Focusing Attention in Anytime Decision-Theoretic Planning *

Peter Haddawy
Decision Systems and Artificial Intelligence Lab
Department of Electrical Engineering and Computer Science
University of Wisconsin-Milwaukee
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 planning algorithm 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.

1 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 or set of plans, where optimality is defined as maximizing expected utility (EU).

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 such that the EU of the plan or the lowest EU in the set of candidate plans is a monotonically increasing function of computation time. This definition is similar to Dean and Boddy's [1988] definition of anytime algorithm and Horvitz's [1988] definition of incremental-refinement policy. 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. We define an *optimal* anytime decision-theoretic planning algorithm as one for which the plan or set of candidate plans converges to the optimal plan or set of plans. In this paper we will be concerned with optimal anytime decision-theoretic planning.

*Thanks to AnHai Doan, Manonton Butarbutar, Vu Ha, and Mark Kaplan for helpful discussions. This work was partially supported by NSF grant IRI-9509165.

The above basic properties of anytime planning algorithms 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. The quality of the candidate plans produced by an anytime planning algorithm with this property will be characterized by rapid improvement that tapers off as the algorithm is run. In other words, the algorithm will have a performance profile with a steep initial increase.

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 and 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 and 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 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 and 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 optimal 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 and 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. The instances of this set of abstract plans constitutes a set of candidate plans that satisfies the conditions for an anytime decision-theoretic planning algorithm. Thus DRIPS is an optimal anytime decision-theoretic planner.

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.

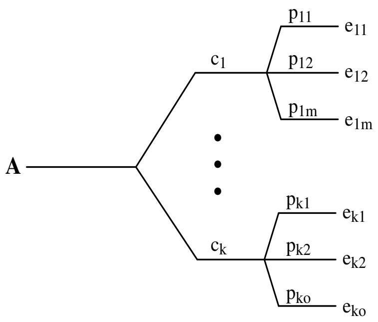

Figure 1: General form of an action description.

2 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 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.

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*

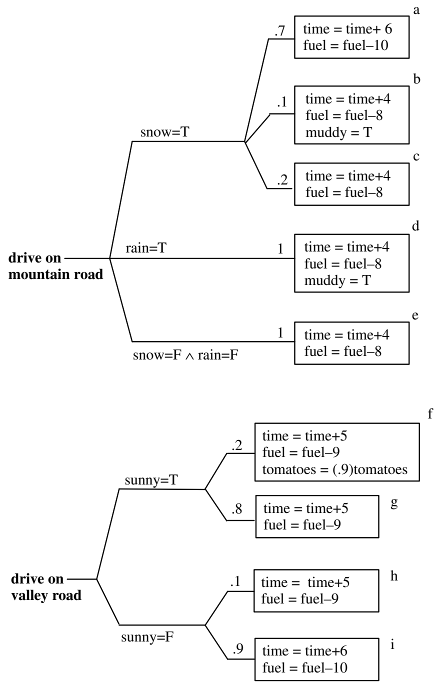

Figure 2: Example action descriptions.

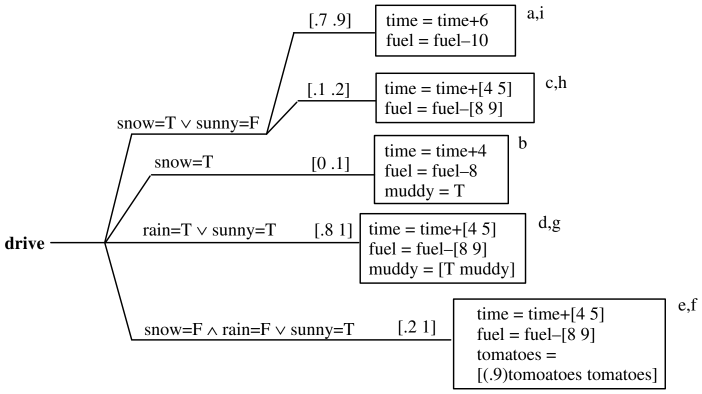

Figure 3: Example of inter-action abstraction. The letters labeling the branches indicate the pairings of the primitive action branches.

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.

3 Decision-Theoretic Refinement Planning

3.1 Abstracting Actions

The DRIPS planner primarily uses two types of abstraction: interaction-abstraction and sequential abstraction. 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 two abstraction tools that partially automate the process of creating abstract actions [Finigan, 1995]. An inter-action abstraction tool generates abstract action descriptions given a specification of which actions to combine and how to group their branches. A composition tool generates a decomposable action given a sequence of actions to compose [Doan and Haddawy, 1995]. The abstraction tools virtually eliminate the possibility of introducing errors in the process of constructing abstractions, since abstracting actions in the ways permitted by the tools is always correct. For a general theory of action abstraction which includes inter-action and sequential abstraction see [Doan, 1995].

