Two Techniques for Tractable Decision-Theoretic Planning*

Gary H. Ogasawara† **Stuart J. Russell**
Computer Science Division
University of California
Berkeley CA 94720
garyo@cs.berkeley.edu russell@cs.berkeley.edu

Abstract

Two techniques to provide inferential tractability for decision-theoretic planning are discussed. *Knowledge compilation* is a speedup technique that pre-computes intermediate quantities of a domain model to yield a more efficiently executable model. The basic options for compiling decision-theoretic models are defined, leading to the most compiled model in the form of parameterized, condition-action decision rules. We describe a general compilation algorithm that can be applied to dynamic influence diagrams for sequential decision-making. *Decision-theoretic metareasoning* is a control technique that regulates the online replanning. This algorithm focuses on the most promising areas to replan and considers the time cost of replanning.

1 Introduction

Applying decision-theoretic reasoning with action sequences rather than single actions can be used to find an optimal plan. However, the standard method of computing the expected utility of each action sequence and then choosing the sequence of maximum utility as the optimal plan is generally intractable. This paper describes two techniques that we have been investigating to ensure inferential tractability. The first technique, knowledge compilation, is done at the pre-processing stage to reduce the complexity of the model; the second technique, decision-theoretic metareasoning, exerts control upon the planning and execution during run-time.

Knowledge compilation pre-computes intermediate quantities of a domain model to yield a more efficiently executable model (e.g., [2, 10, 7]). We describe a knowledge compilation algorithm that generates a compiled model that is parameterized by prior probabilities of the uncompiled model. Because it is parameterized, the same compiled model can be used repeatedly in sequential decision-making.

Decision-theoretic metareasoning is used to control the planning process. At each decision point, a default plan is available. Metareasoning computes an estimate on the expected value of replanning. If the value of replanning is positive, the replanning algorithm is invoked; otherwise, the next step of the default plan is executed. Because a utility measure of the time cost of replanning is included, timely responses can be ensured.

*This research was supported by NSF grants IRI-8903146, INT-9207213, and CDA-8722788, and the Lockheed AI Center.

†Also affiliated with the Lockheed AI Center, Palo Alto, CA.

2 Knowledge Compilation

The terms will be clarified below, but in brief, the knowledge compilation problem is shown in Table 1.

Given: an uncompiled model with the following properties:
• It is represented by a *dynamic influence diagram* that allows temporal projection from time 0 to time k.
• *Prior probabilities* of time 0 are parameterized for generality.
• *Evidence variables* are specified.
• The *utility node* of the dynamic influence diagram is specified.
Determine: a decision policy with the following properties:
• *Completeness:* There is a a set of IF-THEN, *"condition-action" rules* where a rule applies to every possible world state.
• *Behavioral equivalence:* The recommended action is the same as the best action that would be recommended by the uncompiled model.
• *Parameterized conditions:* The conditions are a conjunction of evidence values and/or probability-inequality conditions on the prior probabilities of the uncompiled model.

Table 1: The knowledge compilation problem

2.1 Input: Uncompiled Model

Dynamic influence diagrams We adopt the *influence diagram* as the basic representational tool. An influence diagram [6] (e.g., Figure 1) is a graphical knowledge representation that accounts for the uncertainty of states and provides inference algorithms to update beliefs given new evidence. An influence diagram represents actions as decision nodes, outcome states (including goals) as probabilistic state nodes, and preferences using value nodes. Depending on the node types which they connect, the arcs in the influence diagram represent probabilistic, informational, or value dependence.

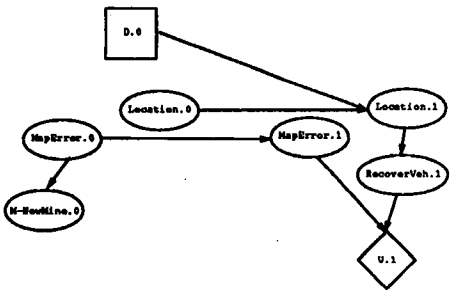


Figure 1: A sample influence diagram.

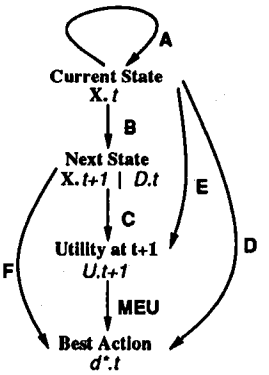
In our applications, several influence diagrams representing a single time slice have been chained together to form a *dynamic influence diagram* (DID). The DID forms a rolling-horizon planning window that modifies itself as time passes. Figure 1 is a DID of 2 time slices where the suffix of 0 or 1 specifies the node's time slice. Assuming the Markov property of successive states, as time passes, the window is expanded by adding a new time slice and the oldest time slice is integrated out to preserve the same amount of lookahead. In the DIDs we have used, the earliest time slice has been used to represent the current time $t0$, and a constant k amount of temporal projection has been implemented by having a fixed number $k+1$ of time slices. On each decision cycle, the observations are entered as evidence into the evidence nodes of the time slice $t0$. The overall utility of the current plan (i.e., the current settings of the decision nodes) is then measured by examining the value node $U.k$. For this reason, though other nodes and links are eliminated during the compilation, the current time's evidence nodes and the top-level value node cannot be removed.

