RALPH-MEA: A Real-Time, Decision-Theoretic Agent Architecture

by

Gary Hayato Ogasawara

B.A. (University of California at Berkeley) 1987
M.S. (University of California at Berkeley) 1990

A dissertation submitted in partial satisfaction of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION
of the
UNIVERSITY of CALIFORNIA at BERKELEY

1993

The dissertation of Gary Hayato Ogasawara is approved:

Chair	Date
	Date
	Date

University of California at Berkeley

1993

RALPH-MEA: A Real-Time, Decision-Theoretic Agent Architecture

by

Gary Hayato Ogasawara

Abstract

RALPH-MEA: A Real-Time, Decision-Theoretic Agent Architecture

by

Gary Hayato Ogasawara

Doctor of Philosophy in Computer Science

University of California at Berkeley

Professor Stuart J. Russell, Chair

This dissertation describes the RALPH-MEA agent architecture which uses decision theory combined with real-time control for decision-making in complex domains. In order to achieve the conflicting goals of an accurate representation and a fast decision cycle, several novel techniques are introduced.

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 (EAs) are defined by the different knowledge types that each uses. Depending on the domain and agent capabilities, each EA has different advantages. For example, a reactive, "if (condition) then (action)", production rule system will generally allow fast reactions, while a deliberative, decision-theoretic system will be slow but accurate and easily programmed with new knowledge and goals. A metalevel algorithm to combine the results of multiple EAs is given, and a decision-theoretic representation of the EAs as "extended influence diagrams" is defined.

Knowledge compilation is used to convert knowledge of one type to another. For example, the knowledge used by a decision-theoretic system (e.g., probabilities and utilities of outcome states) can be converted into knowledge used by a condition-action rule systems. A viable strategy is to acquire knowledge in one form and then to use knowledge compilation to convert the knowledge into the most efficiently executable form.

A view of decision-theoretic planning is also presented. Utilizing decision theory for planning facilitates the handling of uncertainty and multiple objectives. However, because of the high complexity of such planning, control of planning becomes a critical issue. *Metalevel control of planning* computes the value of information of planning to compare to the utility of executing the current default plan.

Acknowledgements

The completion of this work owes much to the efforts and contributions of others. Stuart Russell introduced AI in such a thrilling and beguiling way that it captured my interest for life. As an advisor, he has been unstinting with both his time and help, and most importantly, he taught me how to think independently and critically. My research here and in the future will always be an offshoot of the grand RALPH (Rational Agents with Limited Performance Hardware) project that Stuart has defined.

My education at Berkeley has been a long adventure, but with it, I gained the priceless gift of confidence, academically and elsewise. Alice Agogino and John Canny have consistently provided insightful and important feedback on diverse topics and have been invaluable advisors. I also owe thanks to my many other professors, TAs, and fellow classmates at Berkeley through the years.

I gratefully acknowledge the financial support of the National Science Foundation through an NSF Graduate Fellowship and NSF grants IRI-8903146, IRI-9058427, INT-9207213, and CDA-8722788. In addition, the Lockheed Artificial Intelligence Center has provided financial support in 1992 and 1993. At Lockheed, Raj Doshi has been a tremendous morale booster and steadfast supporter. Mike Beltz, Rich Pelavin, and Chris Toomey have helped develop some of the ideas and programs described herein.

To my fellow RUGS-mates (Russell's Unusual Group of Students), thanks for putting up with me over the years. To Sachiyo Arai, Francesca Barrientos, John

Binder, Jeff Conroy, Marie desJardins, Doug Edwards, Othar Hansson, Tim Huang, Sven Koenig, Andrew Mayer, Sonia Marx, Ron Musick, Ron Parr, Sudeshna Sarkar, Scott Schmidler, Jonathon Tash, Eric Wefald, Shlomo Zilberstein, and the others, "Yes!". An extra dose of gratitude to Ron Musick who has been a great friend and colleague. Thanks also to special friends who have helped me finish this thing: Maxine and Rodney, Curtis, Marina, Marcus, Jeff, Margaret, Jimmy and Mikey, and to all my relatives here and there. And finally and with most heart, thanks to my family (Dad, Mom, David, Lisa and Steve) who give me my home base.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The popularized goals of artificial intelligence have usually included not only creating intelligence but also explaining intelligence. Of course, these are not independent notions. If we can explain how intelligent behavior can come about, we can construct such an agent. If we can provide an agent *architecture* that can act as a foundation for intelligent behavior, we can create a general, intelligent agent.

This dissertation describes RALPH-MEA (Rational Agent with Limited Performance Hardware— Multiple Execution Architectures), a control architecture to make intelligent decisions in real time. Fundamental to this research program is the focus on providing a theoretical foundation for real-world decision-making that is bounded by limited resources of time and computing power. With such an architecture, control systems and expert systems can be developed for a wide range of applications (e.g., autonomous vehicle control and sensor management).

Artificial Intelligence was founded in the belief that there is a distinction between knowledge and the underlying physical layer that uses that knowledge. Then, it becomes reasonable to imagine that intelligence can be created as an organization of the "knowledge level" and applied to different physical implementations. Turing's article "Computing Machinery and Intelligence" [97] and Shannon's discussion of computer chess [87], both of which appeared in 1950, presupposed such a world view. McCarthy and Hayes [60] and Newell [64], among others, precisely targeted the organization/implementation distinction. For example, Newell discussed the "knowledge level" which lies above the symbol and hardware levels and operates on knowledge, e.g., beliefs and goals.

The organization of the knowledge level can be regarded as the *agent architecture* that remains fixed for any particular domain, so that the same architecture can be used whether reasoning about an underwater vehicle or a hockey player, and only the domain knowledge is changed. The critical task is to specify the architecture because its generality provides predictive and explanatory power that solutions to particular domains or reasoning problems lack. The philosophy, design, and implementation of such an agent architecture that supports intelligent decision-making in real-time, complex domains are the subjects of this research.

The type of domains and problems that we are interested in are real-world problems such as controlling an autonomous underwater vehicle (AUV) that once launched, must make autonomous decisions with respect to its objectives and environment. The AUV example is typical of tasks that require intelligent decision-making in that it is

1. **Real-time**. *When* a decision is made by the system matters (e.g., deciding

whether to move or not when a spear is flying towards your head).

2. **Complex.** If a decision problem is simple, it is easy to find a special-purpose method to make the decision. However, as the problem becomes more complex in type and scope, a principled architecture becomes more useful.

3. **High stakes.** When the outcome of behavior has high reward or high penalty, it is important to make correct decisions. Otherwise, there is no need for intelligent discrimination among decision choices.

These domain characteristics lead to the two major requirements for the agent architecture:

1. **Representational accuracy.**

   The domain must be represented in a form that can be used by the architecture. But as the domain becomes more complex, it becomes difficult to accurately model the domain and maintain that model. For example, the underwater terrain and currents impact the performance of the AUV, and how and at what level of detail geographical and meteorological data is represented needs to be carefully considered. This knowledge acquisition and modeling task is critical in building effective agents. The "knowledge-acquisition bottleneck" is often cited as the most crucial step in building expert systems (e.g., [46]).

2. **Inferential tractability.**

   Because of the real-time nature of the world in which the agent is situated, inference used to make decisions must be tractable. The world does not freeze while the agent decides what to do next, but is continually changing with time. The agent is situated in this dynamic world and needs to make real-time decisions.

The above two requirements are dependent and counteracting. Achieving one requirement is simple, but irrelevant. If the representation is made more accurate by modelling every tiny detail, inferential tractability becomes much more difficult to achieve. And likewise, inferential tractability can be achieved at the expense of an inaccurate representation. In a broad sense, the general problem of Artificial Intelligence (AI) can be construed as providing representational accuracy and inferential tractability to artificial agents. Systems that work only for small, toy problems like stacking computer-simulated toy blocks do not "solve" the AI problem because they cannot be scaled to real-life problems.

1.2 Approach

Our approach with RALPH-MEA *starts* with a complex problem framework with uncertainty about the world, multiple objectives, and resource-constrained computation. Though we scale up by first looking at simple problems in this framework, this approach is significantly different from assuming perfect knowledge about the world, black-and-white goals, and unlimited computation time.

1.2.1 Decision Theory

Decision theory provides a well-principled framework that explicitly considers world uncertainty and multiple objectives to compute the optimal decision to execute [98, 82]. The fundamental Maximum Expected Utility (MEU) principle states that the optimal decision choice d^* is computed using the probability distribution on the outcome states, $s \in S$, and a utility function on outcome states, $u(s)$:

$$d^* = argmax_{d \in D} \sum_{s \in S} p(s|d)u(s) \tag{1.1}$$

The probability and utility functions are generally assessed from a subjective perspective, and so decision theory is also known as subjective Bayesian decision theory.

As an example, consider the decision of whether to select a fair coin or a biased coin ($D = \{fair, \neg fair\}$) on a certain coin toss with outcomes $S = \{h, t\}$. Assume we have utilities of $u(h) = 1, u(t) = 0$, and the probabilities of the toss outcome depends on which coin is selected: $p(h|fair) = p(t|fair) = 0.5$, $p(h|\neg fair) = 0.6$, $p(t|\neg fair) = 0.4$. The expected utility, $E[u]$, of each d can then be computed:

$$E[u(d = fair)] = p(h|fair)u(h) + p(t|fair)u(t) = 0.5 \tag{1.2}$$

$$E[u(d = \neg fair)] = p(h|\neg fair)u(h) + p(t|\neg fair)u(t) = 0.6 \tag{1.3}$$

Therefore, $d^* = \neg fair$.

If an agent can continuously apply the 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 outcome states. To help alleviate this AI scalability problem, *multiple execution architectures* are employed as the key component strategy of RALPH-MEA.

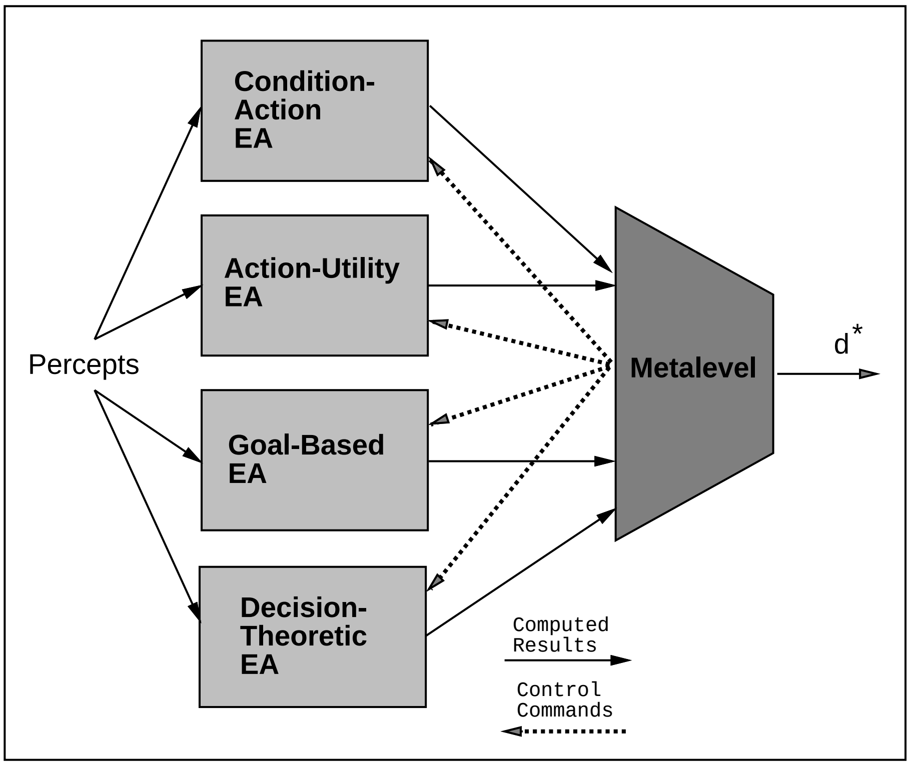


Figure 1.1: RALPH-MEA system architecture.

1.2.2 Multiple Execution Architectures

An agent can be considered as a function or "black box" that receives percept sequences from its environment and selects actions to take. The agent's sensors classify the environment into percepts which can be viewed as propositions, and percept sequences are the percepts received over a period of time. For example, percepts could be specific pixel values of a sensor, and a percept sequence could be those pixels sensed over a period of five seconds. The environment is external to the agent, but note that this does not preclude simulated environments which are independent of the agent. Actions include both external (or base-level) actions that act directly on the environment (e.g., moving the right leg) and internal (or computational) actions that change the agent's internal state (e.g., run the planner for ten more seconds). Instead of the traditional *sense-decide-act* loop where only the final step is under control, the agent executes an *act* loop that controls each phase of sensing, deciding, and external acting.

An execution architecture (EA) [75] is an implementation of the agent function that maps sequences of percepts from the environment to "best" action choices: $f : \mathbf{P}^* \rightarrow \mathbf{D}$[1]. A fixed number of EAs can be defined dependent on the *type* of knowledge the EA uses to implement the agent function. Different types of knowledge include goals, probabilities of states, outcomes of actions, and utility information. In parallel, each EA computes the same function to determine a "best" action choice, but with different types of knowledge. A *metalevel* control system monitors the computation of the EAs and determines an overall "best" action (see Figure 1.1).

Because each EA operates on different knowledge types, each EA has different profiles of cost of computation and quality of computed results. For example, one EA that uses "if $< condition >$, then $< action >$" rules will probably have relatively low computation costs, but the quality of its action recommendations also may be low. In contrast, a decision-theoretic planning system may have very high relative computation costs, but correspondingly very high quality results. Depending on the domain (e.g., the time-pressure and reward/penalty for decisions), a combination of the results of the EAs is the optimal strategy.

The motivation for having multiple EAs stems from the two requirements of an agent architecture: representational accuracy and inferential tractability. The domain

[1]The agent mapping can also be represented as $f : \mathbf{P} \times \mathbf{I} \rightarrow \mathbf{D}$ where $\mathbf{P}$ is the set of current percepts and $\mathbf{I}$ is the internal state of the robot.

will determine the effectiveness of a particular EA knowledge representation with respect to the two requirements. For example, the decision policy for a simple video game domain may be easily representable as "if-then" condition-action rules.

However, for more complex domains, specifying condition-action rules is infeasible because of the combinatorial explosion of the number of possible conditions. Instead, smaller pieces of knowledge such as utilities of states and effects of actions may be specifiable. Inferential tractability also points to the need for multiple EAs. If there is an unlimited amount of time available for computation, the single EA that has the highest expected quality of computed results for any amount of computation time would always be preferable. However, with time-pressure, the optimal decision must be determined with respect to time (e.g., it is useless to figure out the best treatment for a patient, after that patient has died.).

The use of multiple EAs requires development of a host of ancillary techniques. *Metareasoning*, reasoning about reasoning, is required to arbitrate the results of the EAs and determine a final decision choice. *Knowledge compilation* that transforms one knowledge type into another knowledge type also needs to be defined. *Planning* for decision-making can be specialized and analyzed for particular EAs. These topics occupy the following chapters.

1.3 Thesis statement

The RALPH-MEA agent architecture's general approach is to utilize multiple EAs in a decision-theoretic framework. With this approach, we show the following thesis:

Thesis:

The use of a decision-theoretic knowledge representation system, *multiple execution architectures*, and metalevel control of computation supports decision-making in complex, real-time domains where standard systems falter. Specifically, new results include:

1. A knowledge representation that partitions the space of possible decision-making systems or execution architectures (EAs) using an extension of standard influence diagrams. (Chapter 3).

2. A principled method to combine the results of heterogeneous, parallel decision-making systems to choose an action to execute. (Chapter 4).

3. A knowledge compilation strategy that provides a method to convert knowledge of one type into another type that is more compiled for run-time. (Chapter 5).

4. A decision-theoretic planning system that can handle uncertainty and multiple objectives. Value-of-computation analysis ensures tractability and real-time response. (Chapter 6).

5. Temporal reasoning capability for indefinite periods of time. The use of dynamic influence diagrams that can adjust as time passes allows an agent to operate and plan for indefinite periods of time. (Chapter 6).

We have made theoretical advances by showing the points of the above thesis statement. A sound theoretical basis is critical in defining an agent architecture because it can be analyzed, criticized, and built upon, providing the foundation for further technological advances.

In conjunction with the theoretical results, we also have made significant progress on practical issues. The development of this agent architecture has advanced the capability of practical, decision-making systems. For example, we will be describing an implementation of the execution architecture for an autonomous underwater vehicle (AUV) that has been developed at Lockheed Missiles and Space Company. A videotape of the demonstration is available. Other applications that are in progress are a decision-making system to control an automated automobile, a sensor management system, and a control system for a Nomadic mobile robot.

1.4 Outline

Before delving into the technical details of the RALPH-MEA agent architecture, Chapter 2 sets the background by discussing previous work and prerequisite topics. Chapter 3 describes the multiple execution architectures approach that is fundamental to the rest of the dissertation. Chapter 4 provides the algorithms used for the decision making of the execution architectures. Chapter 5 discusses knowledge compilation: how knowledge from one execution architecture can be converted to another execution architecture. Chapter 6 discusses planning in the context of decision-theoretic control and multiple execution architectures, showing how planning can be viewed as computational actions that can be subject to control. The concluding Chapter 7 relates other work, recaps the main points, and points to future directions.

Chapter 2

Background

This chapter briefly discusses previous, related work and prerequisite topics, thereby setting the background for the ensuing development of multiple execution architectures and related topics.

2.1 Agent Architectures

In this section, three broad research programs that share the goal of providing a general architecture for intelligence are discussed. The following three projects, Soar, RALPH, and the subsumption architecture, have been the primary influences of this work: Soar with its generality and its uniform representation and methodology, RALPH with its limited rationality model and its decision-theoretic metalevel theory, and the subsumption architecture with its behavioral decomposition and its implementations in real robots.

2.1.1 Soar

The Soar project's goal is to provide "the underlying structure that would enable a system to perform the full range of cognitive tasks, employ the full range of problem solving methods and representations appropriate for the tasks, and learn about all aspects of the tasks and its performance on them." [54] In Soar, every task is represented as a goal to be achieved, and *universal subgoaling*, where a subgoal can be set up for any decision, is used to decompose the problem into directly solvable pieces. Problem-solving becomes a search through a problem space (a space with a set of operators that apply to a current state to yield a new state). Metalevel problems, such as the choice of a problem-solving method, are considered as tasks themselves and solved through the same mechanism as a base-level problem. Long-term knowledge about search control and task implementation is represented by rules in a production system which provides input to the problem space search.

The ability to use the same method to solve any type of problem is advantageous since it allows for solving a broad domain of problems with a single architectural model. The uniform representation of tasks with problem spaces is also important. Having a single representation facilitates the structuring of the entire problem-solving situation as goal-achieving behavior. The problem space representation also allows flexibility in the system's state through changing the problem space or the production rules.

Soar has extremely general capabilities, but it is making some not so obvious assumptions. The problem space hypothesis is that problem spaces are the fundamental organizational unit of all goal-directed behavior, and the universal subgoaling hypothesis states that goal-oriented thinking can solve any problem. But having goals as Soar's basic unit raises problems with *goal identification* and *goal arbitration*.

The problem of goal identification is defining what a goal is and what its conditions of satisfaction are. Because the real world is dynamic and uncertain, this becomes a difficult problem. How does the search through the problem space proceed when there is uncertainty associated with the states and operators? If I'm 95% certain that I've achieved a particular goal, is that sufficient to say that I've achieved the goal? Soar does not seem to be able to support this type of reasoning under uncertainty.

Arbitration between goals is a difficulty for Soar because values are not associated with the states to express a preference among goals. A goal is simply a goal, and when two or more goals conflict there is no defined way to arbitrate between them. Soar relies on automatic subgoaling at impasses and control knowledge encoded as production rules to do the arbitration, but this requires careful problem-dependent coding of the production rule set. This goal arbitration problem is simply a reflection of the inadequacy of the goal structure hypothesis.

The Soar project also considers *chunking* which is a form of explanation-based learning to be *the* general learning mechanism [55]. Soar saves chunks which are generalizations of problem solving episodes. When a similar situation is reached in the future, a chunk can be used rather than re-solving the problem. Beyond the problem of which chunks to form and keep around (e.g., [96]), there is the problem of the adequacy of chunking. Chunks are generalizations, but they do not change the deductive closure of the original domain theory. Inductive learning (e.g., enumerative induction) which can modify the deductive closure of the domain theory is most likely necessary for an effective autonomous agent.

2.1.2 RALPH

Since this dissertation is part of the "RALPH" research program, previous work of the RALPH project is naturally relevant. As explicated by Russell and Wefald, the RALPH project's goal is the design of "robust software architectures for goal-driven, resource-limited intelligent systems" [78, 79]. Russell and Wefald's approach is to use a decision-theoretic metalevel architecture that explicitly reasons about computational actions as well as base-level actions, addressing the limited resources problem from the start rather than adapting an existing theory to try to fit the real world:

> "The construction of a system capable of rational metareasoning rests on two basic principles: (1) Computations are to be treated as actions, and are therefore to be selected among on the basis of their expected utilities. (2) The utility of a computation is derived for its expected effects, consisting of: (a) The passage of time (and associated changes in the external environment). (b) The possible revision of the agent's intended actions in the real world."

For their analysis, two assumptions are made. The *meta-greedy* assumption is that only single computational steps as opposed to all possible complete sequences of computation need to be considered when estimating the value of the computation; then the single step that appears to have the greatest benefit is selected. The *single-step* assumption is that a computation's value can be evaluated by considering it to be a complete computation (i.e., no further computations are done) rather than a possibly partial computation. For example, in game playing, a search tree is often generated where each node in the tree represents a particular state of the game and the arcs represent player moves. A individual node expansion of the game search tree can be used as the single computational step. Then, the meta-greedy and single-step assumption combine to say that each individual node expansion is considered to be a complete computation which is followed by a player move.

The meta-greedy assumption is made to make the problem tractable, but it is a significant assumption. In observed cases with game-playing trees, it has induced a "meta-greedy barrier" beyond which examining only single computational steps is not useful. The single-step assumption has been found to lead to underestimates of computation values in some cases.

Other research in the RALPH project include proving properties about bounded rationality [76], developing a language and algorithms for anytime algorithms [80, 105], and applications to various domains [20, 67, 69, 70]

2.1.3 Subsumption Architecture

Brooks' subsumption architecture [7] demonstrated the advantages of a *behavioral* rather than a *functional* decomposition of the mobile robot control problem. For example, self-contained task-achieving behaviors such as **avoid objects** and **wander** are used rather than functional modules such as **perception** followed by **modeling**. The behavioral layers are each constructed of finite state machines, and they are designed to run in parallel, as opposed to a sequential functional decomposition. Each layer corresponds to a level of competence— adding a new layer moves the robot's behavior to the next higher level of overall competence, "subsuming" the previous layers. The output of the highest layer always takes precedence over the other layers, but all layers run simultaneously, largely unaware of the other layers. The control is distributed among layers rather than being centralized, and the notion of internal representation or control is downplayed since one of Brooks' aims is to get complex behavior without a complex model.

One of the most notable advantages of the subsumption architecture is that it can be incrementally developed and tested. Lower-level layers can be built and tested independently, and then higher-level layers can be added on top of the lower layers. The mobile robot group at MIT has used the subsumption architecture to control many of the robots that they have built, successfully demonstrating the value of the architecture [27].

The problem with the subsumption architecture is that it is too inflexible and irrational.

Simon [91] distinguished between *procedural rationality* where the system computes the rational thing to do and *substantive rationality* where the system simply does the rational thing. Procedural rationality is dependent on the *process* that produces behavior while substantive rationality is not; it is dependent only on the actual behavior exhibited. The subsumption architecture may possess substantive rationality, but it is intrinsically unable to achieve procedural rationality. The primary reason is that the central control is sparse and cannot control the action with a global perspective. Another limitation is that the architecture is opaque: the control strategies are encoded in finite state machines, and beliefs, goals, utilities, etc. are not explicitly represented. It does not strive to do reasoning with utilities and probabilities in any sense; on the contrary, most of the knowledge is represented procedurally, which makes it difficult to reason, explain, or learn.

Because the architecture is hard-wired by the initial design, any significant reasoning or taskability[1] where external goals can be provided is not possible. One aspect of this inflexibility of the control system is that the final output must be one of the outputs of a behavioral layer. Each layer of the architecture subsumes all of its lower layers taking control through suppression or inhibition of another layer's signals. There are clear cases where the output of the control system should not be the output of one of its layers. For example, if we have the layers *Avoid Obstacles* and *Get to Goal*, they will generally output two different move commands. One layer is trying to avoid obstacles while the other is trying to move towards the goal. The best action might be neither of the moves suggested; it could be a function of the two moves or something else entirely. The answer is not to hard-wire solutions into some layer but to provide more intelligent arbitration between layers.

Another problem is the rigid architecture of the layers. A layer is composed of modules which are finite state machines with specified inputs, outputs, and variables which are used as local variables within the finite state machine. The modules are interconnected with input lines and output lines, and the output of a module is a function of the module's inputs and instance variables. An example module is *Avoid*, with inputs *force* and *heading*, output *command*, and an instance variable *resultforce*.

