Planning Using Multiple Execution Architectures*

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

Abstract

We discuss two techniques used by the RALPH-MEA agent architecture to facilitate decision making in complex, real-time domains. *Multiple execution architectures* are four implementations of the agent function, a function that receives percepts from the environment as input and outputs an action choice. The four execution architectures are defined by the different knowledge types that each uses. Depending on the domain and agent capabilities, each execution architecture has different speed and correctness properties. *Metalevel control of planning* computes the value of information of planning to compare to the utility of executing the current default plan. Examples are presented from an autonomous, underwater vehicle domain.

1 Introduction

The goal of a decision-making system is to select the best action in the current situation. In a decision-theoretic framework, this is expressed as choosing the optimal action, d^*, given the probability distribution on the outcome states, $s \in S$, and a utility function on outcome states, $u(s)$:

$$d^* = argmax_d \sum_{s \in S} p(s|d)u(s)$$

If an agent can continuously apply this equation (decision theory's maximum expected utility (MEU) principle) to select "best" actions, it can be considered to be generating optimal behavior with respect to its goals. However, in even moderately complex domains, the MEU Principle cannot be implemented directly because of limited resource constraints (e.g., time) and uncertainty about the utility of outcome states. This paper discusses two general-purpose techniques used by the RALPH-MEA[1] agent architecture that alleviate this AI scalability problem: *multiple execution architectures* and *metalevel control of planning*.

An agent can be defined as a function $f : \mathbf{P}^* \rightarrow \mathbf{D}$ where $\mathbf{P}^*$ is the set of percept sequences from the environment and $\mathbf{D}$ is the set of action decisions available to the agent. An execution architecture (EA) [Russell, 1989] is an implementation of the agent function that operates on a specific combination of knowledge types (e.g., goals, probabilities of states, outcomes of actions, etc.). Different execution architectures will have different competences and costs in different situations — for example, reactive condition-action rules are good for shooting down missiles and playing standard chess openings, but decision-theoretic planning may be more useful for selecting missile targets and playing tactically sharp middle games. Employing multiple EAs with the appropriate control to arbitrate the final action choice should allow greater competence to be exhibited than is possible for a single EA alone.

The second tool we use to implement the MEU principle is metalevel control of planning. Planning is the method of projecting the current state into the future to predict possible outcomes. If the domain is simple enough, there is no need for planning since the the MEU principle can be applied with exact probability and utility functions for the action outcomes. However, as the domain becomes more complex, we need to consider subsequent action sequences in order to calculate the utility of the outcomes of our immediate actions. Metalevel control of planning determines when and how much planning should be done. By viewing planning as a computational action, it can be integrated into the same decision cycle as base-level actions (i.e., actions that directly affect the external world). On each cycle, a best action must be selected, and it may be a base-level or computational action. Theoretical and practical issues of decision-theoretic metareasoning have recently been an active research area (e.g., [Russell and Wefald, 1989; Doyle, 1990; Henrion *et al.*, 1991]). Our approach calculates a "value of information" [Howard, 1965] of doing various planning actions, selecting the planning or base-level action that has the highest utility.

First, we will discuss the representation used for multiple execution architectures and planning (section 2). Then we will describe the decision cycle that uses those representations to output action recommendations (section 3). Section 4 describes results from an autonomous, underwater vehicle (AUV) domain, implemented at the Lockheed AI Center.

2 Representation

2.1 Knowledge Types

Each execution architecture (EA) is a different representation of the agent function, $f : \mathbf{P}^* \rightarrow \mathbf{D}$, that outputs decisions. The EA categorization is based on the use of six types of

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

[1]Rational Agents with Limited Performance Hardware–Multiple Execution Architectures

Proceedings of the 13th International Joint Conference on Artificial Intelligence. Chambery, France, 1993.

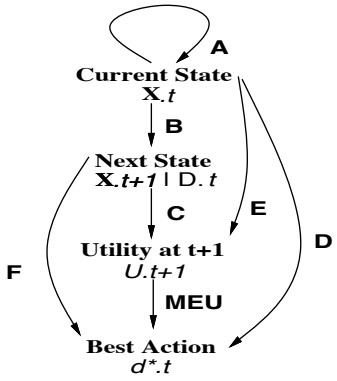

Figure 1: Knowledge forms for multiple execution architectures.

knowledge. The original specification [Russell, 1989] of the knowledge types used a situation calculus representation, but in this paper, we introduce an extended *influence diagram* representation. An influence diagram explicitly accounts for the uncertainty of the 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 among outcome states using value nodes. Depending of the node types they connect, the arcs in the influence diagram represent probabilistic, informational, or value dependence.