Prior probabilities Prior probabilities are probabilities that are not conditioned on other variables. For example, from Figure 1, $Location.0$ is represented by the prior probability $p(Location.0)$, but $MapError.1$ is a conditioned variable since it is represented by the probability function $p(MapError.1|MapError.0)$. Prior probabilities are parameterized. For example, if $Location.0$ has two possible values, $home, \neg home$, the probability is set to $p(Location.0 = home) = p_l$ and $p(Location.0 = \neg home) = 1 - p_l$ where p_l is a user-controlled parameter. With a set of such parameters for the initial state of the model, the compiled model can be reused for different settings. For one domain, one might set $p_l = 0.9$ while for another mission the setting would be $p_l = 0.99$.

Evidence variables and the Utility node On each decision cycle, new evidence from sensors is entered into the DID by setting the corresponding nodes to a specific value. For example, in a mine-mapping mission, on each cycle, the DID receives information on whether a new mine was detected, true or false. Based on this evidence, the DID is used to compute the action with the highest expected utility. The utility is measured at a specific value node (e.g., $U.1$ in Figure 1).

2.2 Output: Decision Policy

The antecedent conditions of the generated rules are algebraic expressions, parameterized by the variables representing the prior probabilities. Shown below is the general



A: Information about the current state.
B: Effects of actions.
C: Utilities of projected state.
D: Best action choice from current state.
E: Utilities of (current state, action) pairs.
F: Goal
MEU: Maximum expected utility principle.

Figure 2: Knowledge forms for multiple execution architectures.

form of the rules specifying the optimal decision $d^* \in D$ when there are evidence variables $evid_j$ $(1 \leq j \leq n)$ with values $value_{jl}$ $(1 \leq l \leq v)$ for evidence variable $evid_j$:

$$
\begin{aligned}
ruleset &\leftarrow rule_1, \ldots, rule_{v^n} \\
rule_i &\leftarrow \textbf{IF } conds \textbf{ THEN } d^* \\
conds &\leftarrow \bigwedge_{1 \leq j \leq n} cond_j \\
cond_j &\leftarrow (evid_j = value_{jv}) \mid (p \; rel \; func(P)) \mid (p \; rel \; c \; p) \\
rel &\leftarrow \leq \mid = \mid \geq
\end{aligned}
$$

P is the set of parameters p representing prior probability values.
$func(P)$ is an algebraic function on P.

An example rule would look like

IF $(MNM = F) \wedge (p_l \geq 0.9 p_m)$ **THEN** $WSEARCH$

The following sections will show how such rules are generated from the uncompiled model.

2.3 Defining Knowledge Compilation

Knowledge compilation can be defined using the *multiple execution architectures* framework introduced by Russell [14]. Briefly, the idea is that an agent can be defined as a function $f : \mathbf{P}^* \rightarrow \mathbf{D}$ that receives percepts P as input from the environment and outputs a decision $d^* \in D$. From Figure 2 it can be seen that there are multiple inference paths from P to d^* using different knowledge types (A, B, C, D, E, F, and MEU, the maximum expected utility principle). The four different paths from P to d^* describe different implementations of f, the multiple execution architectures. Each execution architecture operates on a different set of knowledge types to make its decision choice:

- Decision-Theoretic EA (path *A B C MEU*)
- Goal-Based EA (path *A B F*)
- Action-Utility EA (path *A E MEU*)
- Condition-Action EA (path *A D*)

The various compilation routes can be identified by enumerating all possible combinations of arcs in Figure 2 that are

HOMOGENEOUS	HETEROGENEOUS	
$A+A \rightarrow A$		
$A+B \rightarrow B$	$B+F \rightarrow D$	$C+\text{MEU} \rightarrow F$
$A+D \rightarrow D$	$B+C \rightarrow E$	$E+\text{MEU} \rightarrow D$
$A+E \rightarrow E$		

Table 2: Listing of all compilation routes. The left column shows the 4 types of homogeneous compilation. The middle column shows heterogeneous compilations using projection. The right column shows heterogeneous compilations using utility computation.