This layer structure makes it difficult to make modifications to the system. Suppose we wanted to change the input of a module to some different value. We would have to explicitly change the input of the module, the output function of the module,

[1]The ability to assign different tasks to the agent.

and the instance variable functions. Then we would have to propagate the changes to the adjacent modules, changing them so they can interact with the new input/output behavior of the changed module, eventually possibly modifying each module in the layer and also all other layers since the layers are interconnected with wires that do suppression and inhibition of signals.

The difficulty of changing the structure of the subsumption architecture leads to a limited capability agent which may be able to perform some simple tasks in an expected environment but is unable to achieve autonomy, be taskable, or reason in any significant sense.

Maes and Brooks [58] have added a distributed learning algorithm where each behavior tries to learn when it should become active. The algorithm attempts to learn a conjunction of preconditions for activating a behavior by using feedback from the agent's experiences. This learning procedure should improve the agent's ability to arbitrate between behaviors. However because of limited feedback (e.g., two binary touch sensors), the learning procedure is insufficient to distinguish between many of the behaviors. And if more detailed feedback is provided, the learning task becomes hopelessly complex due to the exponential (in the number of feedback percepts) complexity of the preconditions.

2.2 Decision theory

As mentioned earlier, decision theory is used throughout this research, and so here a quick introduction is provided.

Decision theory [82, 98] explicitly considers world uncertainty and multiple objectives to compute the optimal decision to execute. Decision theory's Maximum Expected Utility (MEU) principle states that the optimal decision choice d^* is computed using the probability distribution on the outcome states, $s \in S$, and a utility function on outcome states, $u(s)$:

$$d^* = argmax_{d \in D} \sum_{s \in S} p(s|d)u(s) \tag{2.1}$$

Certain assumptions are required for the validity of the MEU principle, and these assumptions can be divided into two classes relating to utility theory and subjective probability theory. The four assumptions about utility theory are the following (modified from [66]):

1. Any two possible outcome states resulting from a decision can be compared, expressing preference or indifference among the outcomes.

2. Preferences can be assigned in the same manner to "lotteries" as to outcomes themselves, that is, preferences can be assigned to lotteries like lottery $L = [s_1, p(s_1); \ldots; s_n, p(s_n)]$ where with probability $p(s_i)$, outcome s_i will occur. $p(\cdot)$ is a probability distribution over outcomes $S = \{s_1, \ldots, s_n\}$ satisfying $\sum_i p(s_i) = 1$.

3. There is no intrinsic reward or loss in lotteries, that is, "no fun in gambling". The decision-maker is indifferent about the number of lotteries if the outcome will be the same.

4. There is a continuity assumption on preferences such that there is some probability p where the decision-maker is indifferent between a lottery $L = [A, p; B, 1-p]$

and an outcome C which is preferred over A, but not over B.

The second part of the assumptions of decision theory is to accept the use of subjective Bayesian probability theory. Cox [12] shows that the acceptance of seven fundamental properties for continuous measures of belief in the truth of propositions necessitates the axioms of probability theory. Cox's seven properties have been summarized (from [44] and [9]):

1. *Clarity*: Propositions should be well-defined such that the proposition's value can be specified by a clairvoyant.

2. *Scalar continuity*: A single real number is both necessary and sufficient for representing a degree of belief.

3. *Completeness*: A degree of belief can be assigned to any proposition.

4. *Context dependency*: The belief assigned to a proposition can depend on the belief in other propositions.

5. *Hypothetical conditions*: There exists some function that allows the belief in a conjunction of propositions to be calculated from the belief in one proposition and the belief in the other proposition given that the first proposition is true.

6. *Complementarity*: The belief in the negation of a proposition is a monotonically decreasing function of the belief in the proposition itself.

7. *Consistency*: There will be equal belief in propositions that are logically equivalent.

Cox showed that these seven intuitive properties imply the axioms of probability theory, viz., that there exists a continuous monotonic function ϕ such that:

$$0 \leq \phi(A|e) \leq 1 \tag{2.2}$$

$$\phi(TRUE|e) = 1 \tag{2.3}$$

$$\phi(A|e) + \phi(\neg A|e) = 1 \tag{2.4}$$

$$\phi(A, B|e) = \phi(A|e)\phi(B|A, e) \tag{2.5}$$

where A and B are propositions; e is background knowledge. Bayes' theorem for conditionalization follows directly from the last axiom above:

$$\phi(B|A, e) = \frac{\phi(A|B, e)\phi(B|e)}{\phi(A|e)} \tag{2.6}$$

Accepting the assumptions of decision theory, we can use it and gain significant advantages. Foremost, decision theory provides a **principled theory of decision-making**. If an action has the maximum expected utility according to Eqn. 2.1, it is an optimal action[2]. The problem then becomes shifted down a level to correct assessment of the inputs to the MEU equation ($D, S, p(s|d)$, and $u(s)$). The representation of the problem in terms of utility functions on states and probability functions on states is very general and specifically geared to **handling world uncertainty** through the use of probability statements and **handling multiple, interacting objectives** through the use of utility functions.

Research on decision theory can be interpreted as extending the applicability of

[2]There may be more than one optimal action with the same expected utility.

the MEU principle to different classes of decision-making scenarios. For example, when the outcome state has multiple attributes (e.g., (dollars, health, world peace)), to solve Eqn. 2.1, the probability and utility functions are now over multiple, rather than single, attributes. This topic of *multiple attribute utility theory* [50] is directly relevant to real-life decision-making. Another important extension is the modeling of time-dependent utility functions. These utility functions can be used to extend the set of decisions D to include *computational* actions (e.g., compute for 2 seconds) as well as *base-level* actions that directly affect the world (e.g., move 2 meters right).

2.3 Probabilistic Reasoning

Standard probability theory can be used to combine probabilities through application of the basic axioms and Bayes Rule. The inferred probabilities can take into account particular evidence (e.g., Today's Weather = Sunny) or use the probability distribution of a variable (e.g., $p(Today'sWeather)$). Graphical representations of probabilistic dependencies have become important because they aid in modeling a situation and allow for efficient inference algorithms.

2.3.1 Graphical representations

An **influence diagram** [47] explicitly 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,

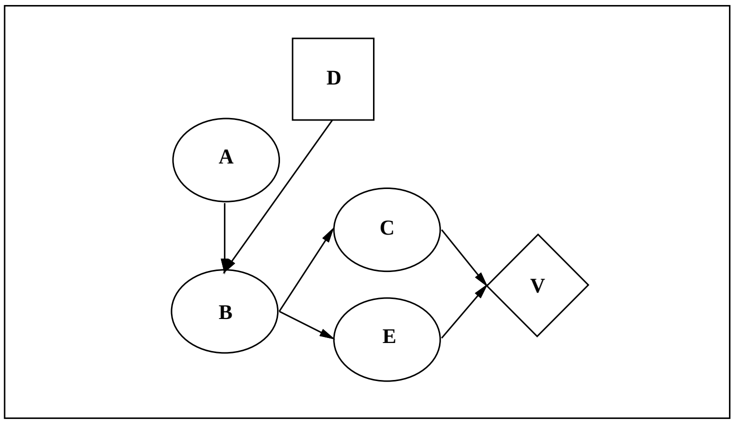


Figure 2.1: Sample influence diagram

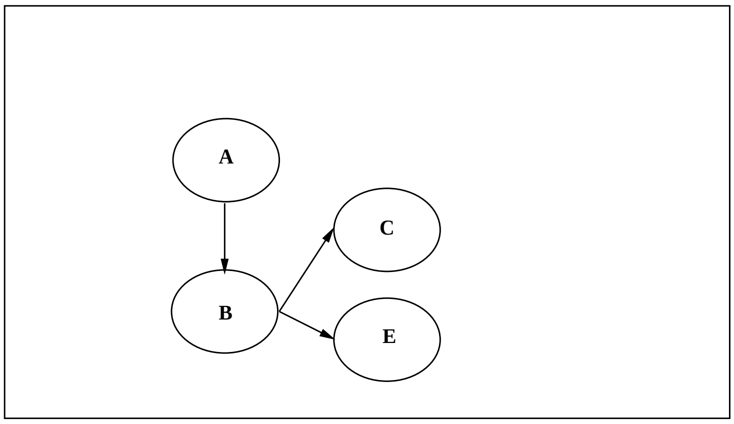


Figure 2.2: Sample belief network

informational, or value dependence. Figure 2.1 shows a sample influence diagram. Decision nodes are denoted by squares; probabilistic chance variables are ovals; and value nodes are denoted with diamonds. Arcs into a chance node indicate probabilistic dependence of variables. Arcs into a decision node indicate informational dependence of the decision upon other variables. Arcs into a value node indicate value dependence on the variables indicated.

A **probabilistic belief network** [72] is similar to an influence diagram but lacks decision and value nodes (e.g., the graph of Figure 2.1 with only the chance nodes and arcs, A, B, C, E). Inference in belief networks is used to determine probability distributions for a particular node or set of nodes given evidence about other nodes. Inference in influence diagrams is similar but adds the ability to evaluate decisions. For example in Figure 2.1, let B be "Tomorrow's Weather" which is dependent on A, "Today's Weather", and D, the decision of whether or not to seed the clouds. Probability distributions of the type $p(B = sunny|A = cloudy, D = seed)$ are given, and quantities like $p(B|A, D = seed)$ can be computed where probability distributions over the values of A and B are used. The key aspect of the inference is that it can be done through a sequence of local (using only the node's neighbors) operations (e.g., [56, 73, 84]). This is because all direct probabilistic dependencies are explicitly represented as adjacencies in the graph. Though fairly efficient in practice, the general inference problem has still been shown to be NP-hard in the worst case [11], and even stochastic simulation to obtain ϵ-error bounds on inferred quantities is NP-hard [14].

In the system described in this dissertation, **dynamic influence diagrams** are used. A dynamic influence diagram or belief network can continue to model the domain over a period of time by dynamically constructing and updating new nodes and arcs for new time slices. Most of these models assume some kind of Markov state assumption where variables of time $t + 1$ are directly dependent on nodes only from time t. Kjaerulff [51] 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. Hanks [32], Kanazawa and Dean [49], and Nicholson [65] also discuss similar temporal projection methods. To simplify the automatic projection, all of these methods (including the one presented here) assume that the dependency structure remains fixed (i.e., there can be no addition or removal of arcs or nodes). Changing the qualitative and quantitative structure of the influence diagram is a current active research topic (e.g., [29, 5, 13]). If the new structure of the influence diagram is known or can be learned, it can be integrated into most temporal projection methods without difficulty.

2.4 Metareasoning

Metareasoning, reasoning about reasoning, is, explicitly or implicitly, implemented in all control systems. However, most metareasoning strategies are fixed at design-time and lack flexibility. For example, Mitchell *et al.*'s Theo architecture [62] uses a fixed policy to always use cached knowledge if available, and otherwise to use a search algorithm to find an action to take.

To provide more control at run-time, decision theory has been applied to dynamically control the amount of reasoning based on utility of computation considerations. Horvitz [42], Russell and Wefald [78, 77] and Dean *et al.* [17], as well as many others (e.g., [21, 30, 39]), have developed the theory and applications of decision-theoretic metareasoning.

There has also been extensive research to implement metareasoning using other methods. Stefik [92] and Wilensky [103] discuss metareasoning in the context of control of planning. Control of planning is a key topic of this dissertation that will be addressed in Chapter 6.

2.5 Knowledge Compilation

Knowledge compilation is the conversion of declarative knowledge to compiled knowledge. Declarative knowledge (e.g., utilities $u(\omega)$ and probabilities $p(\omega|d)$) is generally easier to acquire and learn because the knowledge is available and representable in modular pieces. However, inference with declarative knowledge is usually expensive because it is not specially designed to be efficiently used for inference. In constrast, compiled knowledge (e.g., IF ω THEN d rules) is generally difficult to acquire directly but is designed to be efficient in execution.

Knowledge compilation is a familiar topic in the field of machine learning where "speedup learning" which improves the efficiency of a problem solver is concerned with the same issue [88]. The two general techniques that have been examined in speedup learning are (1) to learn new "macro-operators" by composing sequences of original operators, and (2) to learn metalevel control knowledge that can be applied to determine which operator to try next.

The use of generalized action plans or "macrops" by Fikes, *et al.* [23] in 1972 was of seminal importance in showing the advantages of knowledge compilation to speed up execution and the potential pitfalls such as the problem of maintaining too many compiled pieces of knowledge. Mitchell, *et al.*'s [63] "explanation-based generalization" and Laird, *et al.*'s [55] "chunking in Soar" describe domain-independent methods that generalize a problem-solving experience to help future problem solving. Other work includes Anderson's "knowledge compilation" that was used to learn production rules that chained together other rules [4], and Subramanian and Woodfill's use of situation calculus to generate propositional terms to be used in "if-then" rules [94].

Russell's definitions of knowledge types and compilation to travel between the knowledge types [75] is the multiple execution architectures approach discussed throughout this dissertation, and it is adopted for the compilation framework discussed later in this chapter.

A lot of recent work in the knowledge compilation research area has been to identify and remedy the problems engendered by existing frameworks. One class of problems can be called the "utility" problem where saving and using the compiled knowledge may not be beneficial. For example, Minton [61], Tambe and Rosenbloom [96] and others (e.g., [93]) showed the potential high costs of using generalized knowledge, most notably in the cost to find the applicable rule(s) to apply. Another class of problems is due to the lack of uncertainty in the domain theory. Because rule-based, logic-based systems generally lack the capability to handle uncertainty, if the original domain theory is incorrect or incomplete, the compiled knowledge will be incorrect or incomplete. Research allowing more robustness includes Tadepalli's explanation-based learning of approximate strategies [95] and Flann and Dietterich's use of uncertainty measures of theories updated by multiple examples [26].

A decision-theoretic approach to the compilation problem is interesting because it restricts the issues to the representation of probabilities and utilities. The "utility" and domain uncertainty problems seem to be tailor-made for decision theory. Heckerman *et al.* [35] develop a restricted notion of compilation where the choice is whether to pre-compute and save in a lookup table the effect of observing a subset of the possible evidence variables. An expected utility comparison of the compilation and non-compilation strategies can then be done. To make the problem tractable, assumptions are made about the model (e.g., single, two-value decision; single, two-value hypothesis; two-value evidence variables). Heckerman *et al.* also propose a "situation-action tree" where evidence items are examined sequentially until a decision is reached. Evidence nodes represent inner nodes of the tree, and particular decisions are the leaves. Lehner and Sadigh [57] also use situation-action trees and describe procedures to generate them from influence diagrams. Situation-action trees are limited because they do not admit uncertain variables, only observed evidence. This is an important issue because we would like to do partial compilation using probability distributions rather than only observations.

Herskovits and Cooper [40] discuss the automated design and construction of belief networks, which is relevant to knowledge compilation because multiple topologies of a belief network can represent the same joint probability function, and the "best" topology is the one that is "efficiently" representable and/or executable. They use information-theoretic measures to evaluate networks and a chi-squared stopping test to determine when to stop adding nodes to the network. D'Ambrosio and Shachter [85, 15, 16] also address knowledge compilation in a belief network representation by building intermediate representations that are efficiently executable for certain queries.

To round off the related work section on knowledge compilation, "reactive systems" should be mentioned. In sharp contrast to previous deliberative AI work that focused on knowledge representation, reactive systems were designed to react directly to sensor input without the need for significant internal models or processing. Examples of this type of work are Agre and Chapman's Pengi [1] which operated in a high-speed, dynamic video game environment and Brooks' subsumption architecture [7] for insect-like robots. Reactive systems demonstrated the benefit of having fast-executing, compiled systems and highlighted the range of domains that exist, including some that need deliberation and some that do not.

Chapter 3

Multiple Execution Architectures

This chapter describes the multiple execution architectures (EAs) representation. First, the components of the knowledge representation, the knowledge types, are introduced. Then the different EAs that use the knowledge types are defined. The multiple EAs representation partitions the space of possible decision-making systems or execution architectures using an extension of standard influence diagrams. Examples are presented using the autonomous underwater vehicle (AUV) domain.

3.1 Introduction

If an agent can continuously apply decision theory's maximum expected utility (MEU) principle (Equation 1.1) 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 and uncertainty about the utility of outcome states. This chapter discusses a general-purpose tool to alleviate this AI scalability problem: multiple execution architectures (multiple EAs). In later chapters, we will describe other techniques to control planning and learning that work within the decision-theoretic and multiple EAs framework.

Recall that 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 actions available to the agent. As defined by Russell [75], an execution architecture (EA) 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.). A fixed number of EAs can be distinguished by the different knowledge types that each uses.

The knowledge types originate from the fundamental view of the rational agent as operating in an environment by executing actions to achieve goals. With this view, there are three basic knowledge types that the agent possesses:

1. Knowledge about the environment or state of the world.

2. Knowledge about the results of actions.

3. Knowledge about the agent's preferences of world states.

As disussed in the next section, the three items can be further specialized to yield six knowledge types. For example, knowledge about the agent's preferences of world states can be expressed as either a numeric utility or as a universal goal.

The different EAs, defined by the knowledge types that each operates upon, will have different competences and costs in different situations — for example, reactive, $if < condition >, then < action >$ rules are good for shooting down missiles and playing standard chess openings, but more deliberative, 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.

3.2 Extended Influence Diagrams

The original specification [75] of the knowledge types used a situation calculus representation, but here, we introduce an extended influence diagram representation. The extended influence diagram representation differs from an influence diagram by defining the dependencies using the six knowledge types. These dependencies are more specialized than the general probabilistic, informational, and value dependencies used in an influence diagram. Specialized inference procedures can be used rather than the standard influence diagram inference techniques, allowing more efficient, specialized algorithms. What is being changed is the definition of the arcs in the influence diagram. Instead of having only *three* types of dependence (probabilistic, informational, and value), we define *six* types of dependence.

This approach of achieving inferential tractability by extending the types of influence diagram dependence can be contrasted to the approach of stochastic simulation (e.g., [37, 28, 86]) that maintains the full generality of the influence diagram representation and then uses a stochastic approximation algorithm for inference[1]. The

[1] In general, stochastic simulation for approximation is an orthogonal issue that can be integrated with the multiple EAs approach.

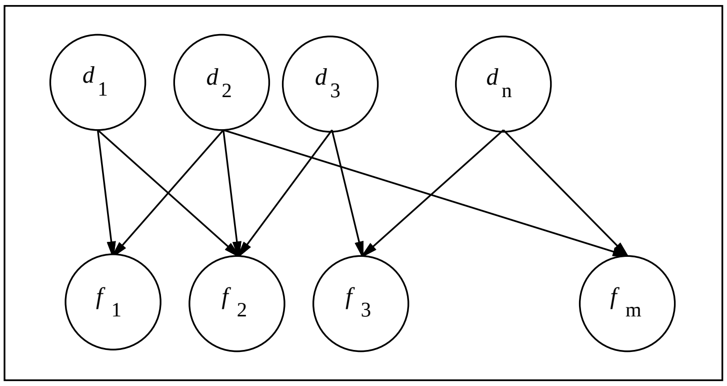


Figure 3.1: From Shwe *et al.*, this is a belief network used for the medical diagnosis system, QMR-DT. The disease nodes are labeled $d_1, ..., d_n$ and the finding or evidence nodes are labeled $f_1, ..., f_m$.

stochastic simulation can be halted at intermediate points to give the current beliefs of the desired inference quantity.

Another contrasting approach to achieve inferential tractability is to limit the topological structure of the influence diagram. An example of this approach is dividing the variables into classes of causes, the hypothesis, and evidence and assuming various types of conditional independence among the classes. An assumption of this type might be that particular pieces of evidence are conditionally independent of other evidence given the hypothesis. This restriction guarantees no dependency arcs between possible evidence variables (see Figure 3.1 [89]). With various topological restrictions, special-purpose inference algorithms can be used (e.g., Quickscore [34], S [89], and TopN [38]).

The extended influence digrams examined are also temporal, dynamically extending to represent new time periods. 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

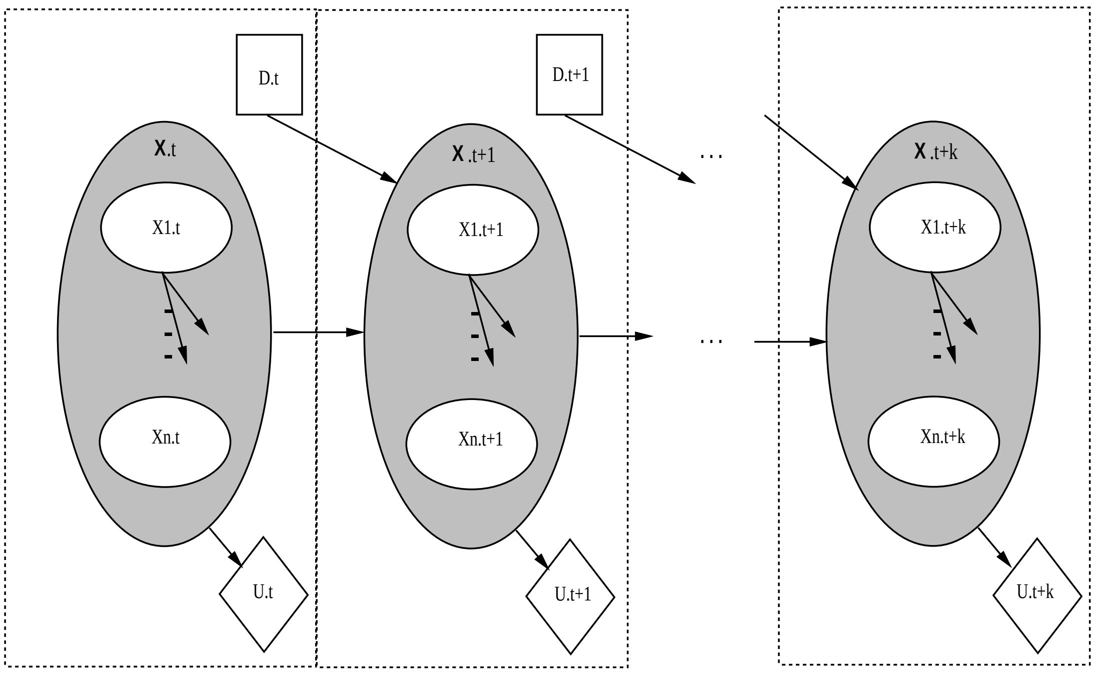


Figure 3.2: A dynamic influence diagram represents $k+1$ time slices.

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 $U.t{+}1$ (Figure 3.2).

We assume a Markov property where the probability of the next state is independent of all previous states given the current state[2]:

$$p(\mathbf{X}.t{+}1|\mathbf{X}.t, D.t) = p(\mathbf{X}.t{+}1|\mathbf{X}.t, D.t, \mathbf{X}.t{-}1, D.t{-}1, \ldots)$$

The Markov property allows specification of the influences on $\mathbf{X}.t{+}1$ using only $\mathbf{X}.t$ and $D.t$.

[2]The Markov assumption can be relaxed to higher-order Markov processes with longer-term dependencies, but doing so complicates the notation.

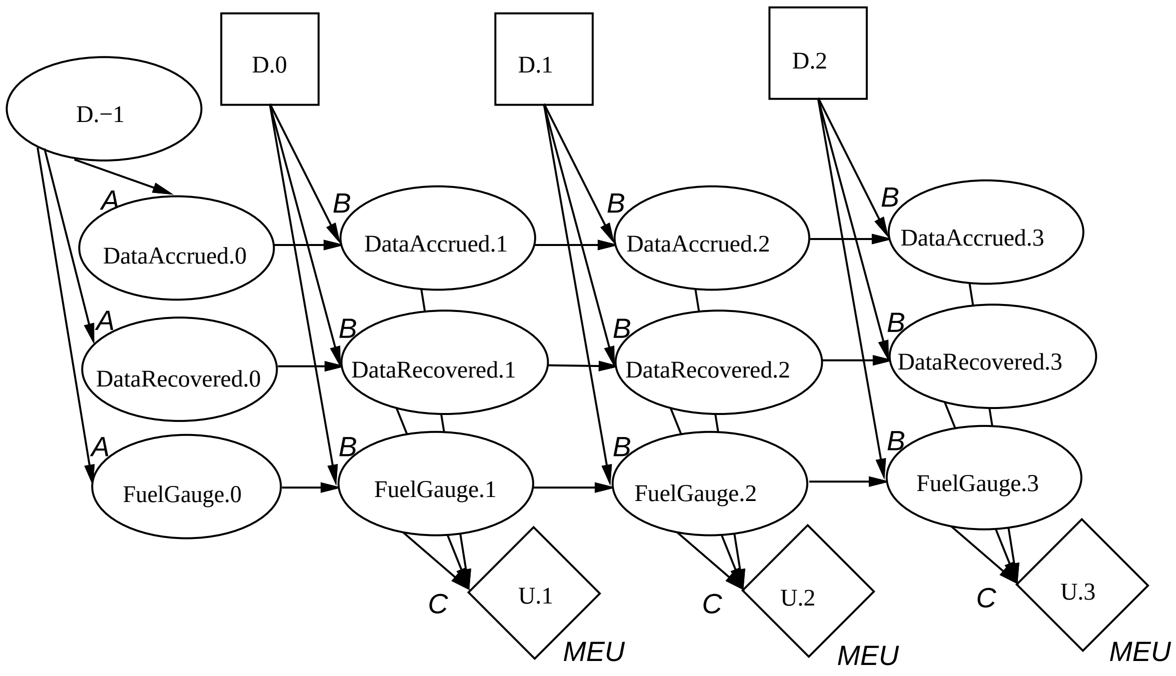


Figure 3.3: A Decision-Theoretic EA influence diagram with a 3-step planning window.