Our extended influence diagram (EID) representation differs from an influence diagram by defining the dependencies using the six types of knowledge. These dependencies are more specialized than the general probabilistic, informational, and value dependencies used in an influence diagram, and therefore, specialized inference procedures can be used rather than the standard influence diagram inference techniques.

To describe the current world state, we use a set of n state variable nodes, $\mathbf{X}.t = \{X_1.t, \ldots, X_n.t\}$, a decision node $D.t$, and a utility node $U.t$. The suffix $.t$ is used to indicate time t. For the next time, $t+1$, the nodes are represented as $\mathbf{X}.t+1 = \{X_1.t+1, \ldots, X_n.t+1\}$, a decision node $D.t+1$, and a utility node $U.t+1$. Assuming a Markov property on the states allows specification of the influences on $\mathbf{X}.t+1$ using only $\mathbf{X}.t$ and $D.t$.

As is shown in figure 1, the six knowledge types of the EID representation, denoted by the letters A, B, C, D, E, F, connect the different knowledge components together. These six knowledge types are given decision-theoretic definitions as follows:

A: $p(X_i.t|\mathbf{X}.t)$
Type A rules specify the likelihood of a state node given information about other state nodes for the same time. Rules of this type can be used for the interpretation of percepts. For example, the dependence of the variable `FuelGauge.t` given the percept `BatteryFailure.t` can be represented as a conditional probability function: $p(FuelGauge.t|BatteryFailure.t)$.

B: $p(X_i.t+1|D.t, \mathbf{X}.t)$
Type B rules describe the effects of actions by specifying the influence of an action on the resulting state. In contrast to a type A rule, an action $D.t$ and conditions $\mathbf{X}.t$ at time t influence the node $X_i.t+1$ at time $t+1$. The example $p(DataAccrued.1|D.0, DataAccrued.0)$ states that the node $DataAccrued.1$ of time 1 is directly affected by $DataAccrued.0$ and $D.0$ of time 0.

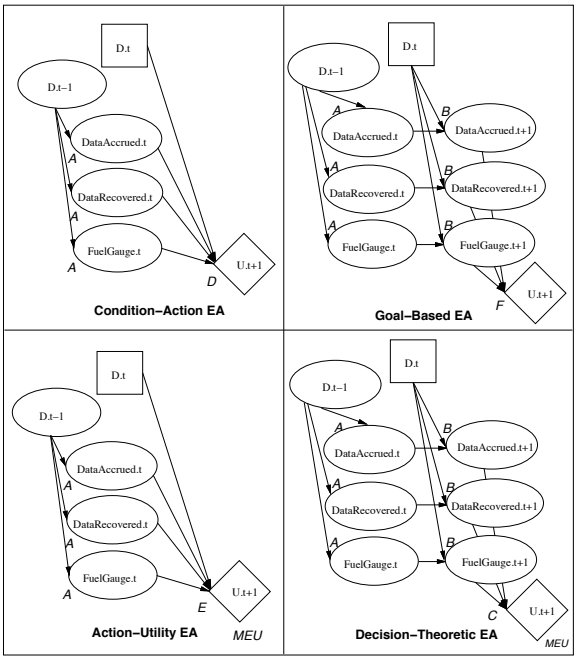

Figure 2: EID repr. for the four execution architectures in the AUV domain. Decision nodes are squares, value nodes are diamonds, and state nodes are ovals. The conditional probability functions or utility functions at the nodes are labeled by their type of knowledge (A, B, C, D, E, F, MEU) they represent. The suffix `t` or `t+1` denotes the node's time.

C: $v(U.t+1|\mathbf{X}.t+1) \in [u^{\perp}, u^{\top}]$
Type C rules are utility functions on the state. In the EID, this is represented using a value function v on the node $U.t+1$ which is bounded by the minimum and maximum possible utilities, $u^{\perp}$ and $u^{\top}$.