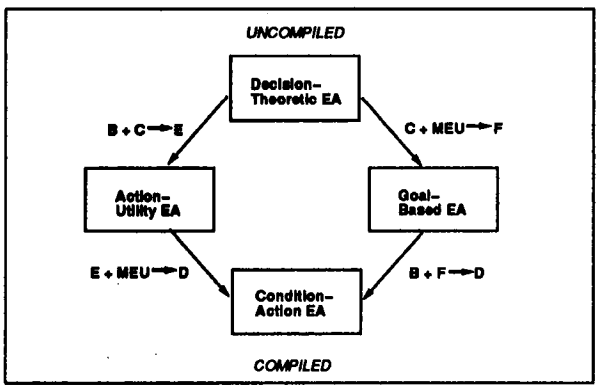


Figure 3: Compilations to convert execution architectures.

adjacent, head to tail. Table 2 shows the eight possible compilation routes where two pieces of some type of knowledge can be converted into a single piece of knowledge. For example, $B+F \rightarrow D$ denotes the combination of type B knowledge and type F knowledge to yield type D knowledge.

The eight types of knowledge compilation can be used to convert an execution architecture from one type to another (Condition-Action, Goal-Based, Action-Utility, or Decision-Theoretic). The most "uncompiled" EA is the Decision-Theoretic EA which can be converted via compilation transformations to any of the other three EAs. The Condition-Action EA is the "most compiled" EA since no compilation transformation can convert it to another EA (Figure 3).

2.4 Implementing Compilation

With the decision-theoretic representation of dynamic influence diagrams, there are two classes of operations to implement compilation:

1. **Integrating out a chance variable.**
   $A+A \rightarrow A, A+B \rightarrow B, A+D \rightarrow D,$
   $A+E \rightarrow E, B+C \rightarrow E, B+F \rightarrow D$

   For example from Figure 1, the compilation $A+B \rightarrow B$ is done by averaging out a probabilistic variable, in this case, $Location.0$:

   $$p(Loc.1|D.0) = \sum_{Loc.0} p(Loc.1|D.0, Loc.0)p(Loc.0) \quad (1)$$

   The result is a reduced model with the probability distribution of $Location.0$ incorporated into the new distribution $p(Location.1|D.0)$.

2. **Pre-computing expected utilities to fix policy.**
   $C+\text{MEU} \rightarrow F, E+\text{MEU} \rightarrow D$

   In the compilation $C+\text{MEU} \rightarrow F$, the value function of type C, v_C, is converted into a value function of type F, v_F by using the Maximum Expected Utility principle.

   $$v_C(U.t1|\mathbf{X}.t1)+\text{MEU} \rightarrow v_F(U.t1|\mathbf{X}.t1)$$

   This is done by assigning $u^\top$ to the outcome state configuration(s) of $\mathbf{X}.t1$ that maximize v_C. $u^\perp$ is assigned to all other state configurations. The Goal-Based execution architecture expects only output states with utilities $u^\top$ or $u^\perp$. Those output states with utility $u^\top$ are "goals" in that a decision sequence leading to a goal is optimal.

2.5 A Compilation Algorithm

This section describes a general knowledge compilation algorithm for the problem as stated in Table 1.

The compilation algorithm is in three steps:

1. *Represent the prior probabilities as parameters.*
   By using parameters for the probabilities (e.g., for a two-valued variable, p and $1-p$ can be used), the compiled network can be re-used many times by instantiating the prior probability variables to different values for different domains/missions.

2. *Propagate the variables through influence diagram reductions.*
   By applying Bayes Rule and marginalization, nodes of the influence diagram can be removed by integrating their influences into the rest of the network.

   - Remove unneeded barren nodes.
     Barren nodes are nodes that do not have any children. Barren nodes can be removed from an influence diagram without affecting the distributions of any of the other nodes, and therefore no changes to existing probability and utility functions need to be made. The barren node removal can be done recursively so that if a node's only children are barren, it, too, can be removed. The evidence nodes, the nodes of time 0 which are parameterized, and the value node of interest cannot be removed because these nodes are used to enter observations and to examine the utility of decisions.
   - Remove nodes of each time slice in order.
     The extraneous nodes of each time slice are removed, starting with Time Slice 0 and progressing to Time Slice k. The node removals are done by applying Bayes Rule and probability variable marginalization.