3.2.1 Planning windows

Dynamic influence diagrams of the type shown in Figure 3.2, can be used to represent action sequences rather than only single actions. Assuming the Markov property of the states, the same decision template can be reproduced repeatedly using only local connections as the world state is projected forward in time. The *planning window* is the multiple time-step influence diagram that is currently being considered.

Figure 3.3 shows a 3-step planning window obtained by chaining together three time slices of an Autonomous Underwater Vehicle (AUV) example. The AUV in this example has three objectives: to let data accrue in a sensor array; to recover the sensor data; and to minimize fuel usage. Each of these variables is dependent on the action taken by the AUV.

3.3 Knowledge Types

With these preliminaries, the extended influence diagram representation can be described using the following definitions of the six knowledge types:

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 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 the action on other nodes. 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.

C: $v_C(U.t{+}1|\mathbf{X}.t{+}1) \in [u^{\perp}_{DT}, u^{\top}_{DT}]$

Type C rules are utility functions on the state. In the extended influence diagram, this is represented using a value function v on the node $U.t{+}1$ which is bounded by the minimum and maximum possible utilities, $u^{\perp}_{DT}, u^{\top}_{DT}$. The DT subscript indicates that the utility function is for the Decision-Theoretic execution architecture. C rules are used only by the Decision-Theoretic EA, and each EA may have different utility functions[3].

D: $v_D(U.t{+}1|D.t, \mathbf{X}.t) \in \{u_{CA}^{\perp}, u_{CA}^{\top}\}$

Type D rules are akin to standard condition-action rules used in production systems. They state that if certain conditions hold (i.e., the $\mathbf{X}.t$ variables), then the specified action is the best action. The utility $U.t{+}1$ is conditioned on $\mathbf{X}.t$ and $D.t$ of time t. Type D rules express absolute certainty about the best decision(s). By the assignment of the value of $u_{CA}^{\top}$ to the best decision(s), it guarantee represent the value of by using the endpoints of the utility range as their only possible values. The CA subscript indicates that the utility function is for the Condition-Action EA.

E: $v_E(U.t{+}1|D.t, \mathbf{X}.t) \in [u_{AU}^{\perp}, u_{AU}^{\top}]$

Type E rules are "action-value" rules that specify the utility of condition-action pairs, for example Watkins' Q-tables [100]. In the extended influence diagram representation, they represent the same influences as D rules but the utility value is generalized to the range $[u_{AU}^{\perp}, u_{AU}^{\top}]$. The relevant utility values are for the Action-Utility EA.

F: $v_F(U.t{+}1|\mathbf{X}.t{+}1) \in \{u_{GB}^{\perp}, u_{GB}^{\top}\}$

The type F rule, which is similar to a classical goal [64], expresses the utilities of states in the future time period, $t{+}1$. The utility is restricted to $u_{GB}^{\perp}$ or $u_{GB}^{\top}$ values. The subscript GB refers to the Goal-Based EA that uses type F rules. A goal state is indicated when $v(U.t{+}1|\mathbf{X}.t{+}1) = u_{GB}^{\top}$. A non-goal state is when $v(U.t{+}1|\mathbf{X}.t{+}1) =$

[3]The execution architectures will be defined precisely in the next section.

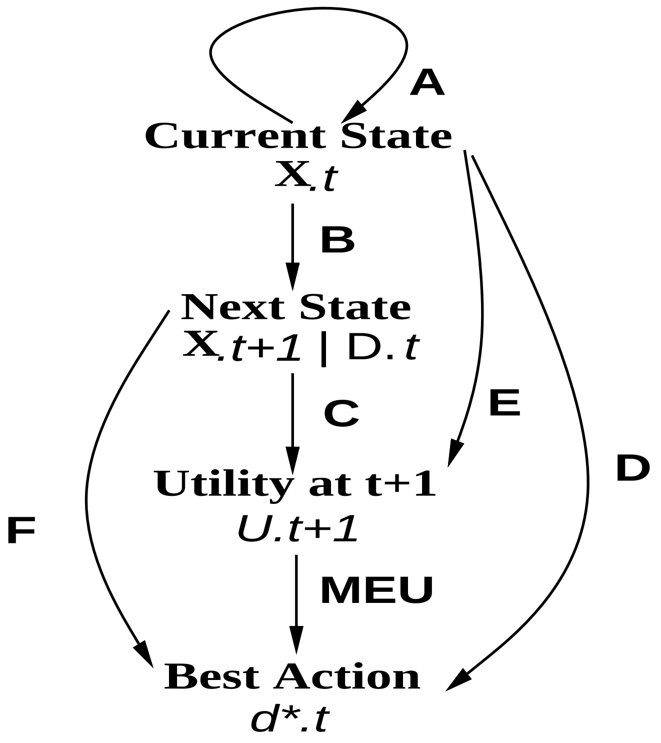


Figure 3.4: Knowledge forms for multiple execution architectures.

$u_{GB}^{\top}$. Type F knowledge is a specialization of type C knowledge which allowed the utility can range between $u_{DT}^{\perp}$ and $u_{DT}^{\top}$. C and F knowledge types in conjunction with B type knowledge are used to do temporal projection and planning as will be discussed in Chapter 6. C and F knowledge provides the utility of future states and B knowledge provides information on the effects of actions. The amount of time that the knowledge types are projected forward determines the "horizon" of the planning.

3.4 Multiple Execution Architectures

Four execution architectures (EAs) can be defined using the four different combinations of knowledge types to make an action decision (see Figure 3.4). Each EA is computing to determine the optimal decision, d^*.

3.5 Decision-Theoretic EA

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. Inference in the Decision-Theoretic EA uses the standard probabilistic and decision-theoretic reasoning techniques of 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 and the action with the maximum expected utility is selected. By chaining Type B knowledge, the Decision-Theoretic EA can project into the future indefinitely. This temporal projection is used for the planning system discussed in Chapter 6.

The Decision-Theoretic EA, in comparison with the other EAs, uses the most "uncompiled" forms of knowledge, including information about the current state, next state, and utilities. Through use of the knowledge compilation transformations, the knowledge used for the Decision-Theoretic EA can be converted to knowledge for the other EAs. Chapter 5 will address this topic in more depth.

Knowledge of Type A (state information), Type B (effects of actions) and Type C (utilities of states) are likely to be available from direct observations of the environment. The agent can learn such knowledge by operating in the environment and recording the results of its actions.

3.6 Goal-Based EA

GB: (path $A\ B\ F$)

Knowledge of types A, B, and F suggests actions that achieve the desired goal condition. And as in the Decision-Theoretic EA, temporal projection can be done using type B knowledge.

A Decision-Theoretic EA can be converted to a Goal-Based EA using the compilation $C + MEU \rightarrow F$. This compilation transforms utility of state knowledge into a fixed policy of executing the best action(s). If the Type C knowledge of the Decision-Theoretic EA remains valid, the compilation into Type F knowledge maintains behavioral equivalence of the Goal-Based and Decision-Theoretic EAs. For execution purposes, having a fixed decision policy rather than comparing utilities via MEU Principle is clearly preferable. However, dynamic changes in the preference structure may require going back to Type C knowledge and re-compilation.

3.7 Action-Utility EA

AU: (path $A\ E\ MEU$)

Knowledge of type E for various actions is combined with the MEU principle to select the most valuable one. As in the Decision-Theoretic 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. Because there is no explicit temporal projection in the knowledge types of the Action-Utility EA, the performance of this EA depends strongly on the quality of the type E knowledge (i.e., the utilities of the state-action pairs).

The knowledge types for the Action-Utility EA can also be generated through compilation of the knowledge types used for the Decision-Theoretic EA. In this case, the $B + C \rightarrow E$ compilation is used to generate Type E knowledge.

3.8 Condition-Action EA

CA: (path A D)

Knowledge of type D provides best action choices directly. The Condition-Action EA is very similar to standard rule-based production systems that use $if - then$ rules. The antecedent of the rules are part of the current state and the consequents are either other state information or an action choice, corresponding to Type A and Type D knowledge, respectively.

The knowledge necessary for the Condition-Action EA can be generated through two compilation paths. Knowledge for a Goal-Based EA can be converted to Type D ($B + F \rightarrow D$) or knowledge for an Action-Utility EA can be used ($E + MEU \rightarrow D$).

3.9 Why multiple EAs?

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.

3.10 Example

Figure 3.5 shows an influence diagram representation of each of the four EAs for a single decision. The example is a simplified version of the autonomous underwater vehicle (AUV) domain, in which the AUV does various information-gathering activities. The utility of its mission, node $U.t{+}1$, 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, $D.t$.

3.11 Representation of Decision Policies

In both the Condition-Action EA and the Goal-Based EA, the decision choice is made by using the decision policy according to Type D or Type F knowledge. There is no direct comparison of utility values using the MEU principle as in the Action-Utility or Decision-Theoretic EA. However, in the extended influence diagram representation, it is necessary to express all types of knowledge as either probabilities or utilities.

When the conditions $\mathbf{X}.t$ are all evidence variables that are observed, there is no problem since there is no uncertainty about observed values. For example, there may be an evidence variable $BatteryHealth.t$ with values *good*, *bad*, and at every cycle it is monitored and a value either *good* or *bad* is entered as evidence for the variable. As will be shown in Chapter 5, we can often compile out all probabilistic variables and leave only observable evidence variables.

In many production systems, uncertainty regarding the preconditions of a rule is not considered. A precondition is either 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 or Goal-Based EA by allowing no uncertainty about the conditions ($p(X_i.t|\mathbf{X}.t) \in \{0,1\}$). The EA then would output the first action, d_i, it considers such that $v(U.t+1|D.t = d_i, \mathbf{Y}.t) = u^{\top}$ where for the rule's preconditions, $\mathbf{Y}.t$, $\forall_{X_i.t \in \mathbf{Y}.t \subseteq \mathbf{X}.t}\ p(X_i.t|\mathbf{X}.t) = 1$.

One heuristic method is to remove the uncertainty from a condition by assigning it its most probable value. If, in combination with the other EAs, probabilities are known for the various values of a condition, the condition value with the highest probability can be selected as the "value" of the variable. For example, if the condition variable $FuelGauge.1$ has possible values $0, low, high$ and the respective probabilities of those three values is 0.4, 0.5, 0.1, the value of $FuelGauge.1$ is assumed to be low.

An alternative method that does not discard information allows uncertainty about the preconditions ($p(X_i.t|\mathbf{X}.t) \in [0,1]$). 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 two values: $\{u^{\perp}, u^{\top}\}$. For example, if $u^{\perp} = 0$, many propagations of values in the influence diagram can be immediately pruned.

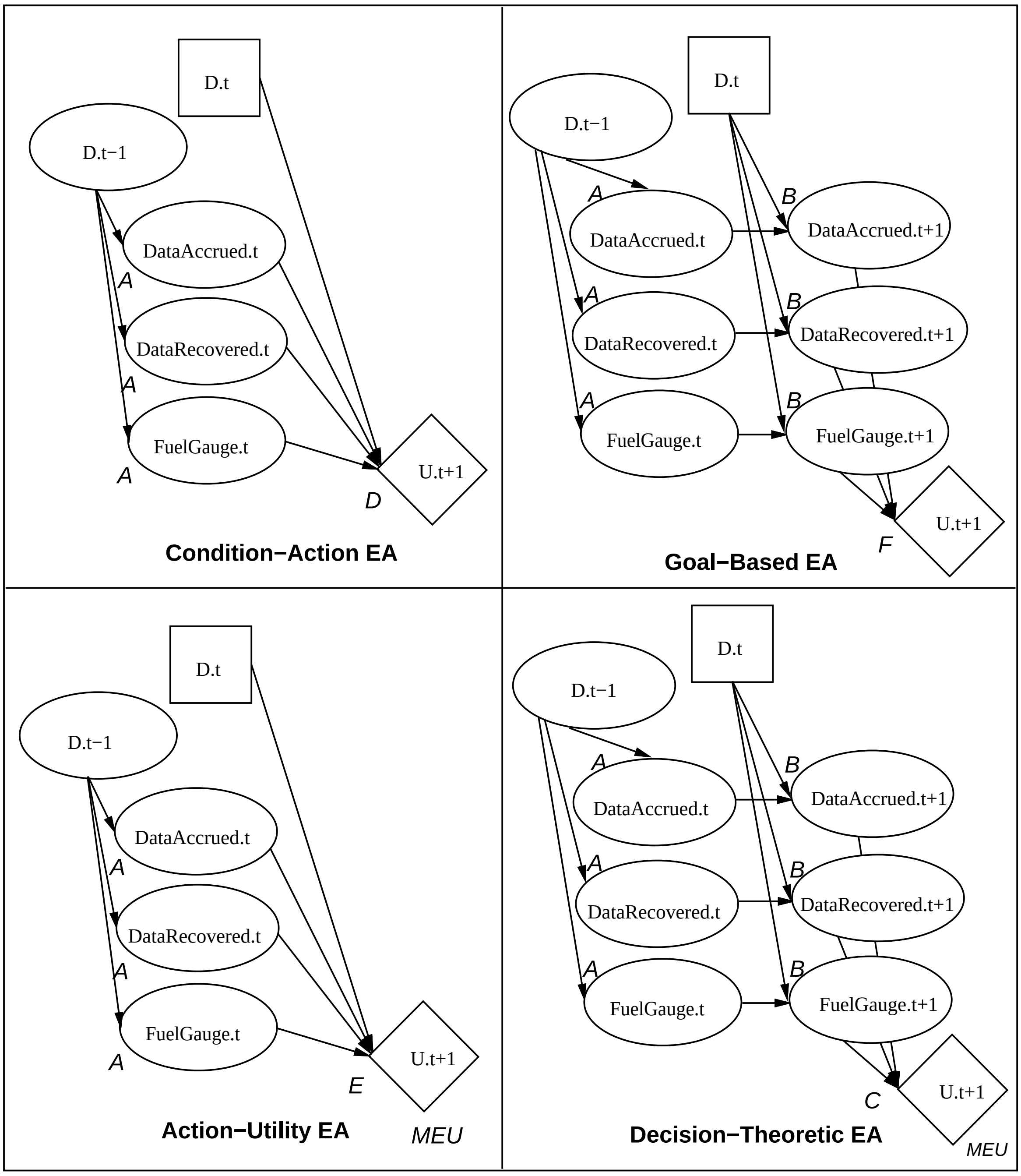


Figure 3.5: Influence diagram representations for the four execution architectures in the AUV domain. Decision nodes are squares, value nodes are diamonds, and state nodes are ovals. The conditional probability functions or utility functions at the nodes are labeled by their type of knowledge (A, B, C, D, E, F, MEU) they represent. The suffix t or $t+1$ denotes the node's time.

Chapter 4

Metareasoning

This chapter discusses how the four EAs are collectively used by employing metareasoning to combine the computation of them. A decision-theoretic approach is taken to combine the results of the multiple EAs. This algorithm provides a general, principled method to combine the results of heterogeneous, parallel decision-making systems.

4.1 The Decision Cycle

Given the multiple execution architecture representation, there is still the issue of how to use the four execution architectures together to make decisions. In our current implementation our strategy is the following: 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, and a separate "Metalevel" which receives updates from the four EAs and makes the final decision on the action choice by checking the values

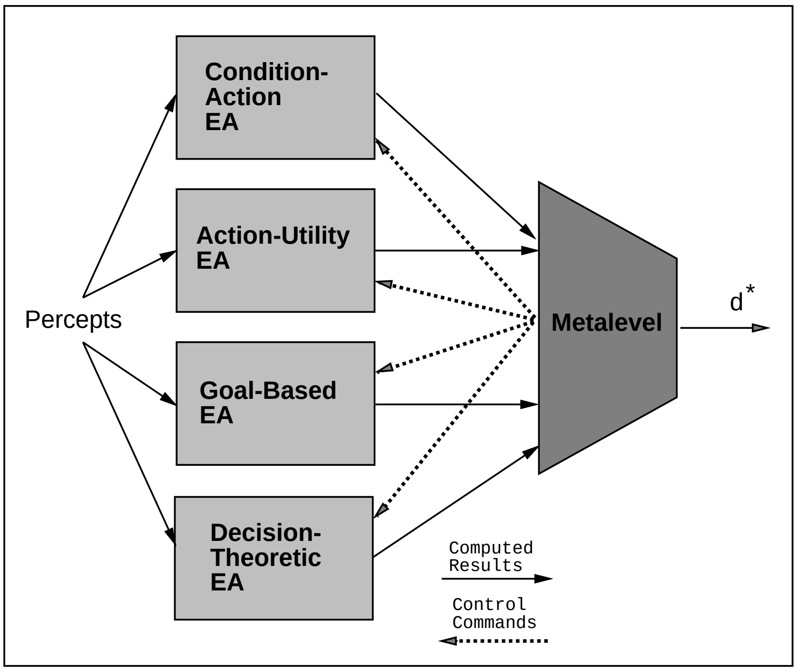


Figure 4.1: RALPH-MEA System Architecture

of a “best action” node. The metalevel has information about the quality of results from an EA given a particular amount of computation time. Using this information and asynchronous updates from the four EAs, the metalevel decides when it is best to execute the current best action rather than to wait for more computation to be done by the EAs.

At this stage of the research, the design method of separating the four EAs has the advantage of modularizing the EAs to allow the design and performance of individual EAs to be isolated[1]. The Metalevel is also separated from the implementation of the EAs and can be developed independently. Figure 4.1 shows the high-level system architecture view of RALPH-MEA.

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

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

occurs:

1. The model is updated by advancing the influence diagram one time step and entering new evidence.
2. Each EA computes its action choice.
3. The Metalevel computes the final action choice.

The following sections examines each of these steps in turn.

4.1.1 Updating the model

As the agent makes decisions, the planning window is updated by modifying the current 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

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 4.2: The Updating the Model step.

of the new influence diagram and new evidence to all the nodes. Figure 4.2 summarizes this Updating the Model step.

As discussed previously, similar techniques have become common recently in the uncertainty community [51, 32, 49, 65].

4.1.2 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 to update *its* assessment of the action utilities and to determine when to stop computing and output the current best action. The EA's run in parallel and if the Metalevel decides to start executing the decision cycle's action before some EA has finished its computation, the EA is interrupted to start the next decision cycle.

There are two types of EA computation: (1) the Action-Utility and Condition-Action EAs select a best action based on the current state; (2) the Decision-Theoretic

1. Read monitor data and enter evidence for each condition variable.
2. Use table lookup to find which rule corresponds to the current condition vector, and select the corresponding action.

Figure 4.3: The EA Computation step for the Action-Utility and Condition-Action EAs.

and Goal-Based EAs use a temporally-projected model and use planning controlled by value-of-planning considerations.

Action-Utility and Condition-Action EAs

The computation of the Action-Utility and Condition-Action EAs is fairly straightforward. Our design decision is to use condition values that are certain by assuming that only evidence variables remain in the model to act as condition variables. Then, the rule whose antecedent matches the current condition vector is selected, and its consequent, an action, is selected. This step is done by table lookup where the table is indexed by the condition vector.

For example, suppose there are two condition variables that we receive evidence for: $DataRecovered = \{T, F\}$ and $FuelGauge = \{0, low, high\}$. For both the Action-Utility and Condition-Action EAs there will be a utility table of size $|DataRecovered| \times |FuelGauge|$ with an entry for each possible condition vector. The table entry that matches the current condition vector is output available to the Metalevel.

Decision-Theoretic and Goal-Based EAs

The basic approach to using decision theory for sequential decisions is to do compute the expected utility of every sequence of decisions (i.e., plan) and execute the first decision of the plan with highest utility. This approach is intractable for all but the simplest domains and shortest length plans because the number of decision sequences is exponential in the length of the plan and each plan must be evaluated by the influence diagram. Goal-based planners have much of the same intractability because a single change in a plan may require much of the rest of the plan to be re-examined (e.g., see [8]).

Because of the intractability of classical approaches to planning, a metalevel control strategy is used that reasons about the cost of computation. With the Decision-Theoretic and Goal-Based EAs in the EA Computation step, the best action is determined using a real-time planner that considers replanning actions to change the choice of base-level actions. The general idea is that the value of executing the next step of the current default plan is compared directly to the value of replanning the default plan. The action with the highest estimated value (either the next default plan step or a replanning action) is executed. Chapter 6 gives a more extensive treatment of the replanning control and algorithms.

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 default plan step as its action choice.

```
1. $R = U.\mathrm{t} + \mathrm{k}$
2. Compute $\hat{VI}(replan(X))$ for each subtree $X$ of $R$ and insert $X$ in ordered-by-VI
   open list.
3. $X_{max}$ = first element of open list
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 4.4: The EA Computation step for the Decision-Theoretic and Goal-Based EAs.

Figures 4.3 and 4.4 summarize the two types of EA computation step.

4.1.3 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 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 similar to the basic metalevel computation algorithm described in Horvitz [41], Horvitz, *et al.* [43], and Russell and Wefald [78]. Breese and Fehling [6] 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. Zilberstein [104] does decision-theoretic analysis of the optimal combination of program modules with

```
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 4.5: The Metalevel Computation step.

different characteristics.

The Metalevel 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 $U_{ML} = EU(d^*.t)$.

Then, in the same way, the expected utility of the optimal action after waiting Δt, $EU(d^*.t')$, is computed by setting the node $Time$ to the sum of the running time of this decision cycle and an increment Δt. Δt can be conveniently selected to be the time required to do the metalevel computation. The expected utility of the optimal decision may change from time t to time t' due to different results being available from the EAs. A function of the time cost of delaying Δt, $TC(\Delta t)$ is also required. We assume that $TC(\Delta t) = c * \Delta t$ where c is an adjustable parameter to measure the time-criticality of the current situation[2].

[2]More complex TC functions can also be used (e.g., functions that use knowledge of the type of decision being made).

The expected value of waiting Δt is

$$EVC(\Delta t) = EU(d^*.t') - EU(d^*.t) - TC(\Delta t) \tag{4.1}$$

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 by repeating the metalevel computation step. Figure 4.5 summarizes the Metalevel Computation step.

Once the Metalevel decides to exit with the final action choice, this action is executed, and the decision cycle repeats going back to the Update Model step.

Computing EU(d)

To find the optimal value $d^*.t$ for D^{ML}, the metalevel decision, after a certain amount of computation time t_c, we can compute the expected utility for every $d_i \in D^{ML}$ by instantiating the decision to d_i, inserting the evidence from the EAs, and then selecting as d^* the d_i that yields the highest utility:

$$d^* \quad = \quad argmax_{d_i \in D^{ML}}\, u(d_i^{ML}, t_c) \tag{4.2}$$

where the utility of a decision is defined as the expected utility of the outcome states $\omega \in \Omega$ given that decision:

$$u(d_i^{ML}, t_c) \quad = \quad E[u(\omega)] = \sum_{\Omega} u(\omega)p(\omega | d_i^{ML}, tc) \tag{4.3}$$

We then define an outcome state ω to be a vector of the decisions recommended by each execution architecture and a context variable ξ:

$$\omega \in \Omega = (D^{CA}, D^{AU}, D^{GB}, D^{DT}, \xi) \tag{4.4}$$

Using the decisions of the individual EAs as the outcome state attributes at the metalevel is equivalent to saying that the attributes at the EA level are distilled respectively into the single attributes of the decision recommendations of the EAs. This is the viewpoint that we want to emphasize. The EA computations are a "black box" to the metalevel; as shown in Figure 4.1, the EAs forward computed results to the metalevel which the metalevel can use to make its decisions. In this case, the computed results of the EAs are the decision recommendations.

Substituting for ω, the decision utility is as in the equation below. (For notational convenience, the remainder of this section will use only the decisions from the Condition-Action (CA) and Action-Utility (AU) EAs. This can be generalized in the obvious way to include the other two EAs.)

$$u(d_i^{ML}, t_c) \quad = \quad \sum_{j,k} u(d_j^{CA}, d_k^{AU}, \xi) p(d_j^{CA}, d_k^{AU}, \xi | d_i^{ML}, t_c) \tag{4.5}$$

From the right-hand side of Equation 4.5, we have multi-attribute utility and probability functions which we need to specify. Given particular (in)dependence properties of the multi-attribute utility and probability functions, different functional forms can be defined.

Utilities First, we consider the multi-attribute utility function. Keeney and Raiffa's text [50] is an excellent source for this topic. The first step is to convert the multiattribute expected utility function into simpler single or multiattribute functions. The intuition is that the uncertainty about the utility is moved from the entire utility to the component attributes that are used to calculate the utility. The transformation can be done if the attributes are either

1. mutually utility independent and probabilistically independent, or
2. additive independent.

