generating abstraction hierarchies. One is based on the idea of generating a set of weights and doing clustering, while the other is based

on the idea of using a rough projection algorithm to estimate the loss due to abstraction. We are also adding to the planner the capability to reason about continuous decision variables so that, for example, the planner can reason not only about what route the truck should take but also at what speed it should travel.

References

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

Dean, T.; Kaelbling, L. P.; Kirman, J.; and Nicholson, A. 1993. Planning with deadlines in stochastic domains. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 574–579.

Dearden, R., and Boutilier, C. 1994. Integrating planning and execution in stochastic domains. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, 162–169.

Doan, A., and Haddawy, P. 1995. Generating macro operators for decision-theoretic planning. In *Working Notes of the AAAI Spring Symposium on Extending Theories of Action*, 68–73.

Doan, A.; Haddawy, P.; and Kahn, Jr, C. 1995. Decision-theoretic refinement planning: A new method for clinical decision analysis. In *Proceedings of the 19th Annual Symposium on Computer Applications in Medical Care (SCAMC95)*, 299-303.

Doan, A. 1995. An abstraction-based approach to decision-theoretic planning for partially observable metric domains. Master's thesis, Dept. of EE & CS, University of Wisconsin-Milwaukee.

Drummond, M., and Bresina, J. 1990. Anytime synthetic projection: Maximizing the probability of goal satisfaction. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 138–144.

Finigan, M. 1995. Knowledge acquisition for decision-theoretic planning. Master's thesis, Dept. of EE & CS, Univ. of Wisconsin-Milwaukee.

Haddawy, P., and Suwandi, M. 1994. Decision-theoretic refinement planning using inheritance abstraction. In *Proceedings, Second International Conference on AI Planning Systems*, 266-271.

Haddawy, P.; Doan, A.; and Goodwin, R. 1995. Efficient decision-theoretic planning: Techniques and empirical analysis. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, 229–236.

Hanks, S.; Pollack, M.; and Cohen, P. 1993. Benchmarks, test beds, controlled experimentation, and the design of agent architectures. *AI Magizine* 14(4):17–42.

Hanks, S. 1990. *Projecting Plans for Uncertain Worlds*. Ph.D. Dissertation, Yale University.

Keeney, R., and Raiffa, H. 1976. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. New York: Wiley.

Thiebaux, S.; Hertzberg, J.; Shoaff, W.; and Schneider, M. 1994. A stochastic model of actions and plans for anytime planning under uncertainty. *International Journal of Intelligent Systems*. To appear.