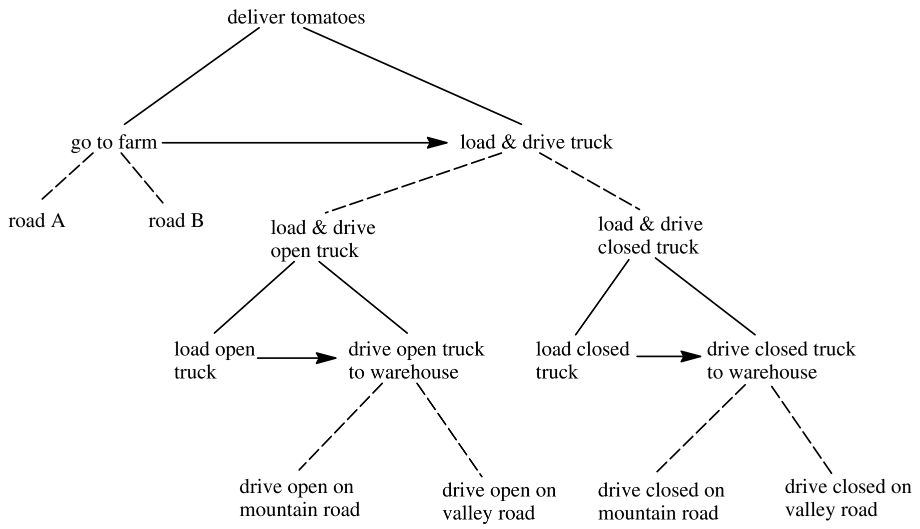

Figure 4: Abstraction/decomposition network. Abstraction relations are shown with dashed lines and decomposition relations are shown with solid lines.

3.2 The DRIPS Planner

A planning problem is described in terms of probability interval constraints on the values of attributes in the initial state, 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 decomposable 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 $\{\mathsf{P}_1, \mathsf{P}_2, ..., \mathsf{P}_n\}$.
   - Compute the expected utility of all newly created plans.
   - Remove P from plans and add $\{\mathsf{P}_1, \mathsf{P}_2, ..., \mathsf{P}_n\}$.
   - 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. The set of instances of this set of abstract plans constitutes the current set of candidate plans. So the algorithm can be stopped at any time to yield the current set of candidate plans. The refinement process guarantees that the lowest EU of any plan in this set monotically increases as a function of computation time. Furthermore, it is triviall to show that for some problems, this lower bound EU strictly increases during some phase of deliberation. So DRIPS is an optimal anytime decision-theoretic planner.

From the set of candidate abstract plans, 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.

4 Search Control Techniques

The extent to which DRIPS is able to eliminate suboptimal plans depends in large part on effectively controlling the plan refinement process. The algorithm above contains two nondeterministic 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 [Haddawy *et al.*, 1995] has shown that if we wish to run the planner to completion, i.e. to find the optimal plan, the most efficient 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 *et al.*, 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.

5 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. To simplify the presentation and allow us to concentrate on the central aspects of the approach, throughout this discussion we will assume *(i)* that utility is a function of only the final state of the world after execution of a plan and is additive, with each sub-utility function defined over one domain attribute; *(ii)* that action effects are sets of attribute value specifications consisting only of incrementing or decrementing each value by a constant (e.g. fuel = fuel - 5); and *(iii)* that actions are not conditional.

We will assume that the user provides a specification of the plan space in the form of a BNF grammar. The BNF grammar forms a partial specification of the abstraction/decomposition network such that at each point in the network where several alternative actions exist, a set of alternative is specified rather than an abstraction hierarchy. For example, we could specify the plan space for a version of the tomato delivery problem in which we have four roads to the farm with the following grammar.

```
deliver_tomatoes = (go_to_farm load&drive)
go_to_farm = (roadA | roadB | roadC | roadD)
load&drive=((load_open_truck drive_open_truck) |
           (load_closed_truck drive_closed_truck))
drive_open_truck = (drive_open_on_mountain |
                    drive_open_on_valley)
drive_closed_truck = (drive_closed_on_mountain |
                      drive_closed_on_valley)
```

Our task is then to find useful abstraction hierarchies for the sets of alternative actions, e.g., for the set of go_to_farm alternatives. In order to do this, we need a method to estimate the information loss due to abstraction. Abstracting actions produces probability and attribute ranges and the wider these ranges, the more information we lose. An abstraction is good if the expected utility interval computed using the abstract action is not much wider than that of each of its member actions. Since computing expected utility intervals requires projecting entire plans on particular initial state distributions, there is no practical way in which we can precisely determine the effect on expected utility of abstracting two or more actions. Rather, we will focus on estimating the effect. To do this, we need some way of determining how an attribute value and changes in an attribute value will impact utility.