Mutual utility independence is the condition when all pairs of attributes are *utility independent* of each other. We say that attribute Y is utility independent of attribute Z when conditional preferences for lotteries on Y given a particular attribute value z do not depend on the particular level of z. In other words, the relative valuation of y_1 and y_2 is the same regardless of the value of z. For example, let Y be the set of times when a show is playing and Z be the set of seat sections in the theatre. If an 8:00 pm show is twice as preferable as a 2:00 pm show when I have orchestra seats, it should also be twice as preferable with balcony seats if the show times are utility independent of the seat sections.

Probabilistic independence means that the probability of attributes are conditionally independent of one another given a particular context. Therefore, for two attributes Y and Z and context ξ, $p(Y, Z|\xi) = p(Y|\xi)p(Z|\xi)$. For example, the probability of rain is conditionally independent of the day of the week: given that it is Sunday rather than Monday does not change the probability of rain.

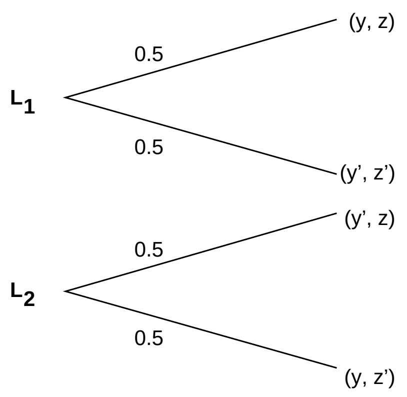


Figure 4.6: If lotteries L_1 and L_2 and equivalent for all amounts of attributes of y and z and fixed values of y' and z', additive independence holds.

Additive independence is a stronger condition than mutual utility independence, and it holds between Y and Z if and only if the lotteries $L_1 \equiv (0.5 : (y, z), 0.5 : (y^{'}, z^{'}))$ and $L_2 \equiv (0.5 : (y^{'}, z), 0.5 : (y, z^{'}))$ are indifferent for all amounts of y, z given a specific $y^{'}, z^{'}$ (figure 4.6). An alternative definition is that Y and Z are additive independent if the preferences over lotteries on Y and Z depend only on their marginal probability distributions for these attributes and not on their joint probability distribution. As an example, let Y, Z be as in the theatre example and $y^{'}$ be the 8:00 pm show and $z^{'}$ be orchestra seats. Now, any choice of y, z should make lotteries L_1 and L_2 equivalent in value. For example, there should be no preference between $L_1 \equiv (0.5 : (2{:}00pm, balcony), 0.5 : (8{:}00pm, orchestra))$ and $L_2 \equiv (0.5 : (8{:}00pm, balcony), 0.5 : (2{:}00pm, orchestra))$.

Once the expected utility function has been converted into an equivalent utility function, the utility function with multiple attributes still must be solved. The two functions that we will examine are the additive utility function and the multilinear utility function that correspond to the additive independence and mutual utility independence cases, respectively. The equations will be stated for three attributes, X, Y, Z, but they can be extended to n attributes. Let x^0 and x^* be the values of X that give the minimum and maximum utilities respectively. y^0, y^*, z^0, z^* are defined analogously.

The *additive utility function* has the form

$$u(x, y, z) = k_x u_x(x) + k_y u_y(y) + k_z u_z(z) \tag{4.6}$$

where

$$u(x^0, y^0, z^0) = 0 \quad u_x(x^0) = 0 \quad u_y(y^0) = 0 \quad u_z(z^0) = 0 \tag{4.7}$$

$$u(x^*, y^*, z^*) = 1 \quad u_x(x^*) = 1 \quad u_y(y^*) = 1 \quad u_z(z^*) = 1 \tag{4.8}$$

Then, for consistency,

$$k_x + k_y + k_z = 1 \tag{4.9}$$

The single variable utility functions can be computed independently, and the k parameters are scaling constants that are generated from certainty and probabilistic considerations. The basic idea for assessing the scaling constants is to obtain a set of three (or n for n attributes) independent equations with three (or n) unknowns, which can then be solved to obtain the k's, and then normalizing them to sum to 1. With additive independence, we are can derive such equations:

$$k_i = u(x_i^*, \bar{x}_i^0) \tag{4.10}$$

$$\sum_i k_i = 1 \tag{4.11}$$

where $\bar{x}_i^0$ is the vector of all attributes except x_i at their values that yield minimum utility.

The additive utility function is valid if and only if the additive independence condition holds.

There are a number of different *multilinear utility functions* dependent on the various utility independence assumptions made. For example, if each of X, Y, and Z are mutually utility independent, then the utility function is

$$\begin{aligned} u(x,y,z) &= k_x u_x(x) + k_y u_y(y) + k_z u_z(z) \\ &\quad + k_{xy} k_x k_y u_x(x) u_y(y) + k_{xz} k_x k_z u_x(x) u_z(z) \\ &\quad + k_{yz} k_y k_z u_y(y) u_z(z) + k_{xyz} k_x k_y k_z u_x(x) u_y(y) u_z(z) \end{aligned} \tag{4.12}$$

In our case, where the attributes are the decision recommendations of the EAs, we use the additive independence form. The appropriateness of this additive independence assumption is dependent on the degree to which the preferences over the multiple attributes are captured by single attribute rather than joint utility functions. For the decisions of the EAs, this is a reasonable assumption because the particular decision of one EA does not markedly affect the preferences over the decisions of other EAs in normal circumstances. Returning to our example for two EAs, the utility function would be

$$u(d_j^{CA}, d_k^{AU}, \xi) = k_1 u_1(d_j^{CA}) + k_2 u_2(d_k^{AU}) + k_3 u_3(\xi) \tag{4.13}$$

k_3 and $u_3(\xi)$ do not have to be computed because that term is the same for any j and k.

The univariate utility functions on the right-hand side of Equation 4.13 are computed by the individual EAs. For instance, $u_1(d_j^{CA})$ is the utility calculated by the Condition-Action EA for d_j. We set the scaling constants, k_i's, to all be equal because of the lack of distinction between the decisions of each EA. In other words, from our example,

$$k_1 = u(d^{CA*}, d^{AU0}, \xi) = u(d^{AU*}, d^{CA0}, \xi) = k_2 \tag{4.14}$$

Probabilities The second component from Equation 4.5 needed to compute the utility of a decision at the metalevel is the probability $p(d_j^{CA}, d_k^{AU}, \xi | d_i^{ML}, t_c)$. This probability represents the degree of belief that the EAs will recommend certain decisions given computation time t_c and a particular decision of the metalevel. With four EA decisions, the metalevel decision, and t_c, this is a probability function in a 7-dimensional space. Like the multi-attribute utility function, we would like to simplify this function also by taking advantage of any independence properties.

For instance, if we assume conditional independence of the EA decisions given the metalevel decision and t_c, we can simplify the function to single EA probability functions:

$$p(d_j^{CA}, d_k^{AU}, \xi | d_i^{ML}, t_c) \;=\; p(d_j^{CA} | d_i^{ML}, t_c)\, p(d_k^{AU} | d_i^{ML}, t_c)\, p(\xi | d_i^{ML}, t_c) \tag{4.15}$$

The probabilities of the right-hand side of Eqn. 4.15 are conditional probabilities that the optimal decision choice D_{opt} is the same as the output of the metalevel or an EA given computation time ct. To simplify the probabilities, we assume that there are only two cases: the optimal decision is the same as the EA or ML decision *or* the optimal decision is not the same as the EA or ML decision. This allows the representation of the conditional probability using a two-dimensional graph as shown in Figure 4.7 giving the probability that the optimal decision is the same as the EA or ML decision (e.g., for $p(d_j^{CA}|d_i^{ML}, t_c)$, the $i = j$ case. For the alternative case (e.g., the $i \neq j$ case), the probability is normalized for a single decision value: $\frac{1}{|D|-1}(1 - p(d_j^{CA}|d_i^{ML}, t_c))$ where $|D|$ is the number of decision values.

Currently the probabilities are fixed, but a learning procedure to update the probabilities can be implemented because each decision of the metalevel with the associated decisions of the EAs can be used as a sample instance to the learning procedure.

In some sense, the probabilities of $p(d_j^{<ea>}|d_i^{ML}, t_c)$ can be viewed as "performance profiles" [18, 80] that give "quality" as a function of computation time. In this case, the "quality" of an EA is the probability that the decision recommendation of a particular execution architecture concurs with the optimal decision of the Metalevel. An example of the differences of performance profiles for EAs can be seen by comparing the Condition-Action and Decision-Theoretic EAs. Because of the restriction to $\{0, 1\}$ values, the Condition-Action EA should be able to use special-purpose, fast inference methods that yield an action recommendation faster than the Decision-Theoretic EA. However, the *quality* of the results of the Condition-Action EA may be less than the results of the Decision-Theoretic EA because of the same restriction—the Condition-Action EA loses information by not allowing uncertainty regarding

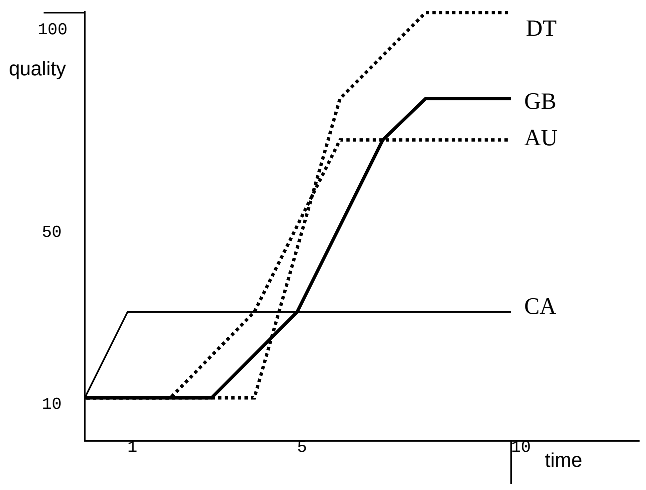


Figure 4.7: Example performance profiles for the four EAs for the AUV example.

condition variables.

Instead of assuming conditional independence of the EA decisions given the metalevel decision, we can consider other methods to compute the 7-dimensional probability function, $p(d_j^{CA}, d_k^{AU}, d_j^{GB}, d_k^{DT}, \xi | d_i^{ML}, t_c)$. For example, we can slightly relax the conditional independence requirement and compute using joint probability distributions (e.g., $p(d_j^{CA}, d_k^{AU}, | d_i^{ML}, t_c))$. Alternatively, we can dismiss altogether the approach of specifying the probability function, and train a neural network to output the probability value given inputs of the decisions. Or another approach is to use non-linear regression to find an appropriate functional form using samples of the decisions. However, as the system implemented, we use the conditional independence assumption and proceed as discussed above.

Examples To further understanding of how the results from the EAs are combined to make a final decision at the metalevel, we will examine two examples. The first example goes over the degenerate case when only one EA is available; the second

	$u_1(d_j^{CA})$	$u_2(d_k^{AU})$
d_1	1	0.2
d_2	0	0.8

Table 4.1: Utility estimates of the decisions as computed by the EAs

example will treat a two-EA situation using numerical data.

Suppose only one EA existed. Then Equation 4.5 would be as follows (using the Condition-Action EA as an example):

$$u(d_i^{ML}, t_c) \quad = \quad \sum_j u(d_j^{CA}, \xi) p(d_j^{CA}, \xi | d_i^{ML}, t_c) \tag{4.16}$$

The utility function $u(d_j^{CA}, \xi)$ would be obtained directly using $u_1(d_j^{CA})$ computed by the Condition-Action EA. The relative ranking of each decision would be the same for u and u_1. This is appropriate since there is no other information usable by the metalevel. The probability function $p(d_j^{CA}, \xi | d_i^{ML}, t_c)$ is directly obtained from $p(d_j^{CA} | d_i^{ML}, t_c)$, the information we have about the performance of that EA. Given the relationship of the utility functions, this probability will be 1 when $j = i$ and 0 otherwise. Therefore, when there is only one EA, the metalevel correctly mirrors the decision recommendations of the single EA.

As the second example, consider a numerical example to compute d^* using the Condition-Action and Action-Utility EAs where there are two possible decisions, d_1 and d_2 and a given computation time t_c. Let the utilities computed by the EAs be as in Table 4.1.

j \ k	1	2
1	0.6	0.9
2	0.1	0.4

Table 4.2: Multiattribute utilities of the outcome states

Equalizing and normalizing the scaling constants gives

$$k_1 = k_2 = 0.5 \tag{4.17}$$

The term $k_3u_3(\xi)$ is set to 0. With this information, the outcome state utility function can be generated using

$$u(d_j^{CA}, d_k^{AU}, \xi) \;=\; k_1u_1(d_j^{CA}) + k_2u_2(d_k^{AU}) + k_3u_3(\xi) \tag{4.18}$$

and the entries of Table 4.2 can be determined.

For the probability functions, we assume the following values from $t_c = 1$ to 4 as shown in Figure 4.8.

The value of $p(\xi|d_i^{ML}, t_c)$ is the same over all d_i^{ML} and so it is not relevant to the probability computation. The multiattribute probability values can be generated using

$$p(d_j^{CA},d_k^{AU},\xi|d_i^{ML},t_c) \;=\; p(d_j^{CA}|d_i^{ML}, t_c)\; p(d_k^{AU}|d_i^{ML}, t_c)\; p(\xi|d_i^{ML}, t_c) \tag{4.19}$$

The graph in Figure 4.9 displays the probabilities for d_1^{ML}.

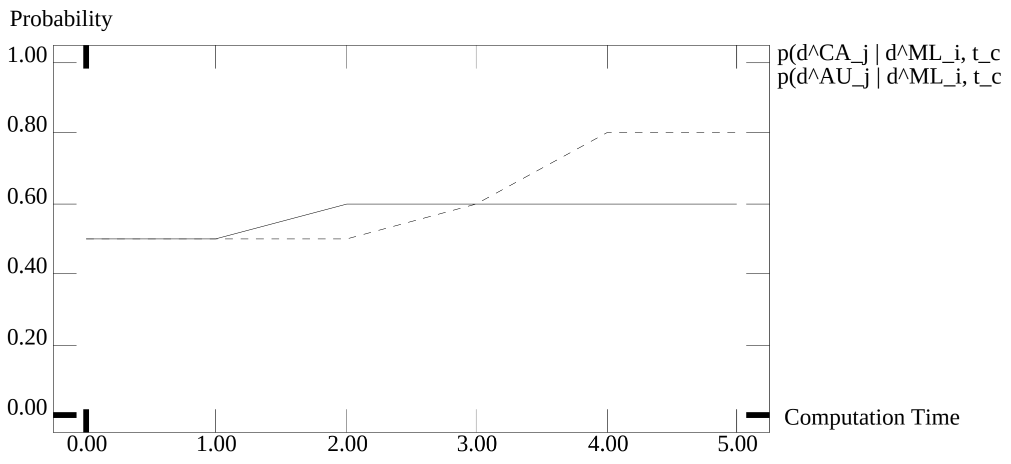


Figure 4.8: Performance probabilities for the two EAs

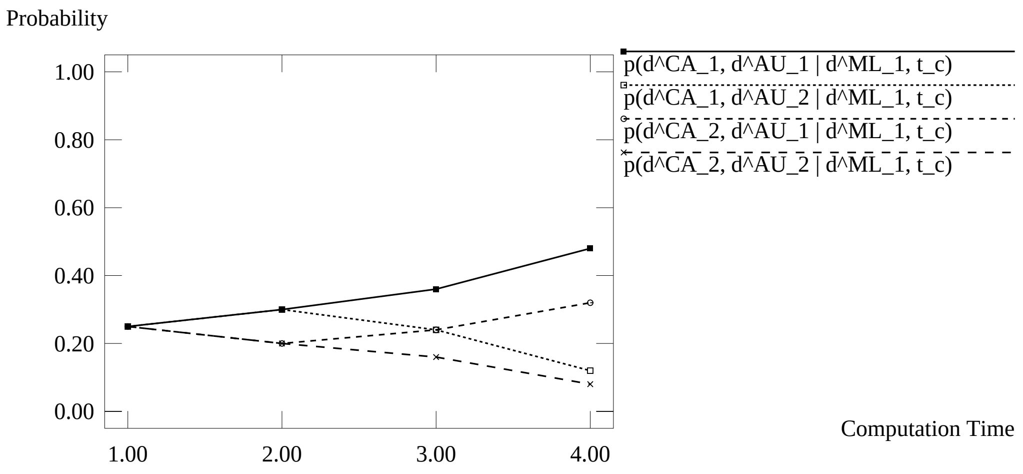


Figure 4.9: Graph of multiattribute probabilities for d_1^{ML}.

The values for both d_1^{ML} and d_2^{ML} are displayed in Table 4.3.

Finally, combining the multiattribute utilities of Table 4.2 and the probabilities of Table 4.3, we can compute the utilities of the decisions:

$$u(d_i^{ML}, t_c) = \sum_{j,k} u(d_j^{CA}, d_k^{AU}, \xi) p(d_j^{CA}, d_k^{AU}, \xi | d_i^{ML}, t_c) \quad (4.20)$$

t_c	1	2	3	4
$p(d_1^{CA}, d_1^{AU} \| d_1^{ML}, t_c)$	0.25	0.3	0.36	0.48
$p(d_1^{CA}, d_2^{AU} \| d_1^{ML}, t_c)$	0.25	0.3	0.24	0.12
$p(d_2^{CA}, d_1^{AU} \| d_1^{ML}, t_c)$	0.25	0.2	0.24	0.32
$p(d_2^{CA}, d_2^{AU} \| d_1^{ML}, t_c)$	0.25	0.2	0.16	0.08
$p(d_1^{CA}, d_1^{AU} \| d_2^{ML}, t_c)$	0.25	0.2	0.16	0.08
$p(d_1^{CA}, d_2^{AU} \| d_2^{ML}, t_c)$	0.25	0.2	0.24	0.32
$p(d_2^{CA}, d_1^{AU} \| d_2^{ML}, t_c)$	0.25	0.3	0.24	0.12
$p(d_2^{CA}, d_2^{AU} \| d_2^{ML}, t_c)$	0.25	0.3	0.36	0.48

Table 4.3: Multiattribute probabilities for numerical examples.

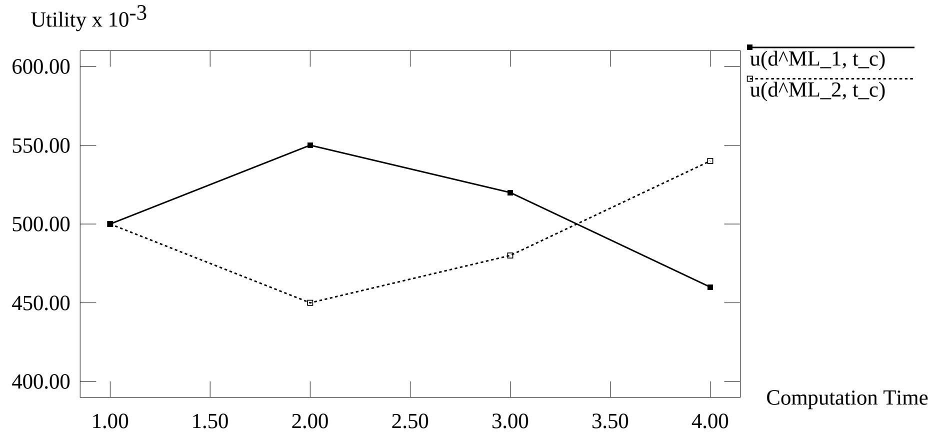


Figure 4.10: Utilities of decisions of the metalevel

From the results of Figure 4.10, we can see that at $t_c = 1$ there is no preference between the two decisions because the performance probabilities of the two EAs at that time are as likely to be wrong as right. At times $t_c = 2$ or 3, d_1^{ML} has the higher utility. d_1 was the decision that had the highest utility in the Condition-Action EA. At time $t_c = 4$, however, d_2^{ML} has the higher utility as the high probability that the Action-Utility EA gives the correct decision recommendation overwhelms the recommendation of the Condition-Action EA.

4.2 AUV Implementation

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 commercial software package [3] which operates only on *probabilistic belief networks* [73]. Therefore, the extended influence diagram is converted to a corresponding belief network as in [10] 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.

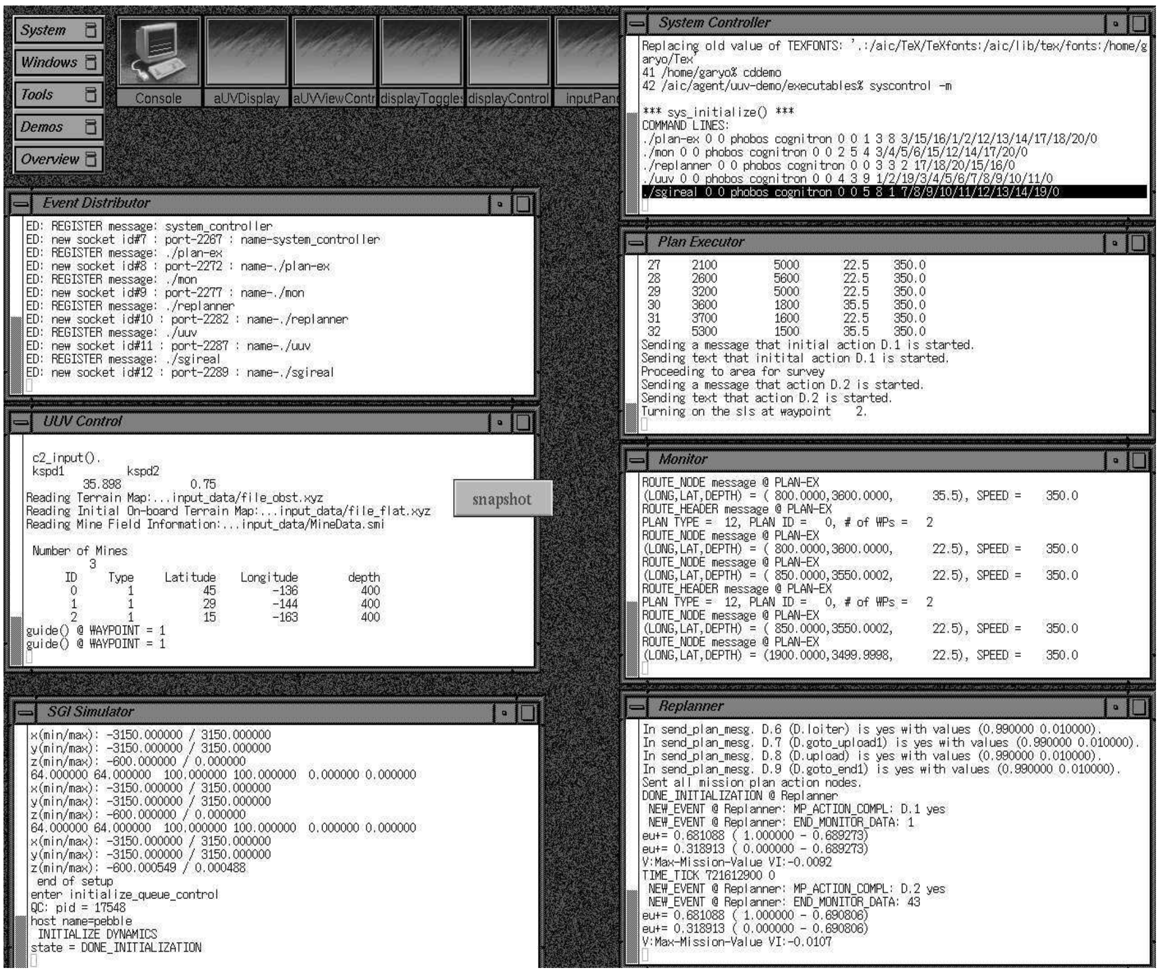
Since each EA has a different "performance profile" (decision quality as a function of computation time), in the multiple EAs implementation an interesting interaction that occurs between EAs as computation time varies can be observed. In our implementation, the Condition-Action performance profile quickly 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.

Implementation of the AUV example has been done by writing C code that can access the HUGIN belief network inference functions. The four EAs and the metalevel are implemented as separate parallel processes that communicate via message passing. One of the experiments that we have run is to change the weight of the time cost at the metalevel. As time cost rises, the metalevel makes quicker decisions and strongly weights the “fast” EAs.

Figure 4.11 shows a screen display of the AUV simulator. The simulator maintains a 3-dimensional terrain model. In addition to the simulator and the control system that will be discussed in Chapter 6, other processes that are running include a low-level path planner that generates the actual navigation commands, and a plan interpreter that converts high-level commands from the control system into “waypoints”, 3-D absolute location coordinates.

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. This example will be presented in more depth in Chapter 6.

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.