D: $v(U.t+1|D.t, \mathbf{X}.t) \in \{u^{\perp}, u^{\top}\}$
Type D rules are akin to standard condition-action rules used in reactive or production rule systems. They state that if the specified conditions hold now, then the specified action is best. The utility $U.t+1$ is conditioned on $\mathbf{X}.t$ and $D.t$ of time t. Type D rules express absolute certainty by using the endpoints of the utility range as their only possible values.

E: $v(U.t+1|D.t, \mathbf{X}.t) \in [u^{\perp}, u^{\top}]$
Type E rules are "action-value" rules that specify the utility of condition-action pairs, for example Q-tables [Watkins, 1989]. In the EID representation, they represent the same qualitative influences as D rules but the utility value is generalized to the range $[0, 1]$.

F: $v(U.t+1|\mathbf{X}.t+1) \in \{u^{\perp}, u^{\top}\}$
The type F rule, which is similar to a classical goal [Newell, 1982], expresses the utilities of states in the future time period, $t+1$. The utility is restricted to $u^{\perp}$ or $u^{\top}$ values. A desirable goal is when $v(U.t+1|\mathbf{X}.t+1) = u^{\top}$.

2.2 Execution Architectures

Exactly four execution architectures can be defined using the four different combinations of knowledge types to make an action decision (figure 2).

CA: Condition-Action EA: (path $A\ D$)
Knowledge of type D provides best action choices directly. In many production systems, uncertainty regarding the preconditions of a rule is not considered. A precondition either is true or false, and the action of any rule whose preconditions are matched is executed. This type of system can be implemented by the Condition-Action EA by allowing no uncertainty about the conditions ($p(X_i.t|\mathbf{X}.t) \in 0, 1$). The Condition-Action EA then would output the first action, d_i, it considers such that $v(U.t{+}1|D.t = d_i, \mathbf{Y}.t) = u^{\top}$ (the maximum possible utility) and the rule's preconditions hold. The alternative Condition-Action EA implementation is to allow uncertainty about the preconditions. However in this case, the expected utilities of each action must be computed and compared to select the best action. Computational savings can still be achieved by taking advantage of the restriction of utility values to the minimum and maximum possible utilities: $\{u^{\perp}, u^{\top}\}$. For example, if $u^{\perp} = 0$, many propagations of values in the EID are immediately pruned.

GB: Goal-Based EA: (path $A\ B\ F$)
Knowledge of types A, B, and F suggests actions that achieve the desired goal condition. Similar to the case with the Condition-Action EA, two possible implementations of the Goal-Based EA can be considered. If the probability values of conditions are restricted to 0 and 1, the first action satisfying the goal utility function is selected. If uncertainty of conditions is allowed, the expected utilities of actions need to be computed, but computational savings results from the restriction of utility values to $\{u^{\perp}, u^{\top}\}$.

AU: Action-Utility EA: (path $A\ E$ *MEU*)
Knowledge of type E for various actions is combined with the MEU principle to select the most valuable one. Inference in the Action-Utility EA uses the standard probabilistic and decision-theoretic reasoning techniques done in influence diagrams: the conditional probabilities of each state node are revised by propagating the effect of evidence through the network, the expected utility of each action is computed, and the action with the maximum expected utility is selected. The Q-tables from Q-learning [Watkins, 1989] are knowledge of type E.

DT: Decision-Theoretic EA: (path $A\ B\ C$ *MEU*)
Knowledge of types A, B, and C is combined to find the best action using the MEU principle. As in the Action-Utility EA, standard decision-theoretic reasoning needs to be used in the Decision-Theoretic EA because of the lack of restrictions on the possible utility values.

In order to succeed in complex environments, an agent will need all four execution architectures, which will come into play at different times. If we consider chess, for example, it seems obvious that action-utility rules have restricted usefulness (perhaps just for describing the value of exchanges, knight forks etc.); condition-action rules constitute the "opening book", some endgame knowledge and perhaps plausible move generators; goal-based reasoning occurs when identifiable goals (such as checkmate, trapping a queen, queening a pawn etc.) become achievable with reasonable likelihood; and the remaining situations are covered by decision-theoretic planning using a utility function on states (material, center control etc.). It would be absurd to try to build a chess program based entirely on condition-action rules or action-utility rules, because such a program would be unimaginably vast. Compact representations of good decision procedures require a variety of knowledge types. It would also take a long time to learn a complete set of condition-action rules for chess, whereas it takes only an hour or two for a human to learn type B rules describing how the pieces move. This provides another motivation for multiple execution architectures: learning is more effective for the explicit knowledge types (A,B,C) but execution is more efficient for the compiled types (D,E,F); thus we learn for one architecture and compile, where reasonable, for another[2].