3. *Generate parameterized, if-then, condition-action rules.*
   IF *condition*, THEN *action* rules can be generated from the Type D knowledge of the Condition-Action EA. This procedure is in two steps:

   - Convert utilities to fixed decision policy.
     Once all extraneous chance variables are removed, then the utilities (which are now Type E knowledge) can be converted to a fixed decision policy using the $E+\text{MEU} \rightarrow D$ compilation.
   - Convert policy to if-then rules.
     The fixed policy can be converted to IF-THEN rules where the condition of the IF statement is one of the possible evidence/percept vectors and a function of

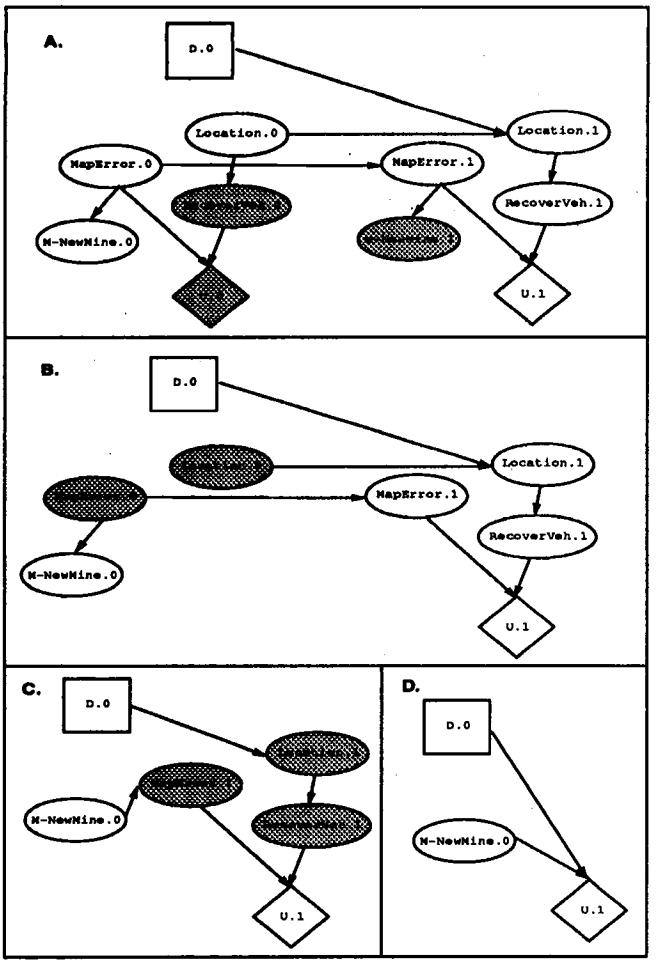


Figure 4: DID undergoing knowledge compilation.

the prior probability variables, and the consequent is a decision recommendation.

2.6 Example

Figure 4 displays four views of a DID of two time slices in different stages of the knowledge compilation process. $MNewMine.0$ is an evidence node that receives sensor information at every time slice. The value node of interest is $U.1$. $Location.0$ and $MapError.0$ are prior probabilities; they are modeled as binary-valued nodes and parameterized with p_l and p_m:

$$p(Loc.0 = home) = p_l \qquad p(Loc.0 = \neg home) = 1 - p_l$$
$$p(MapErr.0 = T) = p_m \qquad p(MapErr.0 = F) = 1 - p_m$$

In (A), the barren nodes, $U.0$, $RecoverVeh.0$, $MNewMine.1$, are identified and removed directly without any changes to the probability or utility tables of other nodes. In (B), the nodes of time 0 to be removed are highlighted. $Location.0$ can be simply removed via the reduction of Equation 1. The parameter p_l is carried through the multiplications and summation. The removal of $MapError.0$ is more complicated, requiring the application of Bayes Rule to reverse the arc between $MapError.0$ and $MNewMine.0$.

In (C), the nodes of time 1 are removed in a similar manner to the previous step. Figure 4D. shows the final graphical form after the $B + C \rightarrow E$ compilation is applied to convert the Decision-Theoretic EA into an Action-Utility EA. Type E knowledge gives utilities of state-action pairs as shown in Table 3. In this case, the state corresponds to the evidence vector $MNewMine.0$, and the possible decisions of $D.0$ are $NOOP, WS, BS, GO$. The utilities are represented as functions $f_i(p_l, p_m)$ of the prior probabilities.

$MNewMine.0$	$NOOP$	WS	BS	GO
F	f_1	f_2	f_3	f_4
T	f_5	f_6	f_7	f_8

Table 3: Type E knowledge: $f_i(p_l, p_m)$

The decision policy is to select $max(f_1, f_2, f_3, f_4)$ if $MNewMine = F$ and $max(f_5, f_6, f_7, f_8)$ if $MNewMine = T$.

Finally, to create the Condition-Action EA, the Type E utilities are converted to best decision choices directly using the $E + \text{MEU} \rightarrow D$ compilation. For example,

IF$(MNM = F) \wedge (p_l \geq 0.9 p_m)$ **THEN** $WSEARCH$
IF$(MNM = F) \wedge (p_l < 0.9 p_m)$ **THEN** $BSEARCH$
$\vdots$

2.7 Discussion

Previous work In a decision-theoretic framework, Heckerman *et al.* [3] compare two strategies: "Compute" and "Compile". Whereas "Compute" uses all of the evidence to make a decision, "Compile" pre-computes and stores the effect of some subset of the evidence variables. The "Compile" strategy can be implemented as a "situation-action tree" where a path from the root to a leaf is the sequence of evidence examined, branching on different evidence values, and terminating at a leaf node that indicates the best decision choice. Lehner and Sadigh [8] follow up on the situation-action tree method, providing procedures for generating and analyzing them.