System Controller

```
Replacing old value of TEXFONTS: '.:/aic/TeX/TeXfonts:/aic/lib/tex/fonts:/home/g
aryo/Tex'
41 /home/garyo% cddemo
42 /aic/agent/uuv-demo/executables% syscontrol -m

*** sys_initialize() ***
COMMAND LINES:
./plan-ex 0 0 phobos cognitron 0 0 1 3 8 3/15/16/1/2/12/13/14/17/18/20/0
./mon 0 0 phobos cognitron 0 0 2 5 4 3/4/5/6/15/12/14/17/20/0
./replanner 0 0 phobos cognitron 0 0 3 3 2 17/18/20/15/16/0
./uuv 0 0 phobos cognitron 0 0 4 3 9 1/2/19/3/4/5/6/7/8/9/10/11/0
./sgireal 0 0 phobos cognitron 0 0 5 8 1 7/8/9/10/11/12/13/14/19/0
```

Event Distributor

```
ED: REGISTER message: system_controller
ED: new socket id#7 : port-2267 : name-system_controller
ED: REGISTER message: ./plan-ex
ED: new socket id#8 : port-2272 : name-./plan-ex
ED: REGISTER message: ./mon
ED: new socket id#9 : port-2277 : name-./mon
ED: REGISTER message: ./replanner
ED: new socket id#10 : port-2282 : name-./replanner
ED: REGISTER message: ./uuv
ED: new socket id#11 : port-2287 : name-./uuv
ED: REGISTER message: ./sgireal
ED: new socket id#12 : port-2289 : name-./sgireal
```

Plan Executor

```
 27     2100        5000      22.5    350.0
 28     2600        5600      22.5    350.0
 29     3200        5000      22.5    350.0
 30     3600        1800      35.5    350.0
 31     3700        1600      22.5    350.0
 32     5300        1500      35.5    350.0
Sending a message that initial action D.1 is started.
Sending text that initital action D.1 is started.
Proceeding to area for survey
Sending a message that action D.2 is started.
Sending text that action D.2 is started.
Turning on the sls at waypoint    2.
```

UUV Control

```

 c2_input().
 kspd1          kspd2
        35.898          0.75
Reading Terrain Map:...input_data/file_obst.xyz
Reading Initial On-board Terrain Map:...input_data/file_flat.xyz
Reading Mine Field Information:...input_data/MineData.smi

 Number of Mines
        3
     ID      Type     Latitude    Longitude       depth
      0         1           45         -136         400
      1         1           29         -144         400
      2         1           15         -163         400
guide() @ WAYPOINT = 1
guide() @ WAYPOINT = 1
```

Monitor

```
ROUTE_NODE message @ PLAN-EX
(LONG,LAT,DEPTH) = ( 800.0000,3600.0000,      35.5), SPEED =    350.0
ROUTE_HEADER message @ PLAN-EX
PLAN TYPE =  12, PLAN ID =   0, # of WPs =   2
ROUTE_NODE message @ PLAN-EX
(LONG,LAT,DEPTH) = ( 800.0000,3600.0000,      22.5), SPEED =    350.0
ROUTE_NODE message @ PLAN-EX
(LONG,LAT,DEPTH) = ( 850.0000,3550.0002,      22.5), SPEED =    350.0
ROUTE_HEADER message @ PLAN-EX
PLAN TYPE =  12, PLAN ID =   0, # of WPs =   2
ROUTE_NODE message @ PLAN-EX
(LONG,LAT,DEPTH) = ( 850.0000,3550.0002,      22.5), SPEED =    350.0
ROUTE_NODE message @ PLAN-EX
(LONG,LAT,DEPTH) = (1900.0000,3499.9998,      22.5), SPEED =    350.0
```

SGI Simulator

```
x(min/max): -3150.000000 / 3150.000000
y(min/max): -3150.000000 / 3150.000000
z(min/max): -600.000000 / 0.000000
64.000000 64.000000  100.000000 100.000000  0.000000 0.000000
x(min/max): -3150.000000 / 3150.000000
y(min/max): -3150.000000 / 3150.000000
z(min/max): -600.000000 / 0.000000
64.000000 64.000000  100.000000 100.000000  0.000000 0.000000
x(min/max): -3150.000000 / 3150.000000
y(min/max): -3150.000000 / 3150.000000
z(min/max): -600.000549 / 0.000488
 end of setup
enter initialize_queue_control
QC: pid = 17548
host name=pebble
 INITIALIZE DYNAMICS
state = DONE_INITIALIZATION
```

Replanner

```
In send_plan_mesg. D.6 (D.loiter) is yes with values (0.990000 0.010000).
In send_plan_mesg. D.7 (D.goto_upload1) is yes with values (0.990000 0.010000).
In send_plan_mesg. D.8 (D.upload) is yes with values (0.990000 0.010000).
In send_plan_mesg. D.9 (D.goto_end1) is yes with values (0.990000 0.010000).
Sent all mission plan action nodes.
DONE_INITIALIZATION @ Replanner
 NEW_EVENT @ Replanner: MP_ACTION_COMPL: D.1 yes
 NEW_EVENT @ Replanner: END_MONITOR_DATA: 1
eu+= 0.681088 ( 1.000000 - 0.689273)
eu+= 0.318913 ( 0.000000 - 0.689273)
V:Max-Mission-Value VI:-0.0092
TIME_TICK 721612900 0
 NEW_EVENT @ Replanner: MP_ACTION_COMPL: D.2 yes
 NEW_EVENT @ Replanner: END_MONITOR_DATA: 43
eu+= 0.681088 ( 1.000000 - 0.690806)
eu+= 0.318913 ( 0.000000 - 0.690806)
V:Max-Mission-Value VI:-0.0107
```

Figure 4.11: Screen view of the AUV simulator with multiple processes running in the various windows. The display is on a Silicon Graphics Iris workstation.

4.3 Discussion

One of the benefits of modelling uncertainty through the use of decision theory is that the assumptions required to implement the domain model become explicit. The probabilistic and utility independence assumptions can be explicitly stated. Then, we can consider the effectiveness of different independence assumptions for a given domain and use the corresponding probability and utility functions.

Another benefit of a decision-theoretic approach is that knowledge necessary to modify the control system's behavior is in the form of probabilities and utilities (e.g., the time cost of a domain). Learning can occur by updating the probabilities and utilities, and standard Bayesian updating methods can be used.

Chapter 5

Knowledge Compilation

Knowledge compilation is the conversion of declarative knowledge to compiled knowledge. Declarative knowledge (e.g., utilities $u(\omega)$ and probabilities $p(\omega|d)$ where ω is an outcome state and d is a decision value) is generally easier to acquire and learn because the knowledge is available and representable in modular pieces. However, inference with declarative knowledge is usually expensive because it is not specially designed to be efficiently used for inference. In contrast, compiled knowledge (e.g., IF ω THEN d rules) is generally difficult to acquire directly but is designed to be efficient in execution.

An agent can be programmed or learn with greater facility using declarative knowledge, but, for a real-time agent that knowledge needs to be compiled into more efficient forms. This scenario motivates our study of knowledge compilation.

This chapter investigates knowledge compilation, providing a definition of it within the Multiple EAs framework, showing how it is useful and how compilation is used and controlled.

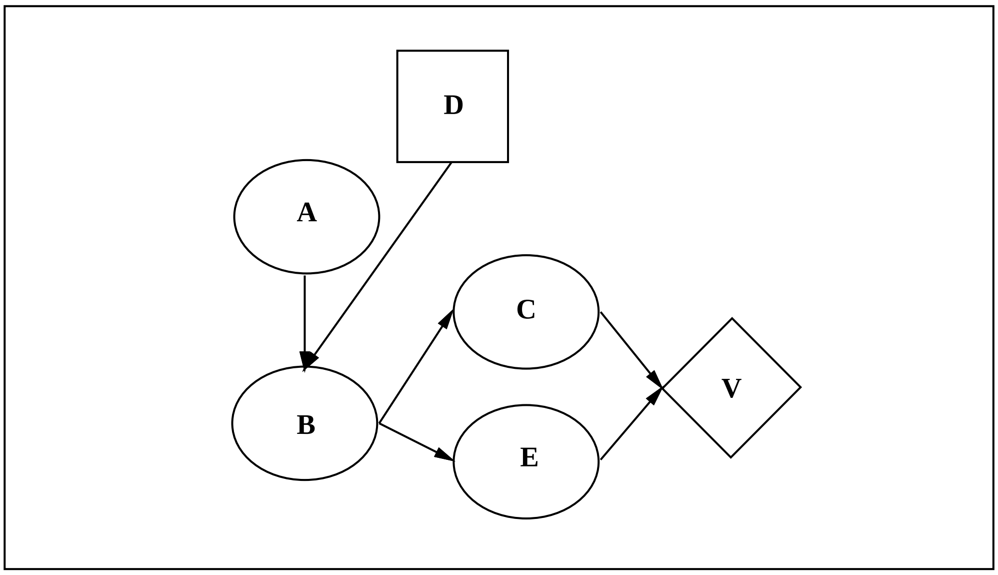


Figure 5.1: Sample influence diagram

5.1 The Advantages of Compilation

The standard goal of compilation is to obtain more efficient execution while maintaining behavioral equivalence to the uncompiled system. Generally, "efficient" in this case, means faster without using too much memory space. In the following, we will be more precise about the concepts of "efficient execution" and "behavioral equivalence".

5.1.1 Efficient execution

Clearly, the speed of the execution is dependent on what queries are presented and the particular algorithms used for inference. In a decision-theoretic representation, which arguably can subsume any other declarative representation, "execution" is inference to compute different probability distributions and utilities.

For example, given the influence diagram of Figure 5.1, we may want to compute

$u(A = a_1, D = d_5)$[1] or $p(C, E|A = a_2)$ or a number of other quantities. If we are often computing quantities where particular evidence (i.e., a subset of variables are known to have particular values, for example, $A = a_1, D = d_5$.) is common, it might be useful to save intermediate quantities. For example, suppose we are often computing $u(A, D)$ for different values of D. The full expression of this computation with the given influence diagram is

$$u(A, D) = \sum_C \sum_E \sum_B u(C, E)p(C|B)p(E|B)p(B|A, D) \tag{5.1}$$

An alternative way to compute the same quantity that would require fewer computations is to "compile" and save quantities such as $u(B)$ by first averaging over variables C and E:

$$u(B) = \sum_C \sum_E u(C, E)p(C|B)p(E|B) \tag{5.2}$$

and then using the intermediate quantity directly:

$$u(A, D) = \sum_B u(B)p(B|A, D) \tag{5.3}$$

The interesting/difficult aspect of the problem is that there are too many intermediate quantities to consider saving them all because of memory and management considerations. For every variable, we can consider particular instantiations (e.g., $A = a_1, B = b_3, ...$) and since there are an exponential number (in the number of vari-

[1]The utility u is computed at the value node V.

able values) of instantiations of the variable vector, there are at least that number of possible intermediate quantities to save. Because of the intractably large number, the choice of which compilations to do must be done intelligently.

Another aspect of the "execution" issue is the algorithms used to do inference. The above equations used multiplications and summations over probability and utility tables. These operations are the usual ones required to do inference in an influence diagram, but they can take different forms. For example, the use of parallel processors might be able to cut the complexity of the multiplications and summation. Another well-known alternative implementation is Lauritzen and Spiegelhalter's use of junction trees that allow local computations [56]. Dependent on the inference algorithm, compilation may be to different forms and have different values, but in a decision-theoretic representation, all compilation is to different probability or utility quantities.

5.1.2 Behavioral equivalence

If two systems are behaviorally equivalent, they each will select the same decisions given the same percept history. It is desirable to maintain behavioral equivalence as knowledge compilation is done. However, knowledge compilation will still be useful if the system can execute much faster and result in differences in only a small fraction of decisions. Depending on the utility model of the domain, the 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

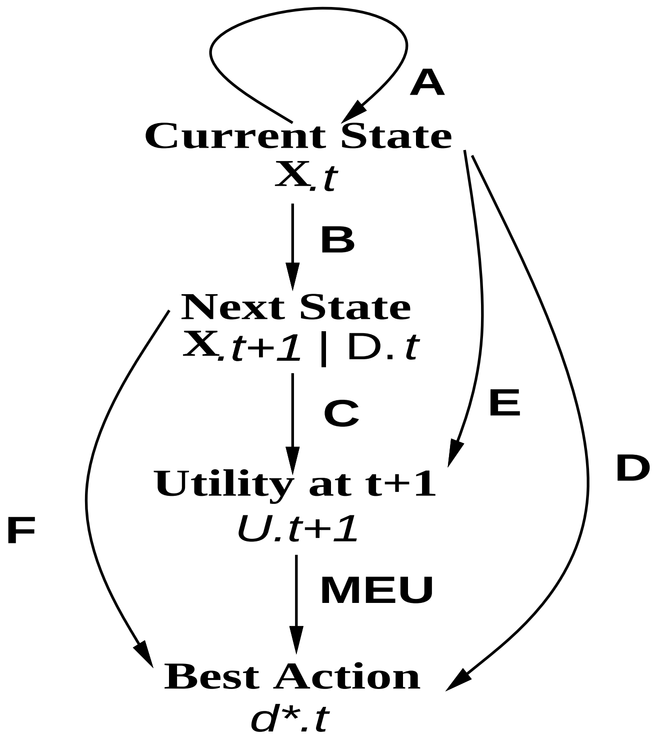


Figure 5.2: Knowledge forms for multiple execution architectures.

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.

We will mostly ignore the issue of approximation and work in the framework of behaviorally equivalent compilations in this chapter.

5.2 Defining Knowledge Compilation

Knowledge compilation can be defined precisely using the multiple execution architectures framework. Recall that the agent function $f : \mathbf{P}^* \rightarrow \mathbf{D}$ receives percepts

A+A ->A		
A+B ->B		
A+D ->D	**B+F ->D**	**E+ MEU ->D**
A+E ->E	**B+C ->E**	**C+ MEU ->F**

Table 5.1: Enumeration of all compilation routes. The left column shows the four types of homogeneous compilation. The middle column shows heterogeneous compilations involving projection. The right column shows heterogeneous compilations involving utility computation.

P as input from the environment and outputs a decision $d^* \in D$. There are multiple inference paths from P to d^* that are different implementations of f (Figure 5.2).

The various compilation routes can be identified by enumerating all possible combinations of arcs in Figure 5.2 that are adjacent, head to tail. Table 5.1 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 a piece of type B knowledge and a piece of type F knowledge can be combined to yield a piece of type D knowledge.

Russell [75] discussed the various compilation routes in general terms, independent of a particular knowledge representation. For example, Russell distinguishes between *homogeneous compilation* where the combined knowledge types are the same and *heterogeneous compilation* where the combined knowledge types are different.

With our choice of the decision-theoretic representation, there are two main 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, the compilation $A+A \rightarrow A$ is done by averaging out a probabilistic variable, in this case, E.

   $$p(A|B,C) \;=\; \sum_{E} p(A|E,C)p(E|B) \tag{5.4}$$

   A, B, C, E are all probabilistic variables in this example. The result is a reduced model with the probability distribution of E incorporated into the new distribution $p(A|B,C)$.

2. **Pre-computing expected utilities to fix policy.**

   $C + \mathrm{MEU} \rightarrow F$, $E + \mathrm{MEU} \rightarrow D$

   For example, $C + \mathrm{MEU} \rightarrow F$, In this case, 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.

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

   This is done by assigning $u_{GB}^{\top}$ to the outcome state configuration(s) of $\mathbf{X}.t1$ that maximize v_C. $u_{GB}^{\perp}$ is assigned to all other state configurations.

5.2.1 The eight compilation types

Each of the eight compilation types are defined in decision-theoretic terms as below. The figures for these compilations are shown for the affected fragments of the influence diagram. In general, the network may first require arc reversal and node elimination manipulations that will be discussed in Section 5.4.

1. $A + A \rightarrow A$

Knowledge of the current state, $p(X_i.t|\mathbf{X}.t)$ can be combined to get different pieces of knowledge of the current state, $p(X_j.t|\mathbf{X}.t)$. In the case below, the variable $DataAccrued.1$ is compiled out by averaging (or integrating) over its values. From the initial probability distributions of Type A, $p(DataAccrued.1|SensorCap.1)$ and $p(DataReceived.1|SensorState.1, DataAccrued.1)$, the compilation produces another probability distribution of Type A, $p(DataReceived.1|SensorState.1, SensorCap.1)$.

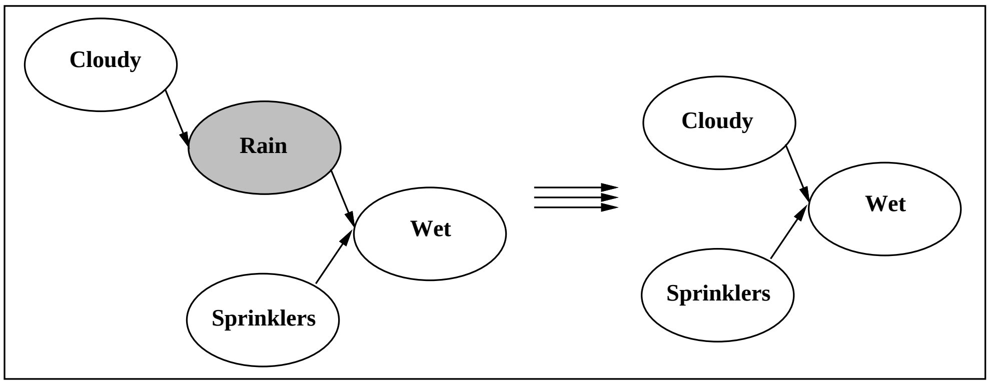


Figure 5.3: Example of $A + A \rightarrow A$ compilation.

2. $A + B \rightarrow B$

   The combination of Type A knowledge and Type B knowledge, $p(X_i.t{+}1|D.t, \mathbf{X}.t)$, can be combined to get new Type B knowledge. In the example, $p(DataAccrued.2|SensorCap.1, D.1)$ is generated by integrating out $DataAccrued.1$.

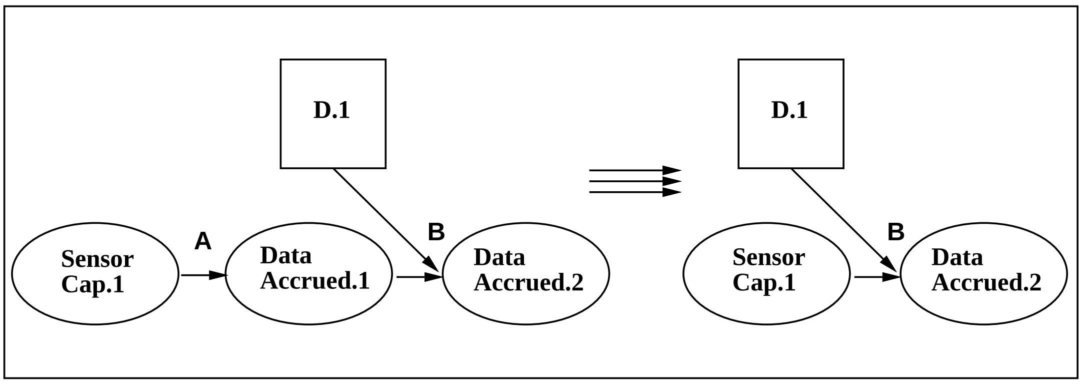


Figure 5.4: Example of $A + B \rightarrow B$ compilation

3. $A + D \rightarrow D$

   In this type of compilation, Type A knowledge is compiled out to get a more compact/efficient influence diagram using Type D knowledge, $v(U.t{+}1|D.t, \mathbf{X}.t) \in \{u^{\perp}_{CA}, u^{\top}_{CA}\}$.

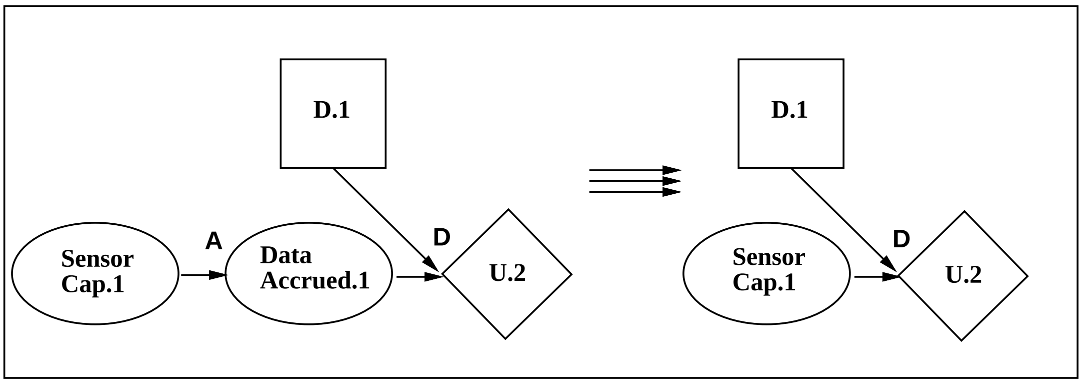


Figure 5.5: Example of $A + D \rightarrow D$ compilation

4. $A + E \rightarrow E$

   This type of compilation is similar to $A + D \rightarrow D$ compilation, except in this case the value function can be an arbitrary value in the utility range, indicating Type E knowledge, $v(U.t{+}1|D.t, \mathbf{X}.t) \in [u_{AU}^{\perp}, u_{AU}^{\top}]$.

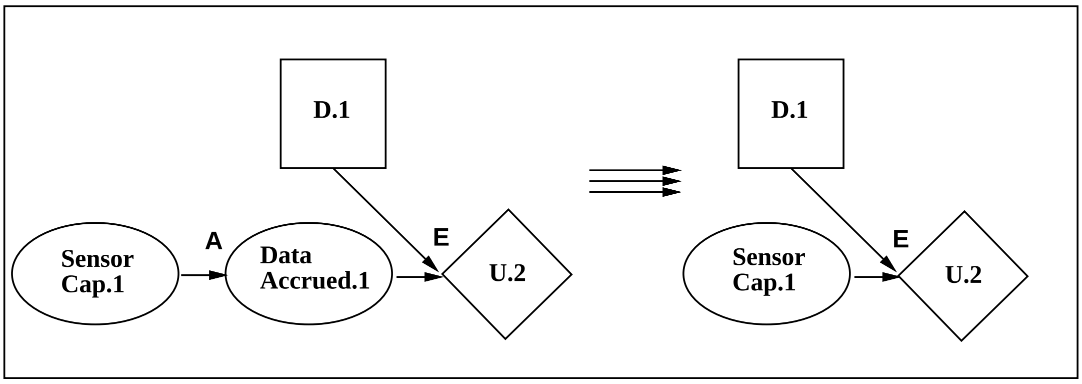


Figure 5.6: Example of $A + E \rightarrow E$ compilation

5. $B + C \rightarrow E$

   The combination of Type B and Type C knowledge, $v(U.t{+}1|\mathbf{X}.t{+}1) \in [u_{AU}^{\perp}, u_{AU}^{\top}]$, can be converted to Type E knowledge by eliminating a hidden variable at the time $t{+}1$.

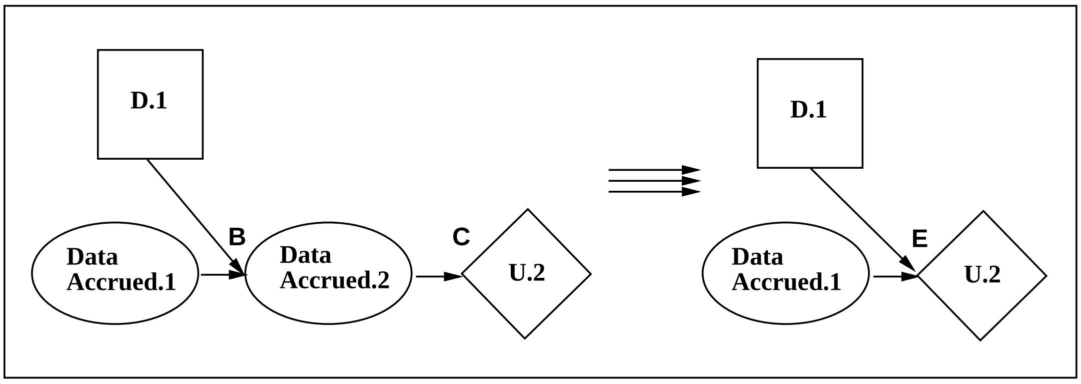


Figure 5.7: Example of $B + C \rightarrow E$ compilation

6. $B + F \rightarrow D$

This is the analogue to the $B + C \rightarrow E$ compilation where the utility values are restricted to two possible values by the Type F knowledge, $v(U.t{+}1|\mathbf{X}.t{+}1) \in \{u^{\perp}_{CA}, u^{\top}_{CA}\}$.