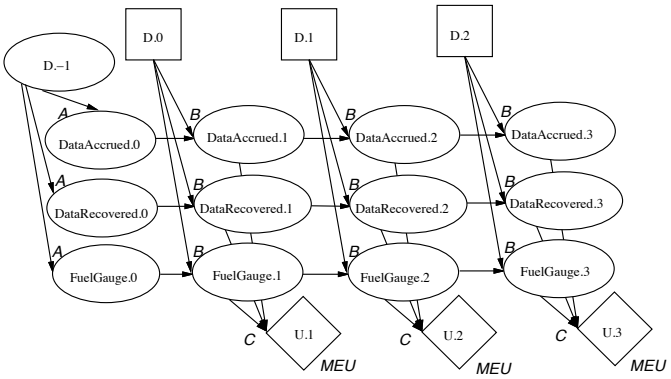

Figure 3: A Decision-Theoretic EA influence diagram with a 3-step planning window.

2.3 Planning windows

For an action *sequence* rather than a single action, many single-decision templates like the ones of Figure 2 can be chained together. Assuming the Markov property of the states, we can reproduce the same decision template repeatedly as the world state is projected forward in time using only local connections. The *planning window* is the influence diagram that is currently being considered. For one-step planning, the length of the window is one time step. For k-step planning, the influence diagram is extended by chaining k decision templates together.

Figure 3 shows a 3-step planning window obtained by chaining together three decision templates of the Decision-Theoretic EA in a simplified version of the autonomous underwater vehicle (AUV) domain. The utility of the AUV's mission is defined in terms of whether it recovers survey data (nodes: `DataAccrued` and `DataRecovered`) and the amount of fuel used (node: `FuelGauge`). The decision node $D.t$ has three possible actions to choose from: `wait`, wait for data to accrue at the sensor; `return`, return to the surface to be recovered; and `pickup`, go to the sensor to pick up the data. The decision node of the previous time, $D.t-1$, becomes evidence for the next decision, $D.t$.

The utility of the action sequence in general can be a function of all the nodes in the planning window, however, a

[2] In [Ogasawara, 1993] we show general conditions under which learning action-utility values (e.g., Q-learning [Watkins, 1989]) works better or worse than learning the utilities of states.

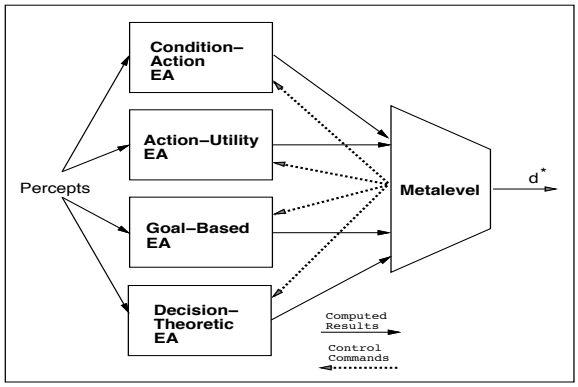

Figure 4: RALPH-MEA system architecture.

common restriction is to make the action sequence utility a function of the utilities at each time slice (e.g., in figure 3, the utility of the $(D.0, D.1, D.2)$ sequence would be a function of the utility nodes $U.1, U.2, U.3$). Another method is to evaluate the action sequence using only the utility of the final time slice (e.g., $U.3$).

It should be noted that planning windows apply only to the Goal-Based EA and Decision-Theoretic EA. The Condition-Action EA and the Action-Utility EA do not contain knowledge describing the effects of actions (Type B knowledge), therefore, there is no way to chain together single decision templates. Also, "planning" in this decision-theoretic framework is somewhat different from the traditional notion of planning for goal achievement. The aim of planning is no longer to achieve goals but to maximize expected utility. As has been often pointed out, the generality of maximizing expected utility rather than achieving goals is necessary to deal with multiple objectives and goal achievement uncertainty (e.g., [Feldman and Sproull, 1977; Wellman and Doyle, 1991]).