The generated IF-THEN rules of the compilation algorithm described in this paper are similar to the situation-action tree in that both the condition-action rules and the situation-action tree provide a decision recommendation based on current evidence. The major difference is that the IF-THEN rules are parameterized by prior probabilities of the model, not only the evidence. In addition, rather than having only Boolean conditions on evidence, the IF-THEN rules also admit probability-inequality conditions on the prior probabilities of the model. These generalizations significantly broaden the reapplicability of the compiled model. Another difference is that the situation-action tree heuristically orders the sequential examination of evidence, potentially identifying the best decision before examining all evidence. Hopefully, the condition-action rules can also be processed to yield an efficient implementation (e.g., a situation-action tree or Rete network).

D'ambrosio [1] and Shachter *et al.* [16] have done work on a goal-directed inference system that performs only those calculations required by the query, rather than being data-directed and updating the value of all possible queries upon receipt of new evidence. Intermediate results of computation are cached for possible future use.

Approximate compilation It is desirable to maintain *behavioral equivalence* as knowledge compilation is done. Two systems that are behaviorally equivalent will each select the same decisions given the same percept history. However,

knowledge compilation will still be useful if the system can execute much faster and result in only a small reduction in decision quality. Depending on the utility model of the domain, an approximate, compiled model may be preferable. For example, if there is a high degree of time pressure, the compiled model may be able to compensate in speed of execution for any decision errors due to approximate compilation. The obvious question is whether an optimal tradeoff point between the "accuracy" of the compilation and the time benefit of using the compilation can be determined. For example, we can make approximations so that all probabilities ϵ-close to 0 or 1 are set to 0 or 1, respectively, and save the compiled probabilities in a compressed form.

Controlling compilation Because the knowledge compilation procedure maintains behavioral equivalence between the uncompiled and compiled models, there is no cost with respect to loss of model accuracy. However, two types of costs that do apply are (1) the computational resources required to carry out the compilation and (2) the cost of maintaining multiple compiled models (i.e., compiled and uncompiled versions of the same domain).

The first cost of compilation can be ignored if the knowledge compilation is done off-line before the models are used for decision-making. If compilation is done on-line, it can be put under metalevel control in much the same way that the marginal utility of replanning is analyzed in the second part of this paper. The efficacy of such metalevel control is directly related to the speed of the metalevel analysis and the availability of knowledge about the time-parameterized effects of on-line compilation, specifically, probability distributions on the relevant variables used for decision-making.

There is a cost of maintaining multiple compiled models when new evidence (in the form of assertions that a variable has been observed to take on a specific value) occurs for a node that has been "compiled out". If the original uncompiled network is saved as well as different levels of compiled networks, then the cost is only the memory cost of maintaining multiple models.

3 Decision-theoretic Metareasoning

3.1 Overview

The objective of planning in a decision-theoretic framework is to select the action *sequence* that has the highest expected utility, and then to execute the first action of that sequence. To lend tractability to the computation, we don't compute the utility of every action sequence at every decision point, but rather compare the effect of continuing with the current plan versus doing some sort of replanning. Replanning is invoked if and only if the estimated expected utility of replanning is greater than the estimated expected utility of executing the plan. The value of replanning may be positive, for example, if there is new evidence since the plan was constructed or if the plan can be easily improved. On each decision cycle, a best action must be selected, and it is either to execute the next step of the default plan or to do replanning.

The planning window is a dynamic influence diagram initialized with a nominal plan that specifies a default decision for each $D.i$ in the planning window. The expected utility if the next plan step were executed is the current utility of the top-level utility node. This is because the nominal plan steps

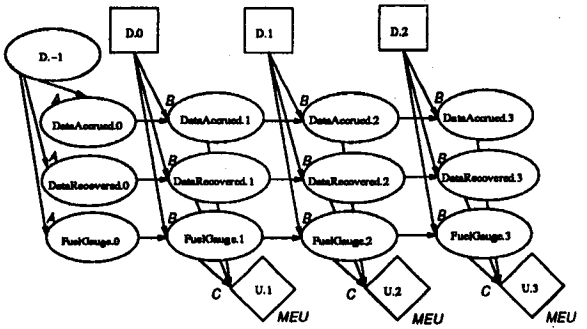


Figure 5: A Decision-Theoretic EA influence diagram with a 3-step planning window.

are "programmed" in by initializing the prior probabilities of the corresponding decision nodes to near 1 for the nominal plan's actions.