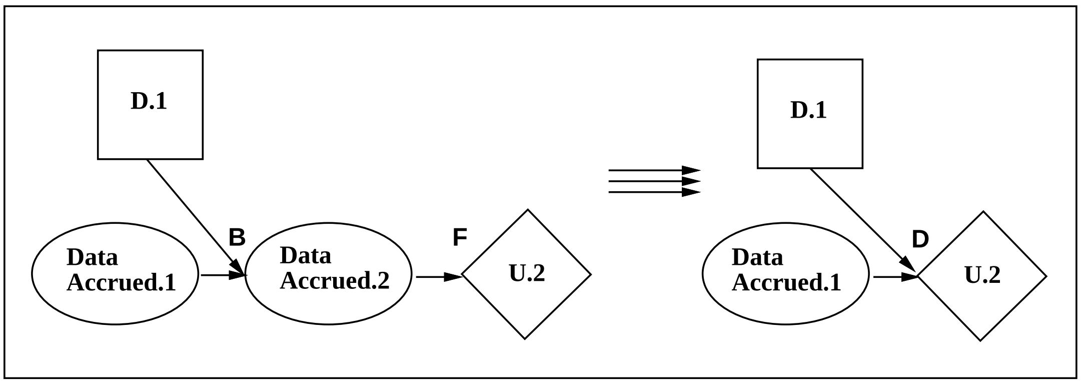


Figure 5.8: Example of $B + F \rightarrow D$ compilation

7. $C + MEU \rightarrow F$

In the next two compilation types, the Maximum Expected Utility principle is used to compile decision policies. In this case, there is no decision variable.

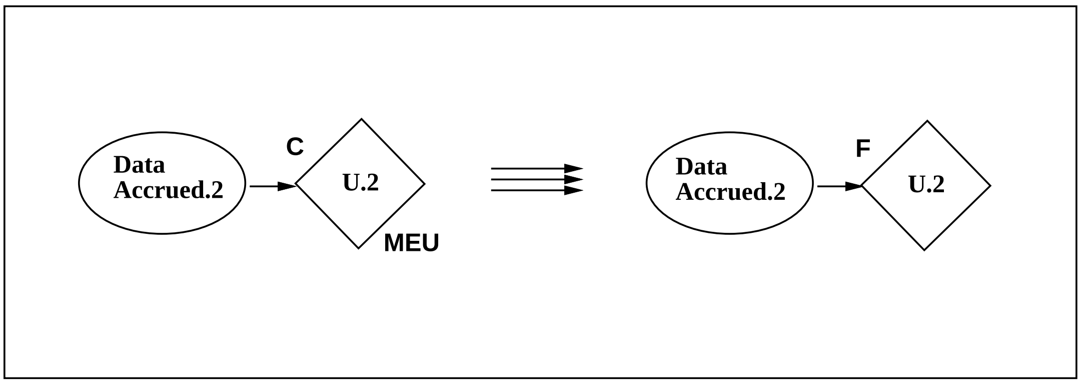


Figure 5.9: Example of $C + MEU \rightarrow F$ compilation

8. $E + MEU \rightarrow D$

In this example, Type E knowledge of utilities is compared using the MEU principle to generate fixed decision policies using Type D knowledge.

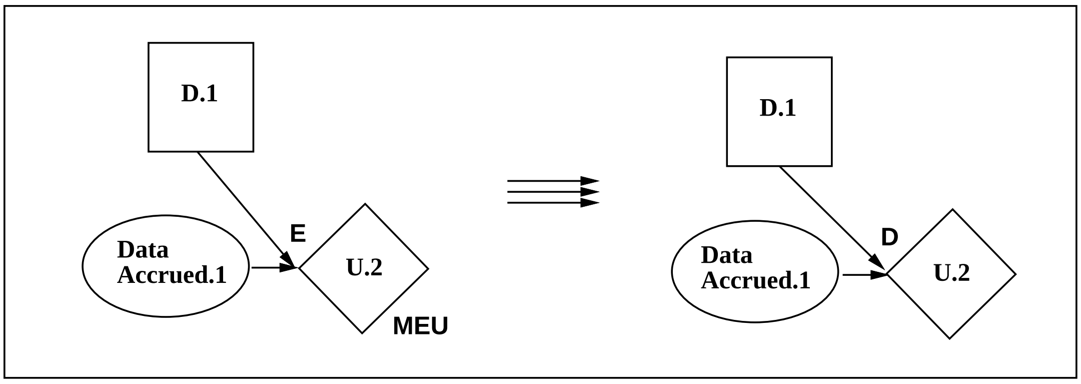


Figure 5.10: Example of $E + MEU \rightarrow D$ compilation

5.3 Compilation of Execution Architectures

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). Figure 5.11 shows the use of $B + C \rightarrow E$ compilation to convert a Decision-Theoretic EA to an Action-Utility EA, and then the use of $E + MEU \rightarrow D$ compilation to yield a Condition-Action EA.

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 5.12 shows the class of compilations that convert EAs.

The compilation ordering is not unique. A Decision-Theoretic EA could also be converted to a Goal-Based EA using $C + MEU \rightarrow F$, and then to a Condition-Action

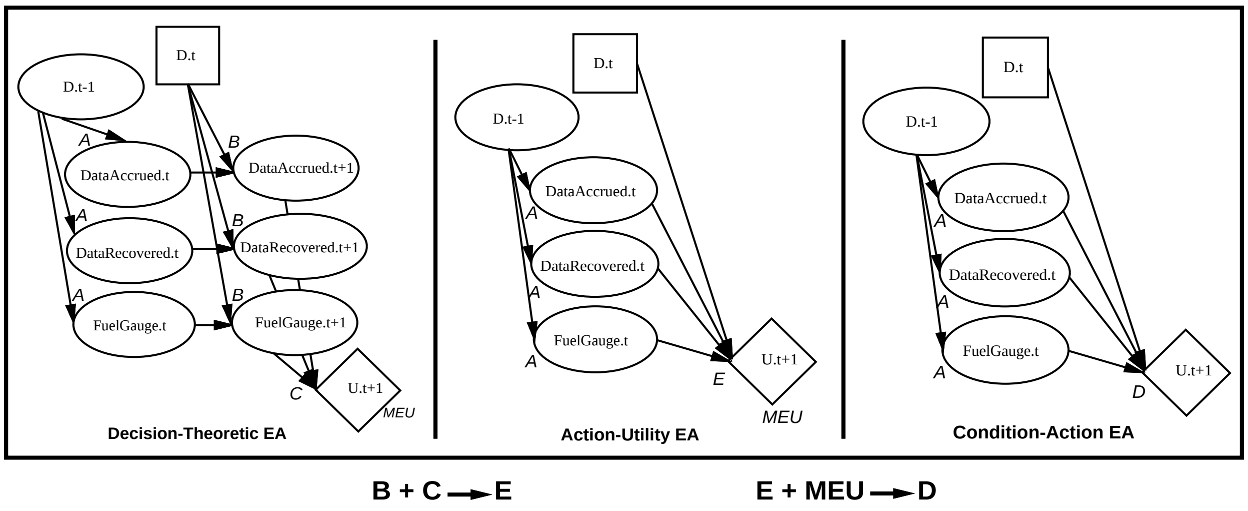


Figure 5.11: Compilation of Decision-Theoretic EA to a Action-Utility EA to a Condition-Action EA

EA using $B + F \rightarrow D$.

5.4 A Compilation Algorithm

This section describes a general knowledge compilation example. From an original, Decision-Theoretic EA, a compiled, Condition-Action EA is generated that retains behavioral equivalence and allows faster decision-making.

The strategy is to implement the following three steps.

1. Represent the prior probabilities as variables.

   By using variables 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.

2. Propagate the variables through influence diagram reductions.

   By applying Bayes Rule and marginalization, nodes of the influence diagram

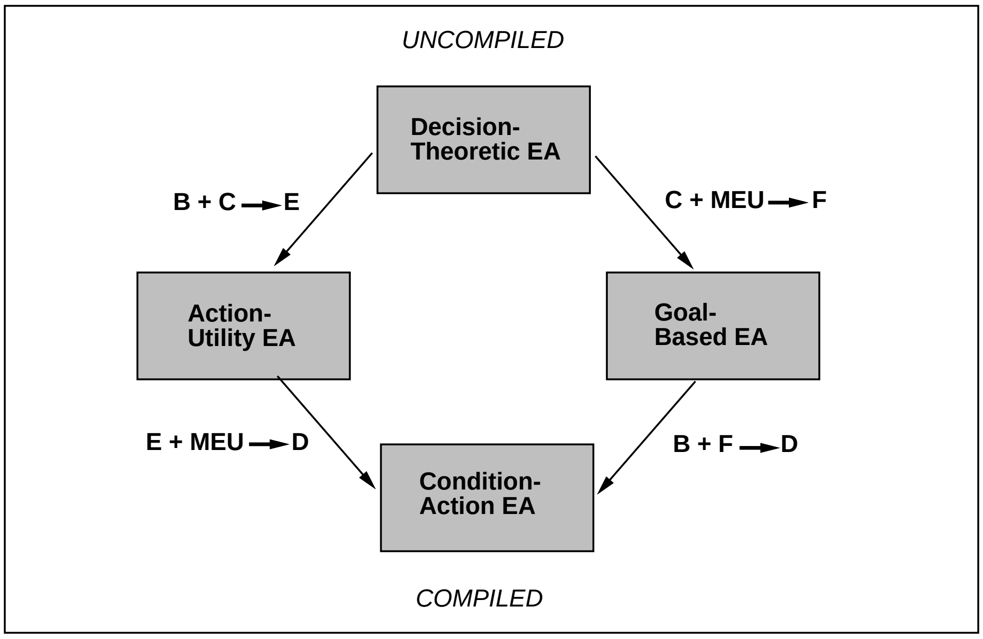


Figure 5.12: Compilations to convert execution architectures.

can be removed by integrating their influences into the rest of the network.

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.

The algorithm to convert a Decision-Theoretic EA to a Condition-Action EA uses the following four steps:

1. 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 nodes that might be used for evidence or the value node cannot be removed because these nodes might need to be examined.

2. Remove nodes of each time slice in order.

   The 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. Convert utilities to fixed decision policy.

   Once all chance variables except the possible evidence 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.

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

5.4.1 Example

An example of the general compilation strategy is given in this section. The domain of this example is an AUV in a mine mapping mission.

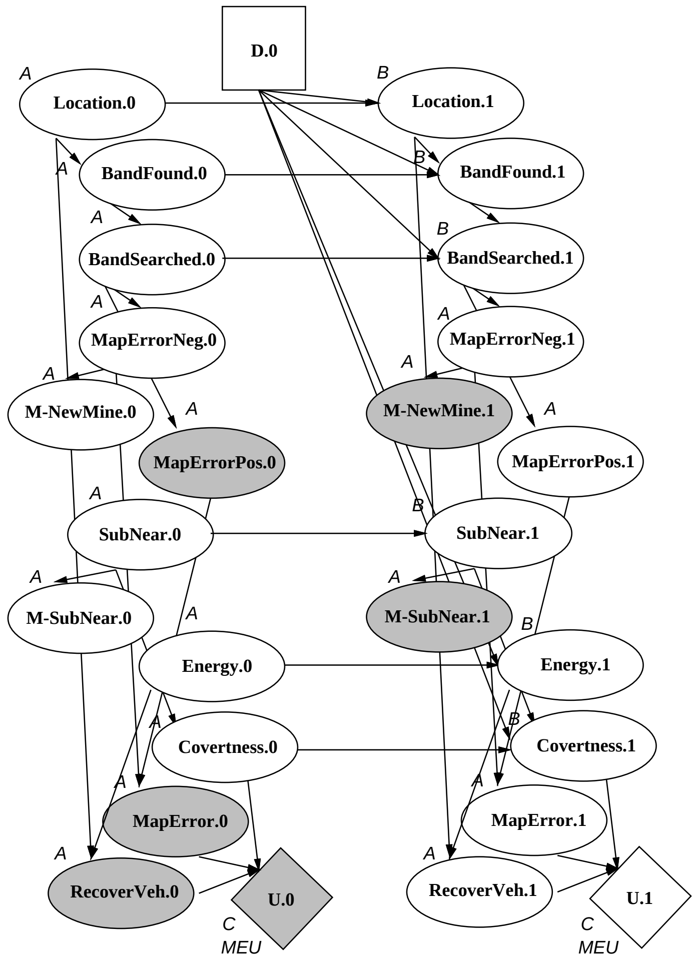


Figure 5.13: The original influence diagram is shown. Nodes that are highlighted will be removed by Barren node removal.

Barren Node Removal

Figure 5.13 shows the initial structure of the first two time slices of an extended influence diagram. The highlighted nodes will be removed by Barren node removal. Evidence from the sensors is available at the $M-NewMine.0$ and $M-SubNear.0$ nodes, that indicate whether a previously undetected mine has been found and whether a nearby sub has been detected.

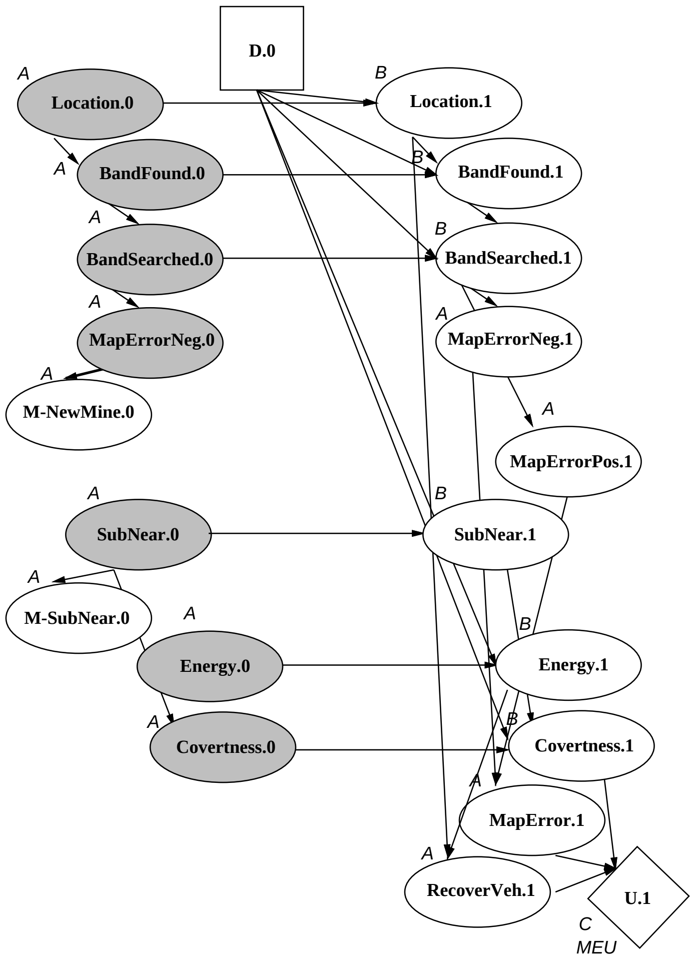


Figure 5.14: Highlighted nodes of Time Slice 0 will be removed.

Probabilistic Node Removal

Probabilistic node removal is done by application of Bayes Rule to reverse arcs and create barren nodes. The barren nodes can then be removed. Figure 5.14 shows the nodes of Time Slice 0 that will be removed. Figure 5.15 shows the nodes of Time Slice 1 that will be removed.

For this network, the nodes $Location.0$, $SubNear.0$, and $Energy.0$ have no parents and are therefore represented using prior probabilities. The priors for these binaryvalued nodes can be represented using three variables: p_l, p_s, p_e, that will be carried through the algebraic computations of knowledge compilation.

As an example of a node removal, the $SubNear.0$ node of Figure 5.14 requires the reversal of three arcs using Bayes Rule:

1. Reverse arc to $MSubNear.0$.

$$p(SN|M) = \frac{p(M|SN)p(SN)}{\sum_{SN} p(M|SN)p(SN)}$$

2. Reverse arc to $Covertness.0$.

$$p(SN|C, M) = \frac{p(C|SN, M)p(SN|M)}{\sum_{A} p(C|SN)p(SN|M)}$$

3. Reverse arc to $SubNear.1$.

The node $SubNear.0$ now has no children and can be removed.

Action-Utility EA compilation

Figure 5.16 shows the most reduced Decision-Theoretic EA. Note that the knowledge types used are of Types A,B, and C. And the Maximum Expected Utility (MEU) principle needs to be applied to the utilities of Type C.

Figure 5.17 shows the Action-Utility EA generated from the previous influence diagram using the $B + C \rightarrow E$ compilation.

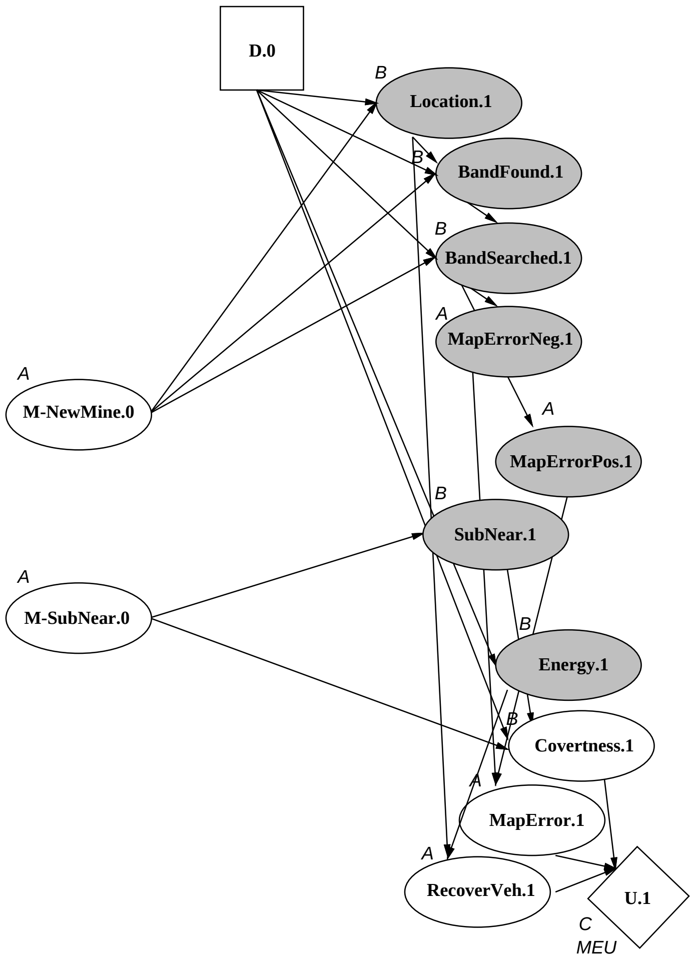


Figure 5.15: Highlighted nodes of Time Slice 1 will be removed.

Condition-Action EA compilation

Finally, the Type E utilities are converted to best decision choices directly using the $E + \text{MEU} \rightarrow D$ compilation. The Condition-Action EA influence diagram is shown in Figure 5.18.

Type E knowledge gives utilities of state-action pairs as shown in Table 5.2. In this case, the state corresponds to the evidence vector $M - NewMine.0$ and $M - SubNear.0$, and the possible decisions are $NOOP, WS, BS, GO$. The utilities are

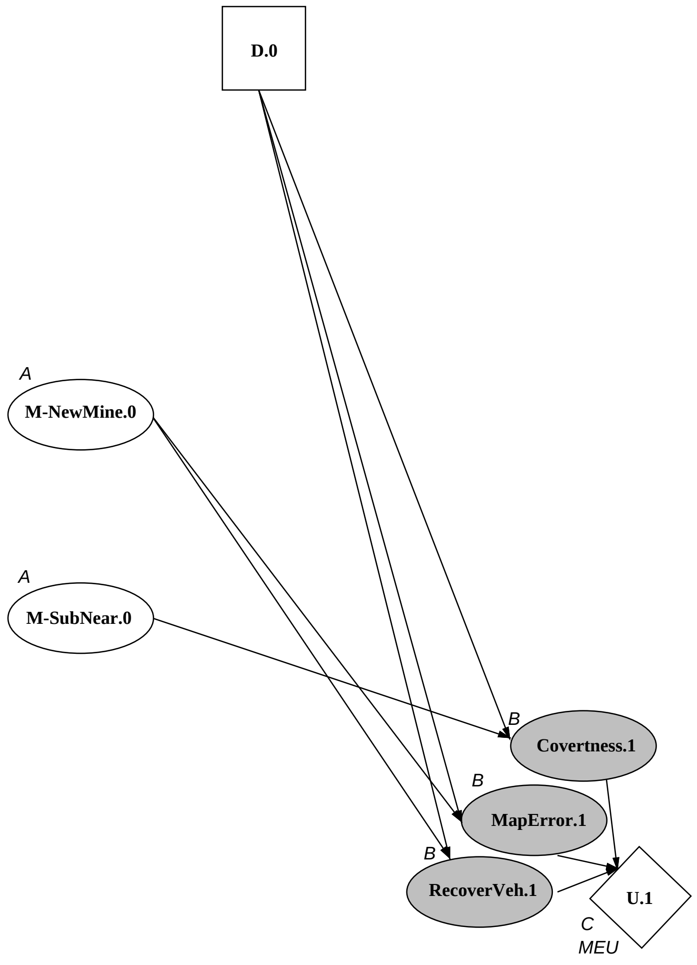


Figure 5.16: The reduced Decision-Theoretic EA.

represented as functions $f_i(p_l, p_s, p_e)$ of the prior probabilities.

NM	SN	$NOOP$	WS	BS	GO
F	F	f_1	f_2	f_3	f_4
F	T	f_5	f_6	f_7	f_8
T	F	f_9	f_{10}	f_{11}	f_{12}
T	T	f_{13}	f_{14}	f_{15}	f_{16}

Table 5.2: Type E knowledge: $f_i(p_l, p_s, p_e)$

From the Type E utility functions, If-Then rules can be generated in terms of the

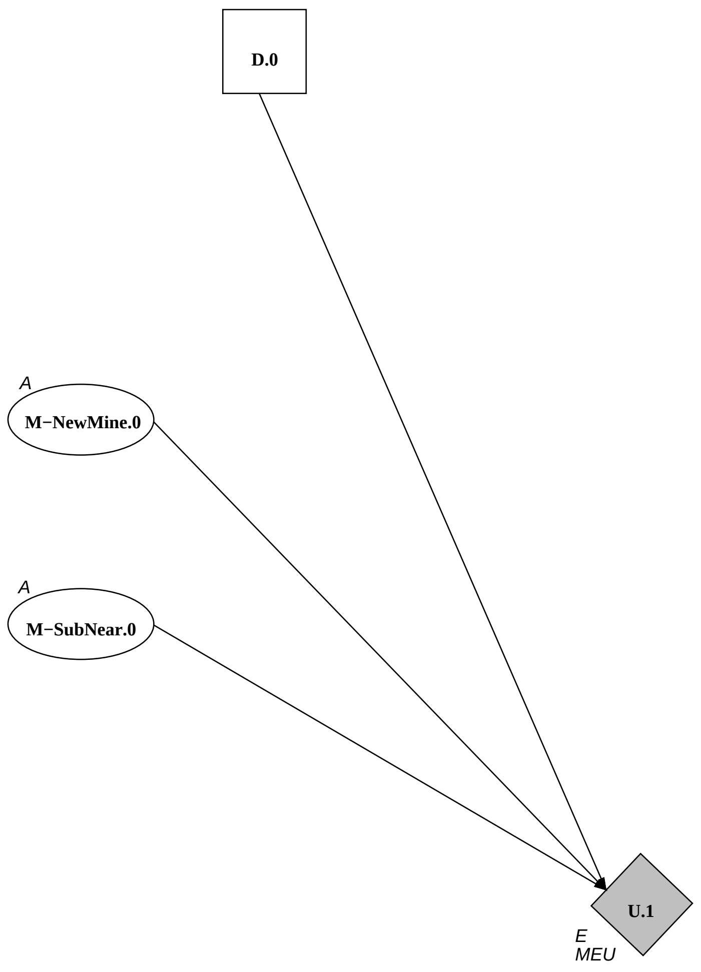


Figure 5.17: Action-Utility EA after $B + C \rightarrow E$ compilation.

prior probability variables. For example, a rule set might include the following rules:

$If((MNM = F)(MSN = F)(p_l \geq 1.4p_e))$

$(D = WSEARCH)$

$If((MNM = F)(MSN = F)(p_l < 1.4p_e))$

$(D = BSEARCH)$

$\vdots$

The actual implementation of how the If-Then rules will be used is unstated. For example, they might be used by a logic-based theorem prover, or by a decision tree

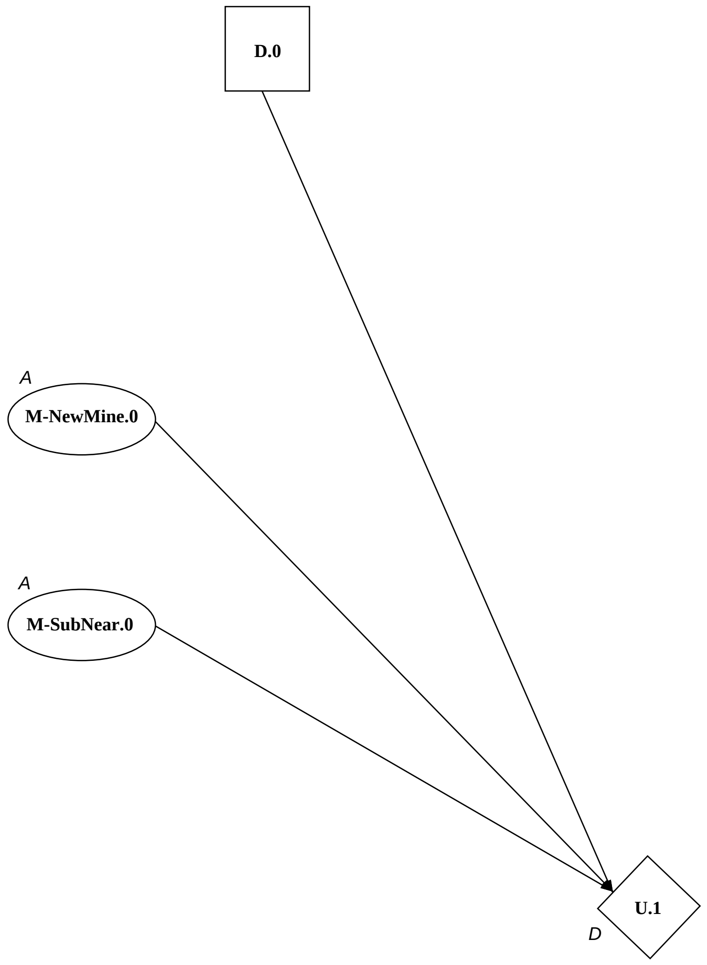


Figure 5.18: Condition-Action EA.

which branches on the condition values and whose leaves are the decision recommendations.