Since actions are not conditional, each effect assignment is only a function of the previous attribute value and only effect attributes that appear in the utility function will have an impact on utility. If each sub-utility function is linear, we can estimate the loss due to abstraction by picking some value for each attribute and doing the analysis around that set of values. Since action effects only increment or decrement attribute values by constant amounts, and the utility function is linear, the estimate will not depend on the reference value chosen. If each sub-utility function is not linear, we can estimate the maximum loss due to abstraction by choosing for each attribute that value for which changes affect utility most. Suppose we wish to estimate the loss due to abstraction resulting from abstracting two actions a^1 and a^2, which may be either concrete or abstract actions. Assume that the sub-utility functions are linear and choose some reference value for each attribute. Now project each action on the reference values and use the utility function to compute a pseudo expected utility interval (PEU) for each action. Create an inter-action abstraction a^* of the two actions and project a^* on the reference values to compute a psuedo expected utility interval for it. We now estimate the loss due to abstraction by determining the degree to which we lose information about the lower and upper PEU bounds on the abstracted actions, using the following formula:

$$
\begin{aligned}
&LDA(\mathsf{a}^1, \mathsf{a}^2) = \\
&\quad | \, lb(PEU(\mathsf{a}^1) - lb(PEU(\mathsf{a}^*)) \, | + \\
&\quad | \, ub(PEU(\mathsf{a}^1) - ub(PEU(\mathsf{a}^*)) \, | + \\
&\quad | \, lb(PEU(\mathsf{a}^2) - lb(PEU(\mathsf{a}^*)) \, | + \\
&\quad | \, ub(PEU(\mathsf{a}^2) - ub(PEU(\mathsf{a}^*)) \, |
\end{aligned}
$$

We can now use this loss estimate to determine a good abstraction hierarchy for a set of alternative actions, e.g. the set {roadA, roadB, roadC, roadD} from the delivery example. For simplicity, assume that we wish to create an abstraction hierarchy that groups actions two at a time. To create an abstraction hierarchy for a set of alternative actions A we proceed as follows.

1. For every pair of actions, determine the loss due to abstraction for every possible pairings of their branches. Choose as the best abstraction of those two actions the one with the smallest loss.
2. While there is more than one action in A do
   - Choose to abstract that pair of actions in A with the smallest loss value. Remove the abstracted actions from the set of actions to be abstracted and add the abstraction to the set.
   - Determine the loss value for the best abstraction between the new abstract action and every other action in the set.

We now have a binary abstraction hierarchy for the set A. The measure indicating the loss due to abstraction can be stored with each abstract action and used as a priority value to control search as discussed in the previous section. 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 *et al.*, 1995] have shown that if the changes in the utility function are not drastic, e.g., a change leaves the relative importance of the attributes unchanged, then the hierarchy will still be effective. If we expect to encounter problems with very different utility functions within a domain, we can store multiple abstraction hierarchies and index them by characteristics of the utility functions.

6 Related Work

Our proposed approach to using abstraction 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.

7 Current and Future Research

We presented a procedure to generate abstraction hierarchies for highly simplified domains. In order to avoid clouding the presentation in myriad details, we made restrictive assumptions concerning the form of the action representation and the utility function. Admitting conditional actions greatly complicates the process of abstraction generation for two reasons. First, the impact on utility of a change in the value of an attribute depends on what actions succeed it in a plan. An attribute that appears in the effect of one action may appear in the condition of a later action. Second, the loss due to abstraction depends on the conditions on the branches that are being grouped. When actions are abstracted, the conditions on the resulting abstract action are not required to be mutually exclusive. Grouping branches so that the resulting abstract action has branches with conditions that greatly overlap can produce wide EU ranges. These two complications make estimating the loss due to abstraction significantly more difficult and impair the accuracy of the estimate. We are currently investigating various schemes to handle conditional actions and more complex utility functions.

We are currently in the process of interfacing the DRIPS planner to the TRUCKWORLD simulator [Hanks *et al.*, 1993]. This involves coordinating the planning and execution modules, updating the set of candidate plans once a next action is committed to, and updating the planner's world model to incorporate information from sensing actions. 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 and Dearden, 1994] C. Boutilier and R. Dearden. Using abstractions for decision-theoretic planning with time constraints. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 1016–1022, Seattle, July 1994.

[Dean and Boddy, 1988] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 49–54, Saint Paul, MN, August 1988.

[Dean *et al.*, 1993] T. Dean, L. Pack Kaelbling, J. Kirman, and A. Nicholson. Planning with deadlines in stochastic domains. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 574–579, July 1993.

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

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

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

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

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

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

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

[Hanks *et al.*, 1993] S. Hanks, M.E. Pollack, and P.R. Cohen. Benchmarks, test beds, controlled experimentation, and the design of agent architectures. *AI Magizine*, 14(4):17–42, Winter 1993.

[Hanks, 1990] S. Hanks. *Projecting Plans for Uncertain Worlds*. PhD thesis, Yale University, January 1990.

[Horvitz, 1988] E.J. Horvitz. Reasoning under varying and uncertain resource constraints. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 111–116, Saint Paul, MN, August 1988.

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

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