As the possible computational actions, replanning decisions affecting different portions of the utility model in the influence diagram are considered. For example, in figure 5, the direct influences on the utility node $U.3$ are the $DataAccrued.3$, $DataRecovered.3$, and $FuelGauge.3$ nodes. The plan repair focuses on the attribute nodes that will yield the maximum net utility gain, that is, the nodes with the maximum value of replanning.

Each node is the root of a subtree of other nodes that influence it. In general, the nodes of an influence diagram can form an directed acyclic graph rather than just a subtree. There are various methods to convert an DAG network into a tree network (e.g., cutset conditioning [13])

Suppose we determine that replanning the $FuelGauge.3$ subtree in the influence diagram has the highest estimated value of replanning. Recursively, the value of replanning the subtrees of $FuelGauge.3$ are computed, and continuing in this manner the influence diagram is examined until (1) a decision node is reached and modified to the action value that maximizes utility *or* (2) all examined subtrees have a negative estimated value of replanning.

3.2 Value of replanning

To show how the value of replanning a subtree is determined, let X be the subtree being considered where "X" is the name of the subtree's root. Assuming that we can separate the time-dependent utility, $TC(X)$, from the "intrinsic" utility, $EU_I(X)$, the estimated value of information (VI) gained from replanning subtree X is

$$\hat{VI}(replan(X)) = \hat{EU}_I(replan(X)) - \hat{TC}(replan(X))$$

where $replan(X)$ denotes the action of replanning subtree X and $\hat{\cdot}$ indicates an estimated quantity.

Subtrees like X are evaluated and the corresponding parent nodes are inserted into a list of nodes ordered by the value of replanning the subtree rooted at that node. The value of replanning is a type of "value of information" [5, 4, 15, 9] used in decision-theoretic metareasoning. If the node is controllable, its value is changed, otherwise, the node is recursively expanded to further narrow the area subject to replanning. Using the ordered list allows "best-first" search where the best subgraph to examine can jump around in the influence diagram.

When the time cost, $\hat{TC}$, is greater than the expected benefit, $\hat{EU}_I$, the value of replanning is negative, and it should not be done. By varying the TC function, we can get different degrees of the time-criticality of the mission. Currently, we define TC to be proportional to the number of cpu seconds used by the replanner: $TC = \tau * \hat{s}$ where τ is a time-criticality parameter and $\hat{s}$ is the estimated number of cpu seconds required to replan the subtree. To estimate s, we maintain statistics over many runs and use Bayesian prediction to incorporate prior knowledge and incrementally update the distribution of s. Note that if TC is 0, there is no reason not to continue to compute without acting.

Utility of replanning a subtree $\hat{EU}_I(replan(X))$, of Equation 2, is the estimated expected utility of the state resulting from replanning node X independent of time. It is computed by taking the utility difference between the subtree after replanning, X', and the current subtree, X. To compute the utility for X', we must know the probability that different X's will occur as a result of replanning. For a decision node, because it is completely controllable, the probability is 1 that any particular value can be achieved and simply the decision value that maximizes utility is selected. But for chance nodes, we need to estimate the probability and update it based on replanning experiences.

As an example, suppose X is the subtree rooted at $FuelGauge.3$. The utility of the current probability distribution of $FuelGauge.3$ node can be computed directly. But to compute the utility of the same node $FuelGauge.3'$ after replanning decisions that might change its probability distribution, we must estimate how likely it is that the replanning will alter actions in the plan to achieve a new distribution for $FuelGauge.3'$.

In the example, if the initial probability distribution for $FuelGauge.3$ has a high probability for the value *empty*, the utility of that subtree will be relatively low according to our utility model (an empty fuel tank is bad). But if we know that there is a high probability that replanning (perhaps by omitting actions to lower fuel consumption) will change the probability distribution to more heavily weight $half$ or $full$, the utility of the $FuelGauge.3'$ subtree will be higher. Thus $\hat{EU}_I(replan(X))$ would be positive.