5.5 Discussion

5.5.1 Controlling compilation

Since in our framework the knowledge compilation is maintaining behavioral equivalence, 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 Chapter 6. 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. In general, some representation of the original uncompiled network must be maintained if any variable can be instantiated with evidence in the future. However, the types of compilation that do not remove nodes but precompute decision policy can always be done. Also, any non-evidence nodes can always be safely compiled out of the influence diagram.

5.5.2 Other issues

There are several important issues that remain for now future work.

Approximate compilation There are many ways to relax the behavioral equivalence condition and use approximate compilation. For example, as in D'Ambrosio [16] for probability computation, only a subset of the probability values of a distribution can be used to generate the new distribution.

Structural changes of model The knowledge compilation discussed assumes that the original influence diagram is correct and complete. When the model requires changes, the unaffected portions of the compiled network should remain constant.

The next chapter will continue the story of knowledge compilation by showing how it relates to the key concept of multiple execution architectures.

5.6 Review

This is a natural stopping point to review what has been discussed heretofore in this dissertation.

The multiple EAs framework was defined as a decision-theoretic extension to standard influence diagrams where six knowledge types are used. Knowledge compilation transformations were defined to convert knowledge types to a more efficient form. Certain combinations of these knowledge types determine four execution architectures which partition the space of possible ways to choose a best decision, d^*. The four execution architectures can be run in parallel, each computing on the same decision, and a Metalevel uses decision-theoretic principles to combine the information from four different implementations of the agent function to output a final action choice.

The next chapter will discuss decision-making and planning in the multiple EAs framework.

Chapter 6

Planning and MEA

In Chapter 3, we introduced the use of planning and planning windows within the EA computation step. This chapter discusses the planning representation and algorithm in depth. The main result is a metalevel control algorithm for planning that focuses on the most promising areas to replan and considers the time cost of replanning. In addition, a view of decision-theoretic planning is proposed that applies the power of decision theory (most notably, the ability to deal with uncertainty and multiple objectives) to the planning problem. Within this framework, we show how a hierarchical abstraction model of the domain model can be exploited rationally.

We show that dynamic replanning using the combination of decision-theoretic metareasoning and abstraction is formally sound and enables successful, real-time performance in a complex domain.

6.1 Introduction

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 Maximum Expected Utility principle can be applied with exact probability and utility functions for the action outcomes. However, as the domain becomes more complex, subsequent action sequences need to considered in order to calculate the utility of the outcomes of immediate actions. For example, the value of the action of buying a baseball game ticket now is dependent on future actions (e.g., whether I go to the game or not).

Many tasks are sufficiently recurrent and well-defined so that pre-existing plans can be used. There may be no need to plan an entire mission, but there is always a need to respond to events that occur during plan execution. We propose the use of *replanning* which repairs or modifies an existing plan rather than starting from scratch. Replanning is really a generalization of planning if we allow the "existing" plan to be empty or random. Two of the approaches to the replanning problem are to have a "reactive" subsystem that can default to a safe behavior [7, 25] or to provide a universal plan [83] or conditional plan [99, 74] before execution to deal with expected dynamic events. However, both approaches lack the ability to reason *during* execution to repair the plan. Dynamic reasoning (i.e., replanning) is preferable because it obviates the need to predict all events prior to execution and also because knowledge gained during execution can be used. Most of the work to date on replanning has been in "transformational" planning where faulty plans are projected into a library of domain-independent faults and an appropriate repair strategy is invoked [90, 31].

This approach is limited by its fixed number of pre-identified faults and strategies.

Our approach to replanning is to use decision-theoretic metareasoning: replanning is invoked if and only if the utility of replanning is greater than the utility of executing the plan, and the plan is repaired by fixing the part that would yield the highest utility. The essence of the strategy is simple— Do the action with the highest expected utility. This approach gives great flexibility in the type of replanning. But the inclusion of *computational* planning actions along with base-level actions (i.e., actions that directly affect the world) further complicates the design and implementation of decision-theoretic control systems and requires design choices on the method of metareasoning[1].

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 is selected, and it may be a base-level or computational action. The approach presented here calculates a "value of information" [45] of doing various planning actions, selecting the planning or base-level action that has the highest utility.

The key question is what the "various planning actions" are. As examples, the planning action may be one node expansion in a search of the plan space tree, or it may be one CPU second of computation of a given planning algorithm. The issue of selection of the type of planning action is largely independent of the decision-theoretic metacontrol method since any computational action can be characterized in decision-theoretic terms. However, the selection of the planning actions deeply affects the

[1]For some other decision-theoretic approaches see [42, 78, 77, 17, 21, 30, 39].

performance of the planning algorithm. The planning steps of the algorithm presented here are various evaluations and modifications of the plan window influence diagram to repair or modify the existing plan.

One of the problems in applying decision theory as well as other declarative approaches is that the complexity of the state space is prohibitively large. And one of the ways to combat high complexity has been to use a hierarchy of abstraction spaces which has been most prominently used in traditional planning (e.g., [81, 53]), and to a lesser extent with a decision-theoretic world view (e.g., [33, 102]). We, too, use a hierarchical model to restrict the size of the state space that needs to be evaluated. As a result of the decision-theoretic control of planning, the hierarchical model is exploited in a rational manner, examining different levels of detail depending on the estimated utility of doing so.

The replanning architecture has three main components— *plan impact analysis*: analysis of the impact on the current plan of monitor data; *plan modification focusing*: a value of computation focus mechanism to restrict the search area that needs replanning; and *plan modification*: the replanning algorithm itself which makes changes to the plan. We demonstrate the architecture with a simplified example of an underwater vehicle whose battery has failed.

6.2 Autonomous underwater vehicle domain

A typical mission for an autonomous underwater vehicle (AUV) involves launching from a surface ship, surveying a region, laying a sensor array, waiting for the sensor data to accrue, picking up sensor data, and returning to a recovery point (Figure 6.1).

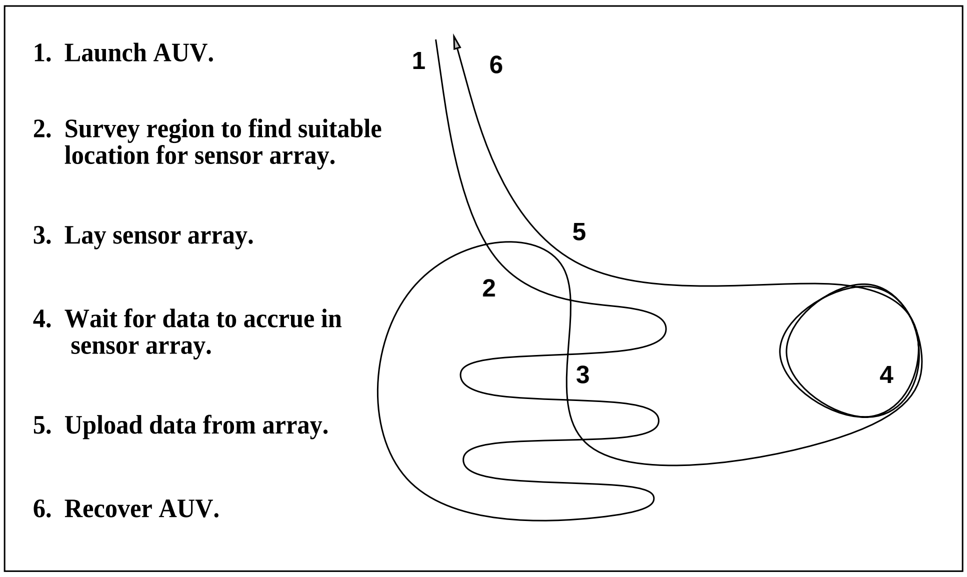

1. Launch AUV.
2. Survey region to find suitable location for sensor array.
3. Lay sensor array.
4. Wait for data to accrue in sensor array.
5. Upload data from array.
6. Recover AUV.
1
6
5
2
3
4

Figure 6.1: A typical AUV mission

In such a domain, the components of the plans are re-used many times, and canned strategies for different survey and other techniques are available. A nominal mission plan can be developed by a human mission planner for the current environment without too much difficulty. The void in capability is not in generating the nominal plan, but in providing autonomy to enable the AUV to operate effectively once launched. This is why there is a need for on-board replanning capability.

6.3 Representation

The *planning window* is the influence diagram currently being considered that can encompass an action sequence rather than just a single action. Assuming the Markov property of the states, we can repeatedly reproduce the same a single decision template that shows the relationship between two time slices[2] Using this automatic up-

[2]As was described in Chapter 3.

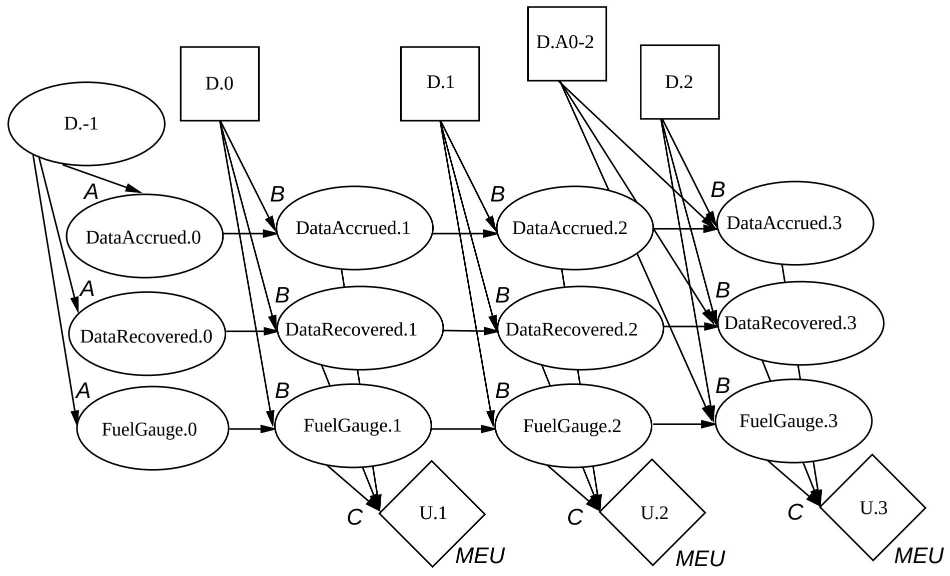


Figure 6.2: A Decision-Theoretic EA influence diagram with a 3-step planning window.

dating, the world state is projected forward in time using only local connections. For k-step planning, the influence diagram is extended by chaining k decision templates together.

Figure 6.2 shows a 3-step planning window obtained by chaining together three decision templates of the Decision-Theoretic EA in the AUV example. Figure 6.3 shows two time slices of a network used for the AUV example.

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. For the EAs that do have temporal projection, as time passes and actions are executed, the planning window is "rolled" forward in time so that a k-step lookahead horizon is always extant. We have developed software to dynamically update a network and interface to the HUGIN software functions. The

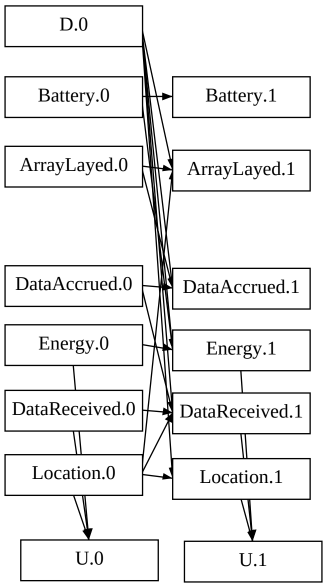


Figure 6.3: A single time slice of a network used for the AUV example. This figure was generated by the HUGIN belief network software which draws all nodes as rectangles.

dynamic network software is similar to Kjaerulff's dHUGIN (Dynamic HUGIN) software package [52] that also provides functionality to advance a HUGIN belief network forward in time. However, whereas dHUGIN provides more generality in the way a network can be moved forward or backward in time, our system is only able to move one time step forward at a time but is able to do this operation very quickly.

Planning in a decision-theoretic framework is somewhat different from the traditional notion of planning for goal satisfaction (e.g., [2]). The utility model of the influence diagram specifies preferences on outcome states, and 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., [22, 101]).

Another important difference from some work in classical planning is that the decision-theoretic approach is selecting decisions/actions that are perfectly controllable[3] and take no time to execute, in contrast to actions that cannot be perfectly controllable in general and may have duration. The decision can be viewed as a switch or dial which can always be set as desired. Of course, the *effects* of such a decision are not controllable. I can make the *decision* to lift 225 pounds, but the effect may be that I am unsuccessful. In contrast, the original STRIPS representation has action descriptions like "pickup" which cannot be executed without the requisite preconditions holding [24].

The objective in the decision-theoretic framework is to select the decision/action *sequence* that has the highest expected utility, and then to execute the first action of that sequence. The obvious method to select the optimal action sequence is to compute the expected utility of each action sequence and select the one with maximum expected utility. However, evaluating action sequences rather than single actions is more expensive by a factor at least exponential in the number of actions. This complexity makes decision-theoretic analysis even more susceptible to computational intractability, thereby emphasizing the necessity for metalevel control to intelligently direct the planning computation.

6.3.1 Abstraction

The abstract decision, `D.A0-2`, of Figure 6.2 shows one type of abstraction employed. An abstract decision `D.Ai-j` is composed of lower-level decisions executed from time i to time j. The values of an abstract decision node are pre-specified set-

[3]In other words, once a decision is selected, there is no uncertainty that it will be executed.

tings that control the decision nodes of its lower-level actions. An abstract decision value may be to omit all lower-level actions or to executed the lower-level actions in a pre-specified way. For example, an abstract decision node could have explicit values for different types of surveys (e.g., `ladder-survey`, `spiral-survey`, `z-survey`) specifying the move commands of lower-level actions. Using fixed instantiations provides the efficiency gains of reasoning at an abstract level while allowing some degree of flexibility at a more specific level. In some cases, reasoning at the abstract level will be too restrictive and a more specific level of action description is necessary. The choice of which level of abstraction to replan at is made using decision-theoretic principles that estimates the utilities of replanning at different levels of abstraction.

6.4 Architecture

The architecture of the replanning system is shown in Figure 6.4. The replanning system is interfaced to a low-level executor/monitor that executes steps of the plan and provides monitor data from the environment. The first module is *Plan Impact Analysis* which monitors the current situation data to detect when replanning has positive value. If replanning is suggested, a message is passed to the second module, *Plan Modification Focusing*, which focuses the search area for the replanner, passing an ordered list of candidate areas in which to replan. If the value of replanning is negative, a message is sent to the Executor to execute the next plan step. The third module, *Plan Modification*, does the actual plan modification and can request further narrowing of the replan area from the Focusing module.

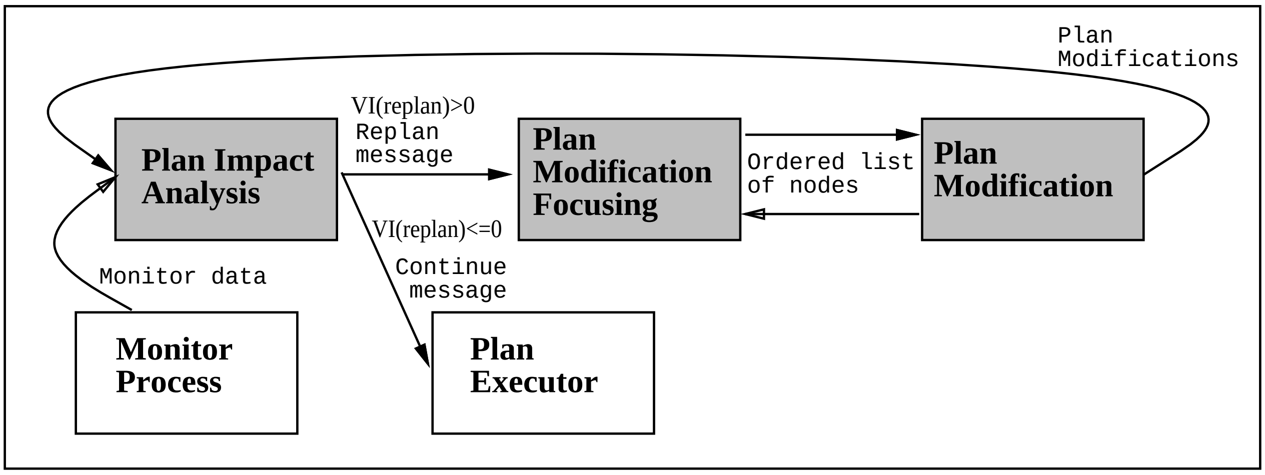


Figure 6.4: Architecture of the replanning and low-level executor/monitor systems.

6.4.1 Plan impact analysis

The Plan Impact Analysis module updates the belief network using evidence from the Monitor process. In the belief network, there is a node for every monitor datum of interest. In the AUV example, the monitor data is information from the sensors about the underwater environment and the AUV status. For example, the three cells of the battery are monitored to detect failure. The monitor data are entered directly into the influence diagram by instantiating a node to the observed value. A decision node is also instantiated to a value as that action is taken. The new evidence is propagated to other nodes using HUGIN's inference algorithm.

The top-level utility if the current plan is followed is approximately the current utility of the `U` node since the nominal plan steps are set in the decision nodes of the influence diagram. The difference between this utility and the state resulting after doing replanning is the value of replanning. If the value of replanning is non-positive, the next step of the nominal plan should be executed. If the value of replanning is positive, some type of replanning computation should occur.

6.4.2 Plan modification focusing

Overview The Focusing module is used to direct the replanning to promising subregions of the planning window. The planning window is 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 `U.t+k` node. This is because the nominal plan steps are "programmed" in by initializing the prior probabilities of the corresponding decision nodes to near 1 for the nominal plan's actions. 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, replanning decisions affecting different portions of the utility model in the influence diagram are considered. For example, in figure 6.2, the direct influences on the utility node $U.3$ are the $DataAccrued.3$, $DataRecovered.3$, and $FuelGauge.3$ nodes. The Focusing algorithm seeks to work on repairs to the plan by focusing 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 acyclic graph rather than just a subtree. There are various methods to convert an acyclic graph network into a tree network (e.g., cutset conditioning [73]) In the following analysis, we assume a tree structure, and later in this section we discuss relaxing this assumption.

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.

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" [45, 78, 59, 36] used in decision-theoretic metareasoning. The ordered list is sent to the Modification module. That module may replan at an abstract level, or it may recursively call the Focusing module to again expand a node, and order its parents 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 belief net.

Time cost When TC is greater than the expected benefit, EU_I, the value of replanning is negative, and it should not be done. By varying the TC function, we 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 use Bayesian prediction which can incorporate prior knowledge and incrementally update the distribution of s.

There are several ways to estimate s. We could do a maximum likelihood estimate by maintaining a running average of the time cost over replanning episodes. But we also have some prior knowledge that should bias our interpretation of new evidence about s. For example, we have prior knowledge of the number of nodes in the subtree, the number of states of the nodes, the replanning algorithm used, etc. Therefore, the most general solution is to use Bayesian prediction:

$$p(s|\mathbf{e}) = \frac{p(s)p(\mathbf{e}|s)}{p(\mathbf{e})} \tag{6.1}$$

where prior knowledge about the value of s is captured in $p(s)$, and evidence is represented as $\mathbf{e}$.

From DeGroot [19], if we use squared error for the Bayes estimator and we assume that s is normally distributed with variance σ^2, and our prediction is also a normal distribution with mean μ and variance v^2, then after n observations of x_i, the posterior distribution of s is a normal distribution with mean μ_1 and variance v_1^2 as follows:

$$\mu_1 = \frac{\sigma^2\mu + nv^2\bar{x}_n}{\sigma^2 + nv^2} \tag{6.2}$$

$$v_1^2 = \frac{\sigma^2 v^2}{\sigma^2 + nv^2} \tag{6.3}$$

These two formulas provide a fast, incremental way to update the s for a particular subtree. After each replanning episode, each subtree that was replanned has its s updated.

Utility of replanning a subtree $\hat{EU}_I(replan(X))$, of Equation 6.1, 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 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. 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):

$$EU_I(replan(X)) = \int p(\phi')u(replan(X))d\phi' \tag{6.4}$$

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

$$\begin{aligned} u(replan(X)) &= u(\phi') - u(\phi) && (6.5)\\ EU_I(replan(X)) &= \int p(\phi')(u(\phi') - u(\phi))d\phi' && (6.6) \end{aligned}$$

To analyze a network to find where to replan, we start at the `U` node and work backwards through the network to the base-level actions. For example in figure 6.2, `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 opti-

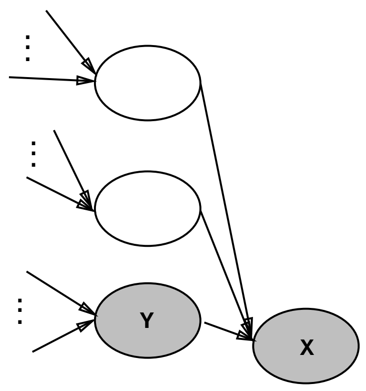


Figure 6.5: Example influence diagram. With the tree structure, each node has at most one child.

mal 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`.

More formally, 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 (e.g., Figure 6.5). 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 6.8:

$$u(\phi, \xi) = \int p(\phi_Y | \phi, \xi) u(\phi_Y) d\phi_Y \qquad (6.7)$$

$$u(\phi) = \int p(\phi_Y | \phi) u(\phi_Y) d\phi_Y \qquad (6.8)$$

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 $u(U.i = 1) = 1$ and $u(U.i = 0) = 0$. 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 6.6, by ranging only over the possible states of X:

$$\hat{EU}_I(replan(X)) = \sum_i p(X = x_i)(u(X = x_i) - u(\phi)) \tag{6.9}$$

where substituting into Equation 6.8

$$u(X = x_i) = \sum_j p(Y = y_j | X = x_i) u(Y = y_j) \tag{6.10}$$

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 [47]. 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 as in the s updating.

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.

Theorem 1

The optimal value for a decision node D is obtained by propagating the evidence back from the U node ($u(U.i = 1) = 1 \;\; u(U.i = 0) = 0$) through nodes on all paths between U and D: $X_1, X_2, \ldots, X_n$.

Proof

By definition, the utility of $u(D = d_i)$ is the expected utility of the utility node U (Equation 6.11). From the belief network construction of the U chance node, the expected utility is the conditional probability that $U = 1$ (Equation 6.12). Using the independence relations represented by the belief network, the conditional probability can be computed by expanding it to the conditional probabilities along all paths between U and D (Equation 6.13) where $X_1, X_2, \ldots, X_n$ represent the nodes on any path between U and D.

$$\begin{aligned} u(D = d_i) &= E[U.i | D = d_i] && (6.11)\\ &= p(U.i = 1 | D = d_i) && (6.12)\\ &= \int_{X_1} \cdots \int_{X_n} p(U{=}1|X_1...X_n)p(X_1...X_n)p(X_1...X_n|D{=}d_i) && (6.13) \end{aligned}$$

The optimal decision d^* is then obtained by

$$d^* = argmax_{d_i \in D} u(D = d_i) \qquad (6.14)$$

Example Using the influence diagram of Figure 6.2, we can compute $\hat{VI}(replan(\texttt{FuelGauge.3}))$. 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,

$$\begin{aligned} u(\texttt{FuelGauge.3}) &= u(U.3 = 1)p(\texttt{U.3=1}|\texttt{FuelGauge.3,DataAccrued.3,DataRecovered.3}) +\\ &\quad u(U.3 = 0)p(\texttt{U.3=0}|\texttt{FuelGauge.3,DataAccrued.3,DataRecovered.3})\\ &= 1 * p(\texttt{U.3=1}|\texttt{FuelGauge.3,DataAccrued.3,DataRecovered.3}) + \end{aligned}$$

$$\begin{aligned} & 0 * p(\texttt{U.3=0}|\texttt{FuelGauge.3,DataAccrued.3,DataRecovered.3}) \\ = \; & p(\texttt{U.3=1}|\texttt{FuelGauge.3,DataAccrued.3,DataRecovered.3}) \\ = \; & 0.29 \end{aligned}$$

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

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

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

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

Putting everything together,

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

$$= \quad 0.07$$

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.