3 The Decision Cycle

Given the execution architecture and planning window representation, there is still the issue of how to use the four execution architectures together to make decisions. Our current implementation works as follows. Corresponding to each of the four EAs, there is a separate influence diagram that is executed in parallel to determine an action recommendation for that EA. A separate Metalevel influence diagram receives updates from the four EAs and makes the final decision on the action choice by checking the values of a "best action" node. At this stage of the research, this method has the advantage of modularizing the EAs to allow the design and performance of individual EAs to be isolated[3]. The Metalevel is also separated from the implementation of the EAs and can be developed independently. Figure 4 shows the high-level system architecture view of RALPH-MEA.

The decision making algorithm follows a cycle in which the following computation occurs:

1. The model is updated by advancing the planning window and entering new evidence.
2. Each EA computes its action choice.
3. The Metalevel ID determines when to stop the EA computations and output the current best action.

[3]Later versions will combine the separate Extended Influence Diagrams in order to share nodes.

1. Instantiate last executed action node, $D.t$, as evidence.
2. Add new time slice $t + k + 1$.
3. Integrate out nodes of time t ($\mathbf{X}.t, D.t, U.t$).
4. Set $t = t + 1$.
5. Enter new evidence for $\mathbf{X}.t$ nodes.
6. Propagate evidence through network.

Figure 5: The Updating the Model step.

The following sections examines each of these steps in turn[4].

3.1 Updating the model

As the agent makes decisions, the planning window is updated by modifying the influence diagram. After $D.t$ is executed, its value is instantiated as evidence. For a k-step planning window, a new time slice is added by connecting nodes of time $t + k$ to the new nodes of time $t + k + 1$. Nodes of time t, $\mathbf{X}.t, D.t, U.t$, are then removed from the network by integrating their values into the rest of the network. In general this is done by converting the conditional probabilities $p(X_i.t+1|\mathbf{X}.t)$ into marginal probabilities:

$$p(X_i.t+1) = \int_{\mathbf{X}.t} p(X_i.t+1|\mathbf{X}.t)p(\mathbf{X}.t)$$

The time variable t is then incremented so that the next decision to execute is again $D.t$. Evidence for the new $\mathbf{X}.t$ nodes in each EA influence diagram is entered at this point. For example, if we detect a battery failure, the corresponding node `BatteryFailure.t` is set to `True`. Finally, propagation is done to transmit the effects of the new influence diagram and new evidence to all the nodes.

Similar techniques have become common recently in the uncertainty community. [Kjaerulff, 1992] discusses an implementation of "model reduction" and "model expansion" that moves a belief network forward in time, allowing a continuously existing agent to maintain an updated world model indefinitely[5]. [Kanazawa and Dean, 1989], [Hanks, 1990], and [Nicholson, 1992] also discuss similar temporal projection methods.

3.2 EA Computation

As each EA computes using its separate influence diagram, it modifies its assessment of the utilities of the possible actions. These action utilities are periodically read by the Metalevel ID to update its assessment of the action utilities and to determine when to stop computing and output the current best action. The EAs run in parallel and if the Metalevel decides to execute its current best action before some EA has finished its computation, the EA is interrupted to start the next decision cycle.

[4]Because of space limitations, many important details have been omitted, but can be found in [Ogasawara, 1993].

[5]This assumes that the dependency structure remains fixed (e.g., there is no addition or removal of arcs or nodes)

1. $R = U.t + k$
2. Compute $\hat{VI}(replan(X))$ for each subtree X of R.
3. $X_{max} = argmax_X(\hat{VI}(replan(X))$
4. if ($\hat{VI}(replan(X_{max})) \leq 0$)
   then $d^*.t =$ next plan step; exit.
   else if (X_{max} is a decision node) then change its action value; goto 1.
   else $R = X_{max}$; goto 2.

Figure 6: The EA Computation step.