To make $\hat{EU}_I(replan(X))$ explicit, let ϕ be the probability distribution of the root of the subtree X (i.e., $\phi = p(X)$), and let ϕ' be the probability distribution of X *after* replanning (i.e., $\phi = p(X')$). Then, the *expected* utility of $replan(X)$ is the average utility over all possible values of ϕ' (i.e., all probability distributions of the node):

$$EU_I(replan(X)) = \int p(\phi')u(replan(X))d\phi' \quad (2)$$

where the *utility* of replanning a node is the difference between the new state resulting from replanning and the old state:

$$u(replan(X)) = u(\phi') - u(\phi) \quad (3)$$

$$EU_I(replan(X)) = \int p(\phi')(u(\phi') - u(\phi))d\phi' \quad (4)$$

To find where to replan in the influence diagram, we start at the top-level U node and work backwards through the arcs to the base-level actions. For example in figure 5, $U.3$ is the child of its three parents, $DataAccrued.3$, $DataRecovered.3$, $FuelGauge.3$. Informally, the replanner wants to set a parent node's value so that its children have a value with maximum expected utility. For example, the replanner wants to set $FuelGauge.3$ to the value such that $U.3$ is at its optimal value, 1. Once the optimal distribution for $FuelGauge.3$ is determined, recursively, the same procedure is invoked so that the propagation of evidence continues to the parents of $FuelGauge.3$.

Let Y be the child node of node X. With the assumption that the network is a tree, each node has at most one child. The utility of a particular distribution, $u(\phi)$ for X, is the expected utility of the probability distribution of X's child, $\phi_Y = p(Y)$, given ϕ and the rest of the belief network ξ. Letting ξ be implicit yields Equation 6:

$$u(\phi, \xi) = \int p(\phi_Y|\phi, \xi)u(\phi_Y)d\phi_Y \quad (5)$$

$$u(\phi) = \int p(\phi_Y|\phi)u(\phi_Y)d\phi_Y \quad (6)$$

Since ϕ is the current probability distribution for X, it can be read directly from the current network with no need for computation. As the base case of the recursive inference, the top-level node U's utilities are defined as $u(U = 1) = 1$ and $u(U = 0) = 0$. The other utilities of ϕ are determined from previous inferences as the belief net is examined recursively.

Specifying $u(\phi')$ is more difficult because every possible probability distribution of X must be considered. Here is where we *estimate* $EU_I(replan(X))$ of Equation 4, by ranging only over the possible discrete state values of X:

$$\hat{EU}_I(replan(X)) = \sum_i p(X = x_i)(u(X = x_i) - u(\phi)) \quad (7)$$

where substituting into Equation 6:

$$u(X = x_i) = \sum_j p(Y = y_j|X = x_i)u(Y = y_j) \quad (8)$$

The probability that the probability distribution $X = x_i$ will result from replanning, $p(X = x_i)$, must also be provided. This probability can be viewed as a measure of the *controllability* of the node or whether Howard's "wizard" is successful [6]. For decision nodes which are perfectly controllable, the probability for the value x^* that maximizes utility can be set to 1: $p(X = x^*) = 1$. For other nodes, we must estimate $p(X = x_i)$. In our implementation, we start with a uniform prior distribution and use Bayesian updating based on replanning episodes.

We can show that if the algorithm considers all subtrees with no approximations, the use of the paths from the utility node to the decision node by the above procedure leads to the determination of the optimal policy [11].

3.3 Example

Using the influence diagram of Figure 5, we can compute $\hat{VI}(replan(FuelGauge.3))$. Looking up the $\hat{s}$ for the $FuelGauge.3$ subtree, suppose we find 0.15 and use $\tau = 1$, so $\hat{TC} = 0.15$. Computing the $\hat{EU}$, we start with the given utility $u(U.3 = 1) = 1$ and $u(U.3 = 0) = 0$. Therefore,

$u(FuelGauge.3)$

$= u(U.3 = 1)p(U.3 = 1|FuelG.3, DataAcc.3, DataRec.3) +$

$$
\begin{aligned}
& u(U.3=0)p(U.3=0|FuelG.3, DataAcc.3, DataRec.3) \\
= \; & 1 * p(U.3=1|FuelG.3, DataAcc.3, DataRec.3) + \\
& 0 * p(U.3=0|FuelG.3, DataAcc.3, DataRec.3) \\
= \; & p(U.3=1|FuelG.3, DataAcc.3, DataRec.3) \\
= \; & 0.29
\end{aligned}
$$

Setting $FuelGauge.3$ to each of its states and propagating the change to its child $U.3$, we get new values for $p(U.3 = 1|FuelGauge.3', DataAccrued, DataRecovered)$ and the utility of specific fuel gauge values:

$$
\begin{aligned}
u(FuelGauge.3 = \text{zero}) &= 0.01 \\
u(FuelGauge.3 = \text{low}) &= 0.75 \\
u(FuelGauge.3 = \text{high}) &= 0.78
\end{aligned}
$$

Assuming that we have not replanned this subtree before, the probabilities are at their default uniform values:

$$
\begin{aligned}
p(FuelGauge.3 = \text{zero}) &= 1/3 \\
p(FuelGauge.3 = \text{low}) &= 1/3 \\
p(FuelGauge.3 = \text{high}) &= 1/3.
\end{aligned}
$$