It is in this EA Computation step that we consider planning actions to change the choice of base-level actions. We assume the planning window is initialized with a nominal plan that specifies a default action for each `D.i` in the planning window. The expected utility if the next plan step were executed is the current utility of the `U.t+k` node since the nominal plan steps are "programmed" in by initializing the prior probabilities of the corresponding decision nodes to near 1 for the actions of the nominal plan. The utility of executing the next plan step is compared to the utility of executing a planning action. The value of planning may be positive, for example, if there is new evidence since the plan was constructed or if the plan can be easily improved.

As our possible planning actions, we consider replanning decisions affecting different portions of the utility model in the influence diagram. For example, in figure 3, the direct influences on the utility node $U.3$ are the $DataAccrued.3$, $DataRecovered.3$, and $FuelGauge.3$ nodes. We seek to work on repairing 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. 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.

Let X be the root of the subtree we are considering replanning. 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.

The estimated utility of replanning a portion of the influence diagram $\hat{EU}_I(X)$ is computed by taking the utility difference between the 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 action value that maximizes utility is selected. But for chance nodes, we estimate the probability and update it based on replanning experiences.

1. Do propagation for the current values to compute $EU(d^*.t)$.
2. Do propagation to compute $EU(d^*.t')$.
3. $EVC(\Delta t) = EU(d^*.t') - EU(d^*.t) - TC(\Delta t)$
4. If ($EVC(\Delta t) \leq 0$)
   then execute $d^*.t$; exit.
   else goto 1.

Figure 7: The Metalevel Computation step.

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 $FuelGauge.3'$. 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. 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 $half - full$, the utility of the $FuelGauge.3'$ subtree will be higher. Thus $\hat{EU}_I(X)$ would be positive.

If the estimated value of replanning, $\hat{VI}(replan(X))$, is positive, the replanning action with the highest value is executed, and the EA computation step is repeated with the new plan by comparing executing the next plan step and executing a planning action. When the value of replanning is non-positive, the EA computation step finishes with the next plan step as its action choice.

3.3 Metalevel computation

The metalevel problem is to take the computed results (in this case, the action utilities) of the four EAs and to output a final action choice. At any time with its current information, the Metalevel ID has a current best action $d^*.t$. As the results from the EAs arrive asynchronously, the Metalevel must decide whether to execute the current best action $d^*.t$ or to wait Δt for possible updates from the EAs. This is the basic metalevel computation algorithm described in [Horvitz, 1988]. [Breese and Fehling, 1988] also discuss a metalevel control architecture that uses multiple reasoning methods, but in their case, a particular method is chosen and then only that method is executed rather than combining the results of the methods.

The Metalevel ID is first used to compute the expected utility of the optimal action, $EU(d^*.t)$, by using the current running time of the decision cycle to instantiate the $Time$ node, propagating the current evidence from the EAs through the network, and determining the optimal action $D_{ML} = d^*.t$ and its expected utility $U_{ML} = EU(d^*.t)$. This calculation requires a measure of the "quality" of the computed results of the EA as a function of the time spent computing, t_c, that is, a "performance profile" for each EA as a function of t_c. We define the performance profile to be the probability that the decision that a particular EA would take after t_c is the same decision that would be taken by the metalevel.

Using the same method, the expected utility of the optimal action after waiting Δt, $EU(d^*.t')$, is also computed by setting

the node $Time$ to the sum of the running time of this decision cycle and an increment Δt.

The expected value of waiting Δt is

$$EVC(\Delta t) = EU(d^*.t') - EU(d^*.t) - TC(\Delta t) \qquad (1)$$

If $EVC(\Delta t) \leq 0$, then there is no benefit in waiting and the current best action, $d^*.t$, is executed. However, if $EVC(\Delta t) > 0$ then we should wait for more computation by the EAs and repeat the metalevel computation step.

4 AUV Example

We have implemented examples (300–400 nodes) of multiple EAs and planning windows for the AUV domain and have been encouraged by the results. For the implementation, we use the HUGIN software [Andersen *et al*., 1989] which operates only on *probabilistic belief networks* [Pearl, 1988]. Therefore, the extended influence diagram is converted to a corresponding belief network as in [Cooper, 1988] by converting decision and value nodes into probabilistic state nodes. On top of the HUGIN software, we have written C code to implement the multiple execution architectures framework and metalevel control of planning.