Putting everything together,

$$
\begin{aligned}
& \hat{EU}(replan(FuelGauge.3)) \\
= \; & \tfrac{1}{3}(0.01-0.29) + \tfrac{1}{3}(0.75-0.29) + \tfrac{1}{3}(0.78-0.29) \\
= \; & 0.22 \\
& \hat{VI}(replan(FuelGauge.3)) = 0.22 - 0.15 = 0.07
\end{aligned}
$$

The estimated value of replanning the $FuelGauge.3$ subtree is 0.07. This value is compared to replanning other subtrees (viz., $DataRecovered.3$ and $DataAccrued.3$) and since it is positive, the best subtree is replanned. If the root of the subtree is a decision node, it is set to its optimal value. Otherwise, recursively, the lower levels of the best subtree are examined until the controllable decision nodes to replan are found. Then, the $\hat{s}$ and $p(X = x_i)$ numbers are updated using the new data from the replanning episode.

3.4 Discussion

Complexity The time complexity of the $\hat{VI}$ calculation is low since only one level of evidence propagation in the network from the node of interest to its children is ever done. The $u(\phi)$, $\hat{s}$, and $p(X = x_i)$ values are obtained using constant time table lookups. The calculation of the $u(X = x_i)$ terms involves the single-level propagation. If there are at most n states each for X and its child Y, the total time for a single $\hat{VI}$ calculation is $O(n^2)$. At each time step, each parent of the child must be evaluated to find the maximum $\hat{VI}$, therefore the total time for one time step is $O(kn^2)$ where k is the indegree of the node.

The space requirements are constant in the number of nodes. For the Gaussians $\hat{s}$ and $p(X = x_i)$, the sufficient statistics are the mean and variance which are stored with each node.

4 Conclusion

Knowledge compilation and metalevel control of replanning were the two techniques discussed to aid in achieving real-time performance without sacrificing too much decision quality. For both techniques, assumptions about the structure of the domain are exploited. For example, by knowing which variables are evidence variables, we can compile out all other internal state nodes. As for future work, interleaving compilation with execution is a crucial ability if the domain model changes significantly with time. For the replanning work, we are investigating more flexible methods to do temporal projection (e.g., variable amount of lookahead, and non-Markov domains) and exploring the advantages and disadvantages of different execution architectures for different domains (see also [12]).

References

[1] B. D'Ambrosio. Incremental probabilistic inference. In *Proc. of the 9th Conf. on Uncertainty in AI*, 1993.

[2] R. E. Fikes, P. E. Hart, and N. J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 4, 1972.

[3] D. E. Heckerman, J. S. Breese, and E. J. Horvitz. The compilation of decision models. In *Proc. of the 5th Conf. on Uncertainty in AI*, 1989.

[4] E. J. Horvitz. Reasoning under varying and uncertain resource constraints. In *Proc. of the 7th Natl. Conf. on AI*, 1988.

[5] R. A. Howard. Information value theory. *IEEE Trans. on Systems, Man, and Cybernetics*, 2, 1965.

[6] R. A. Howard. Influence diagrams. In R. A. Howard and J. E. Matheson, editors, *Readings on the Principles and Applications of Decision Analysis*, vol. 2, Strategic Decisions Group, Menlo Park, CA, 1984.

[7] J. E. Laird, P. S. Rosenbloom, and A. Newell. Chunking in soar. *Machine Learning*, 1(1), 1986.

[8] P. E. Lehner and A. Sadigh. Two procedures for compiling influence diagrams. In *Proc. of the 9th Conf. on Uncertainty in AI*, 1993.

[9] J. E. Matheson. Using influence diagrams to value information and control. In R. M. Oliver and J. Q. Smith, eds., *Influence Diagrams, Belief Nets and Decision Analysis*. John Wiley & Sons, Chichester, UK, 1990.

[10] T. M. Mitchell, R. M. Keller, and S. T. Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1:47–80, 1986.

[11] G. H. Ogasawara. *RALPH-MEA: A Real-Time Decision-Theoretic Agent Architecture*. PhD thesis, UC Berkeley, CS Division, 1993. Report UCB/CSD-93-777.

[12] G. H. Ogasawara and S. J. Russell. Planning using multiple execution architectures. In *Proc. of the 13th Intl. Joint Conf. on AI*, 1993.

[13] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA, 1988.

[14] S. J. Russell. Execution architectures and compilation. In *Proc. of the 11th Intl. Joint Conf. on AI*, 1989.

[15] S. J. Russell and E. H. Wefald. Principles of metareasoning. In *Proc. of the 1st Intl. Conf. on Principles of Knowledge Representation and Reasoning*, 1989.

[16] R. D. Shachter, B. D'Ambrosio, and B. A. Del Favero. Symbolic probabilistic inference in belief networks. In *Proc. of the 8th Natl. Conf. on AI*, 1990.