Since each EA has a different "performance profile" (decision quality as a function of computation time), in the multiple EAs implementation an interesting interaction that occurs between EAs as computation time varies can be observed. In our implementation, the Condition-Action performance profile quickly rises to a medium level of decision quality, while in contrast, the Decision-Theoretic performance profile has a higher decision quality but only after a longer period of computation. Therefore, as computation time increases, the results of the Condition-Action EA are weighted less, and the results of the Decision-Theoretic EA are more heavily weighted. For a given situation, the Metalevel's best decision may change several times as the EA computation time increases. Complex decision-making behavior is thus generated by specifying simple probability and utility functions and exploiting the modularity of the EAs.

We have also implemented the value of replanning mechanism and have seen it successfully replan when various dynamic events occur and successfully *not* replan when there is no or little benefit to be gained. In one example, after the execution of the first step of the nominal plan, a battery failure occurs. This event causes the value of replanning the $FuelGauge$ subtree to be positive, and after recursively computing values of replanning in the network, a decision is changed from waiting two cycles for data to accrue to waiting zero cycles, thereby reducing the use of fuel for the mission.

5 Conclusion

Two techniques used by RALPH-MEA to facilitate decision-theoretic reasoning were discussed. The *Multiple execution architectures* framework uses performance profiles to combine the information from four different implementations of the agent function to output an action choice. *Decision-theoretic planning* uses a projectible decision template to construct a planning window. Within a planning window, the value of replanning is calculated to provide metalevel control over planning and execution.

We are currently building a more complex model of the AUV domain to use for decision-making. As the model becomes more complex, the need for multiple EAs and decision-theoretic planning become more important, so we expect to get more impressive results from metalevel control. We are also working on acquiring and learning more accurate assessments of the requisite probabilities and utilities needed (e.g., the performance profiles).

References

[Andersen *et al*., 1989] S. K. Andersen, K. G. Olesen, F. V. Jensen, and F. Jensen. HUGIN— a shell for building belief universes for expert systems. In *Proc. 11th Intl. Joint Conf. on AI*. 1989.

[Breese and Fehling, 1988] J. S. Breese and M. R. Fehling. Control of problem solving: principles and architecture. In *Proc. 4th Conf. on Uncertainty in AI*. 1988.

[Cooper, 1988] G. F. Cooper. A method for using belief networks as influence diagrams. In *Proc. 4th Conf. on Uncertainty in AI*, 1988.

[Doyle, 1990] J. Doyle. Rationality and its roles in reasoning. In *Proc. 8th Natl. Conf. on AI*. 1990.

[Feldman and Sproull, 1977] J. A. Feldman and R. F. Sproull. Decision theory and A I II: The hungry monkey. *Cognitive Science*, 1:158–192, 1977.

[Hanks, 1990] S. Hanks. Practical temporal projection. In *Proc. 8th Natl. Conf. on AI*, 1990.

[Henrion *et al*., 1991] M. Henrion, J. S. Breese, and E. J. Horvitz. Decision analysis and expert systems. *AI Magazine*, 12(4):64–92, 1991.

[Horvitz, 1988] E. J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In L. N. Kanal and J. F. Lemmer, editors, *Uncertainty in AI*. North Holland, 1988.

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

[Kanazawa and Dean, 1989] K. Kanazawa and T. Dean. A model for projection and action. In *Proc. 11th Intl. Joint Conf. on AI*. 1989.

[Kjaerulff, 1992] U. Kjaerulff. A computational scheme for reasoning in dynamic probabilistic belief networks. In *Proc. 8th Conf. on Uncertainty in AI*, 1992.

[Newell, 1982] A. Newell. The knowledge level. *Artificial Intelligence*, 18(1):87–127, 1982.

[Nicholson, 1992] A. E. Nicholson. Qualitative monitoring of a robot vehicle using dynamic belief networks. In *Proc. 8th Conf. on Uncertainty in AI*, 1992.

[Ogasawara, 1993] G. H. Ogasawara. *RALPH-MEA: A Decision-Theoretic Agent Using Multiple Execution Architectures*. PhD thesis, UC Berkeley, Computer Science Division, 1993. In preparation.

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

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

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

[Watkins, 1989] C. J. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, UK, 1989.

[Wellman and Doyle, 1991] M. P. Wellman and J. Doyle. Preferential semantics for goals. In *Proc. 9th Natl. Conf. on AI*, 1991.