Complexity The time complexity of the $\hat{VI}$ calculation is low since only one level of evidence propagation in the belief 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 $\hat{s}$ and $p(X = x_i)$, the sufficient statistics are the mean and variance which are stored with each node.

Acyclic graphs Currently, though we deal with acyclic graphs rather than trees in our domain models, we treat the graph as a tree, essentially assuming that the children subgraphs of a node are independent of each other. We can extend the replanning algorithm so it considers acyclic graphs rather than subtrees as candidates for replanning. One method is to convert the acyclic graph to a tree structure, for example, by using cutset conditioning [73]. For example, instead of only considering modifying the `FuelGauge.3` subtree, we can also consider simultaneously modifying the `DataRecovered.3` subtree. In general, all subsets of parents are evaluated and the subset that has the maximum $\hat{VI}$ is replanned. For the $\hat{EU}$ calculation, instead of evaluating a node set to a particular state (e.g., $U(X = x_i)$), the cross-product of the subset states are used (e.g., $U(X = x_i, Y = y_j)$). Evaluating multiple nodes does cause the complexity to grow exponentially. For example, instead of evaluating three parents individually, the algorithm must evaluate the powerset: all $2^3 - 1$ subsets of subtrees. To counteract this exponential blowup, we should consider heuristics that favor promising subsets and examine only those subsets.

6.4.3 Plan modification

The Modification module changes the plan and outputs a modified plan. From the Focusing module, we have an ordered list of nodes. If the first node of the list is a decision node, then we set the node to its computed optimal value and exit. The decision node may be at any level of abstraction. For example using figure 6.2 and starting at the top level (the `U.3` node), the parents are evaluated and an ordered list of `FuelGauge.3`, `DataAccrued.3`, and `DataRecovered.3` is returned. At this point, if the abstract decision node `D.A0-2` is at the head of the ordered subtrees list, it is replanned, otherwise the Focusing module is recursively called to expand the head of the list. The utility of setting an abstract action is compared directly to the estimated utility of replanning other subtrees. If setting the abstract action to its best value has a higher utility than replanning, then the abstract action is set by selecting one of its values. A value of the abstract action may either omit or set a pre-specified instantiation for a sequence of lower-level actions.

This procedure provides a rational basis to do hierarchical replanning based on decision-theoretic considerations. It may not always be optimal to start hierarchical replanning at the most abstract level, and by considering the utility of replanning at different levels of abstractions, we have gained a rational way to control hierarchical planning.

Continuing with the battery failure example, suppose that the Focusing module has traveled down the `FuelGauge.3` subtree and has passed a list of subtrees with the `D.1` decision as its first element, and its optimal state setting in the node. The node is reset to its optimal setting and the new evidence is propagated through the network. In this case, `D.1` is changed from `WAIT2` to `WAIT1` reducing the wait time from 2 loops to 1. But in general, changing one action may not be sufficient or may affect other actions. We can also modify multiple controllable nodes as discussed earlier, but still preclude reordering or introducing plan steps. By restricting replanning in this way, we are able to provide baseline functionality, and we can investigate the necessity of more complex planning strategies.

Once the replanning algorithm makes its changes, control is passed back to the Plan Impact Analysis module. It is possible that replanning is again necessary rather than executing the plan.

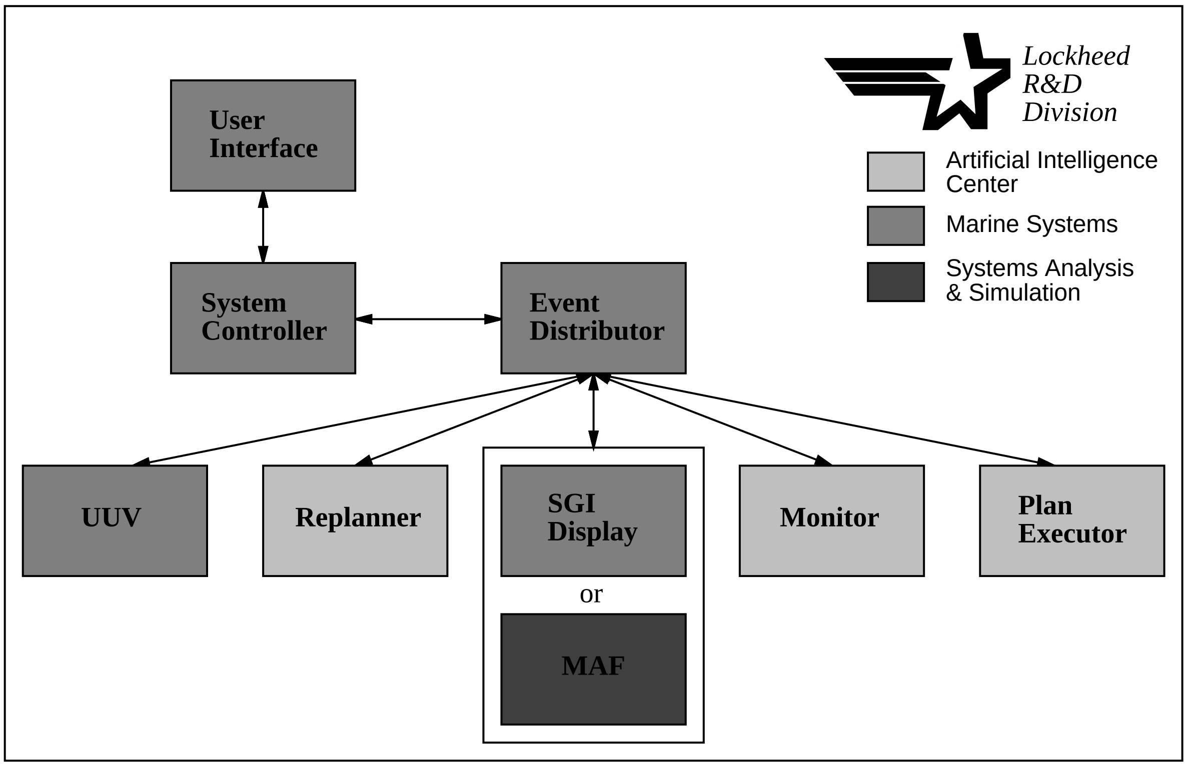


Figure 6.6: A box architecture overview of the AUV control system.

6.5 Example: Gathering Sensor Data

In this section, replanning examples will looked at by examining the behavior of the implemented control system for the AUV. The replanning system is built on top of a multi-process architecture developed at Lockheed. There are 5 parallel-executing processes that pass messages to each other through the event distributor (Figure 6.6).

The main, four processes can be run on various computers. In most experiments, we have used a Sun SPARCstation 10 for the main processes and a Silicon Graphics Iris 4D/310GTX workstation for the graphics simulator process.

Three examples of replanning are shown in Figure 6.7. In the first scenario (the leftmost column), when a Battery Failure occurs, the AUV decides to omit the Loi-

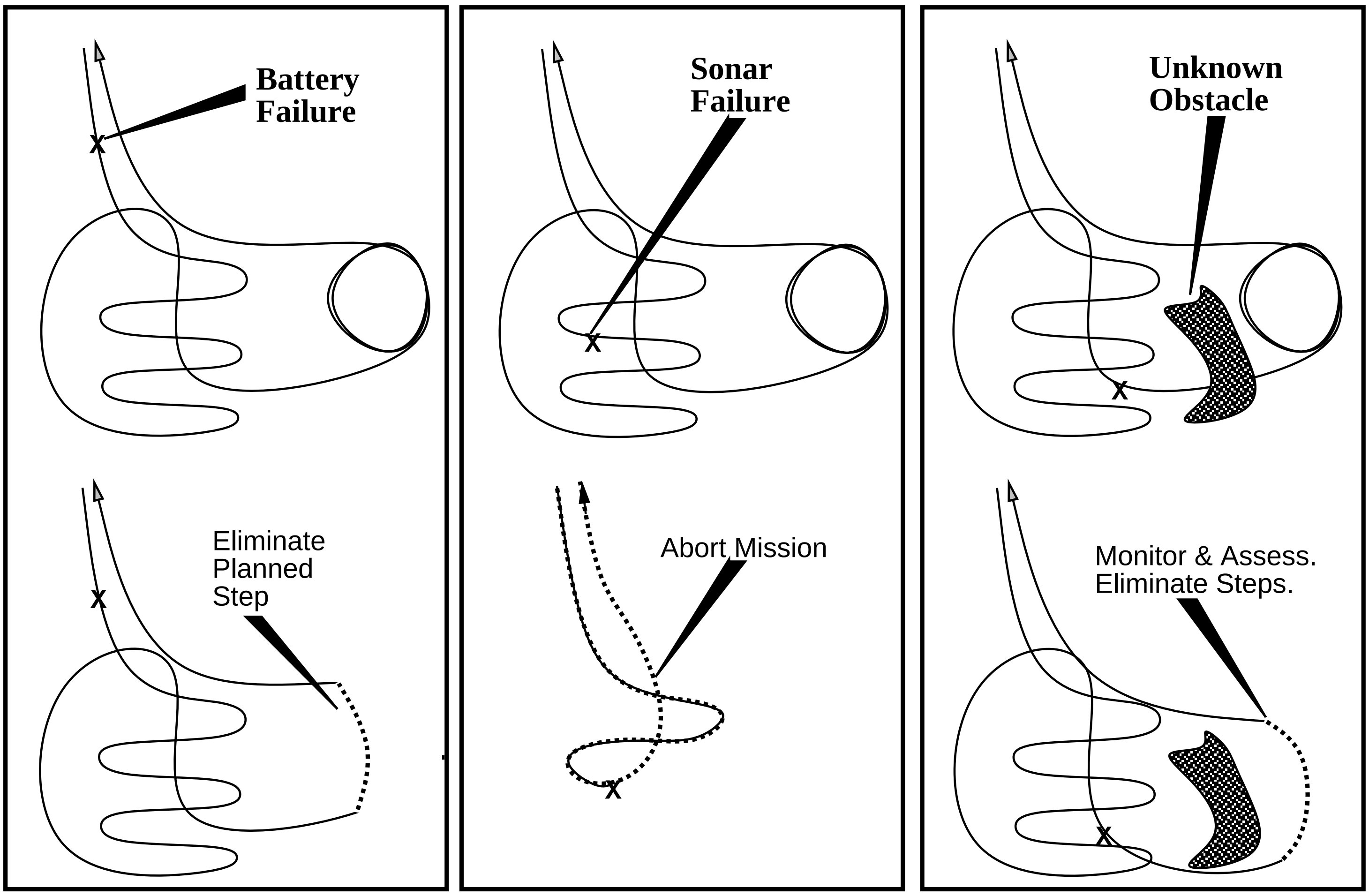


Figure 6.7: Three examples of the autonomous capability. When the event of the top row occurs, the corresponding replanning of the bottom row occurs.

ter step to conserve fuel. This scenario demonstrates several points. First, there is flexibility in when the fault occurs. The Battery Failure evidence can be entered at an arbitrary time in the mission and an appropriate replan will occur. Second, this scenario demonstrates an "intermediate" response in changing the Loiter action, rather than just aborting mission. This functionality is available due to the multiattribute utility model that combines preferences for vehicle safety, data recovery, and fuel usage.

In the second scenario (the middle column of Figure 6.7), a navigation sonar failure occurs. In this case, the increased danger to the vehicle causes the replanner to alter the remaining plan to return immediately for pickup.

The third scenario (the rightmost column of Figure 6.7) shows how the AUV reacts when it encounters an unexpected obstacle. The low-level path planner plots a new course and sends the new course to the replanner which recomputes the energy usage and sensor data accrual projections. In this case, there is no need for extra loitering for data to accrue do to the detour caused by the obstacle. In all three scenarios, the same belief network is used that models the AUV and the domain and can thereby handle multiple faults.

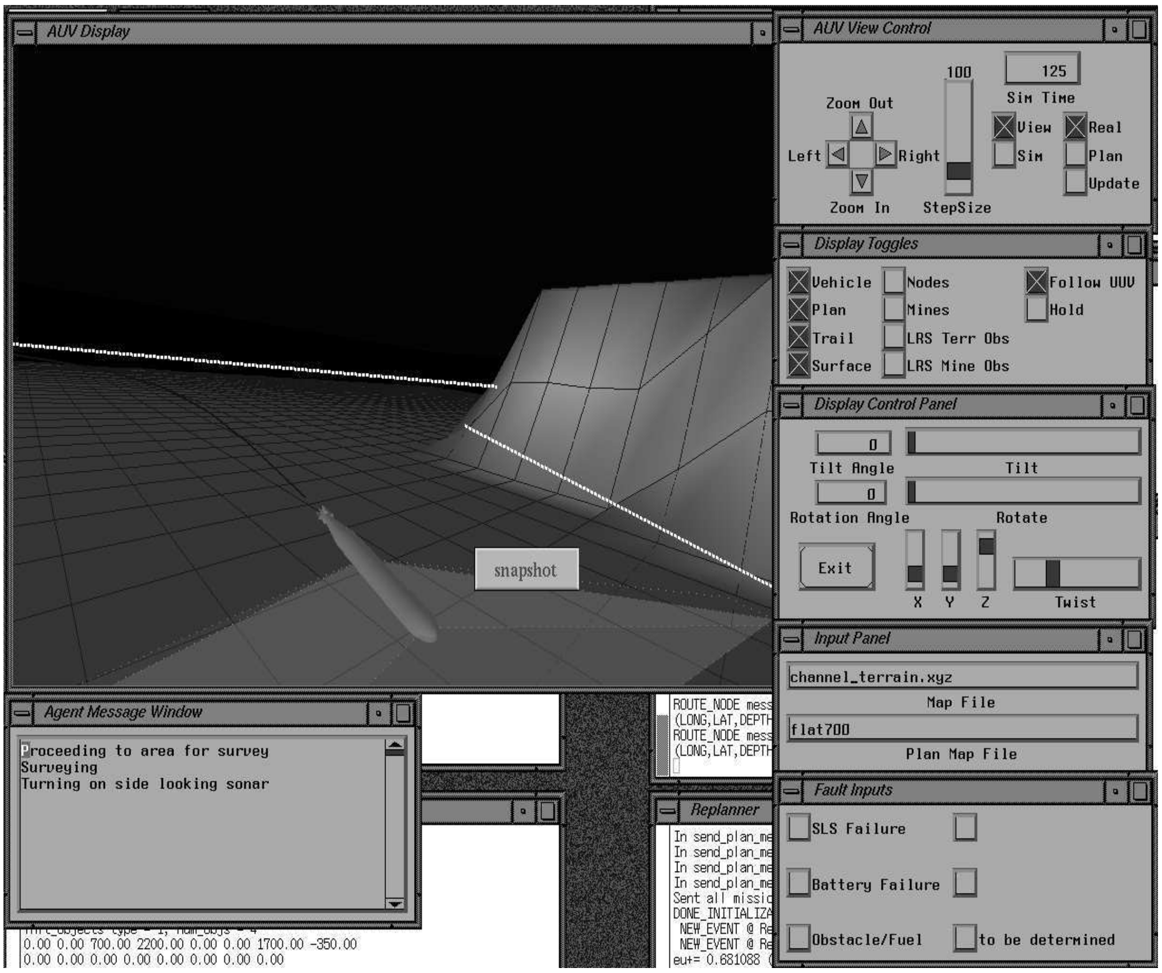
6.5.1 SGI Simulation

The following figures are screendumps of the SGI simulation, showing the operation of the replanning system.

6.6 Example: Mine Mapping Mission

As a second example domain, we have been experimenting with the "Mine Mapping Mission". As shown in Figure 6.13, the mine mapping mission requires the AUV to map a possible path through a mine field. An expeditionary force will then use the map to find a passage to the shore.

The interesting replanning behavior exhibited in the Mine Mapping Mission was the indefinite length of replanning. As shown on the left side of Figure 6.14, the AUV continued to re-examine the possible clear path through the minefield when it learned



Tilt Angle: 0
Rotation Angle: 0
Map File: channel_terrain.xyz
Plan Map File: flat700

```
Proceeding to area for survey
Surveying
Turning on side looking sonar
```

```
0.00 0.00 700.00 2200.00 0.00 0.00 1700.00 -350.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
```

```
ROUTE_NODE mess
(LONG,LAT,DEPTH
ROUTE_NODE mess
(LONG,LAT,DEPTH
```

```
In send_plan_me
In send_plan_me
In send_plan_me
In send_plan_me
Sent all missic
DONE_INITIALIZA
 NEW_EVENT @ Re
 NEW_EVENT @ Re
eu+= 0.681088 (
```

Figure 6.8: A closeup view of the AUV navigating in the SGI simulation.

that it had made a map error. This type of open-ended, indefinite length planning is possible because the dynamic influence diagram can extend in time indefinitely.

6.7 Conclusion

This chapter described replanning using decision-theoretic metareasoning and a hierarchical plan structure in a real-time, decision-making architecture. We have shown how our replanning strategy can yield an optimal policy, and how use of value

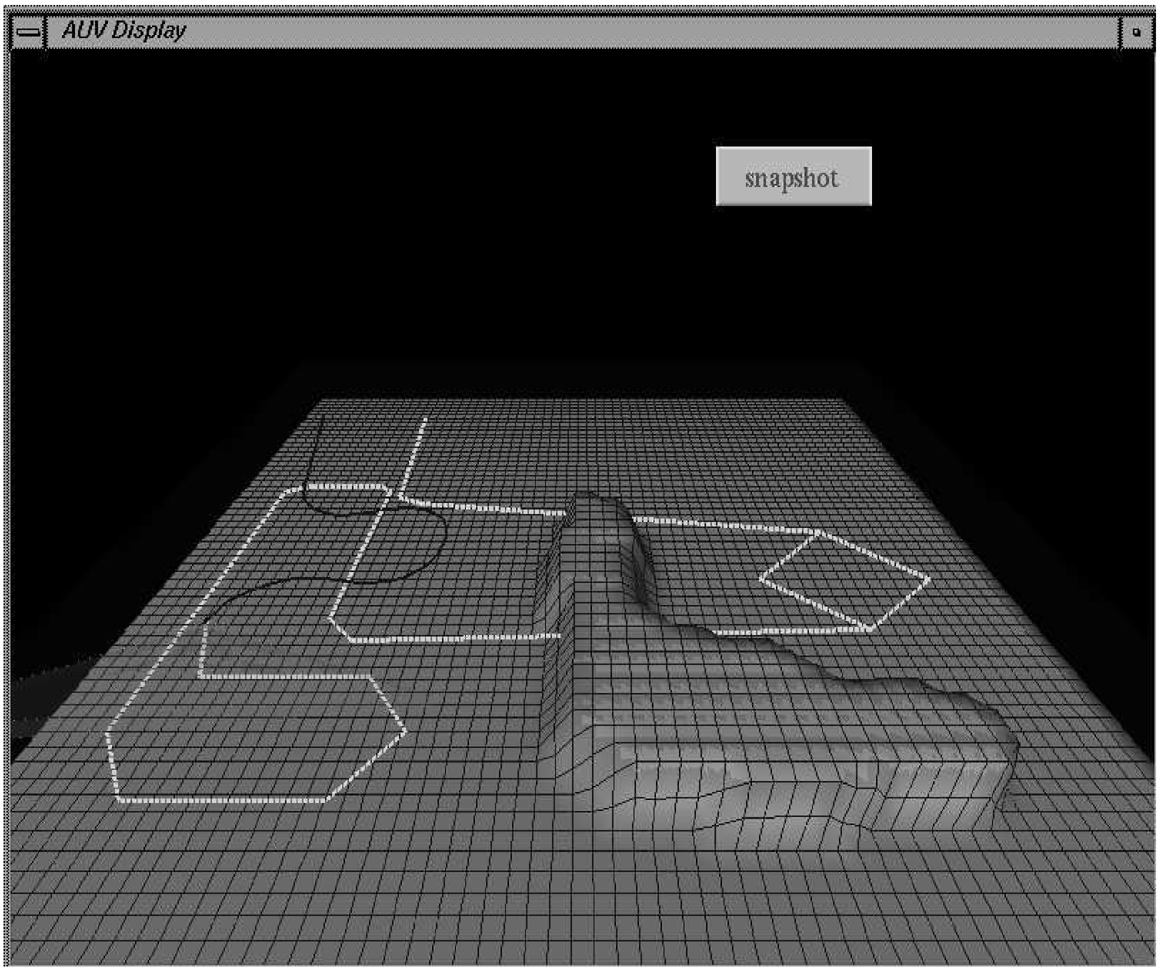


Figure 6.9: SGI simulation shot showing the initial, default plan.

of replanning considerations to control that replanning strategy can provide real-time decision-making and replanning on *opportunity* as well as *crisis*. For example, if time cost is very low, there is a high value to replanning extensively to get a good plan even when everything is going as expected. In contrast, when time cost is very high or if the probability that replanning will succeed is low, even when there is a problem like a battery failure, replanning may still have negative value. We have also shown how the use of abstract actions can be naturally integrated into the decision-theoretic framework to increase the efficiency of deliberation by controlling when and how to

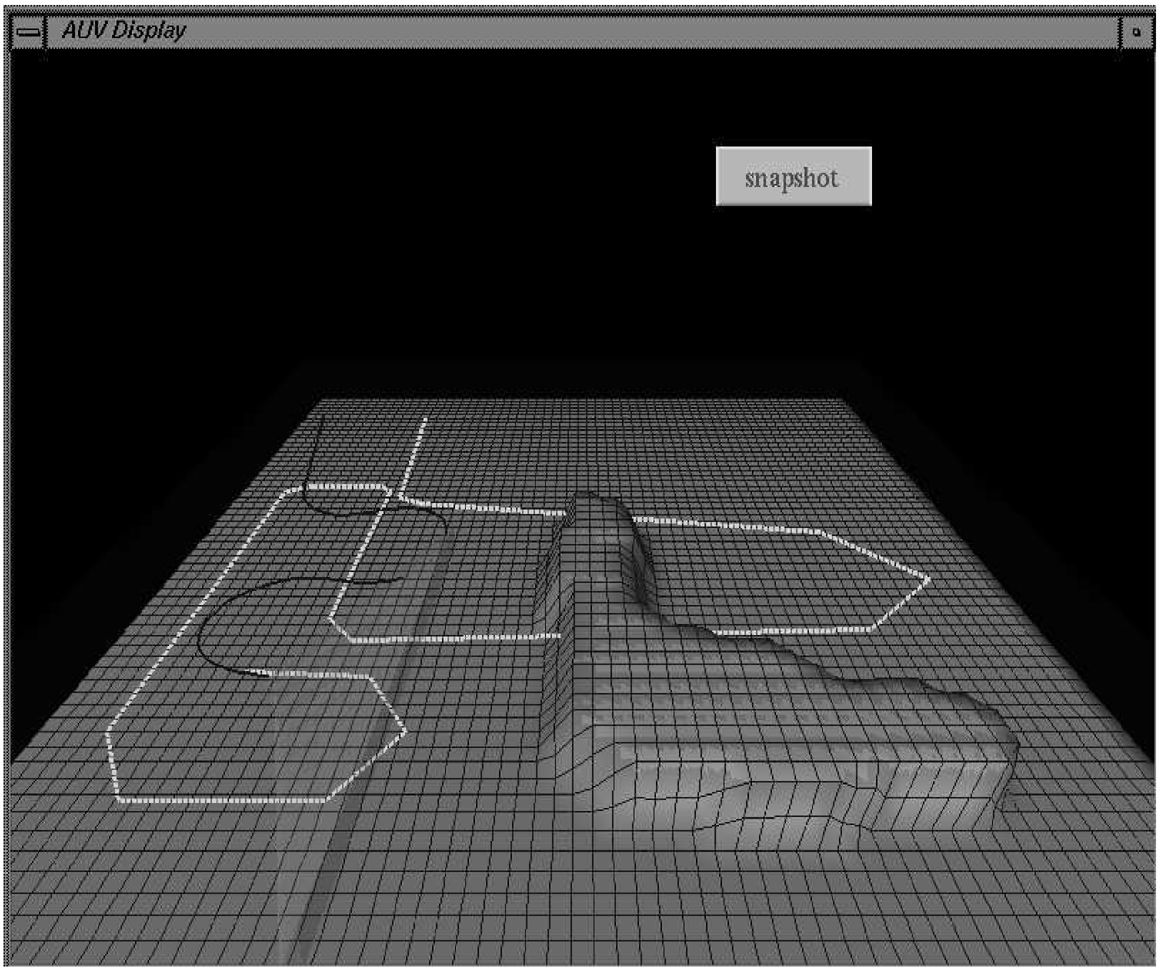


Figure 6.10: SGI simulation after Battery Failure has occurred. The new plan in white omits the Loiter loop step.

use abstraction. The benefits of using an abstraction hierarchy have been widely discussed, but abstraction also has negative aspects because of loss of detail and needs to be subject to decision-theoretic control.

Several areas need further work. Computing the value of replanning currently requires a few estimates, and we would like to improve their accuracy and efficiency. The full use of abstraction hierarchies have yet to be implemented. Finally, building large, complex domain models is a crucial requirement to obtain the inputs to test

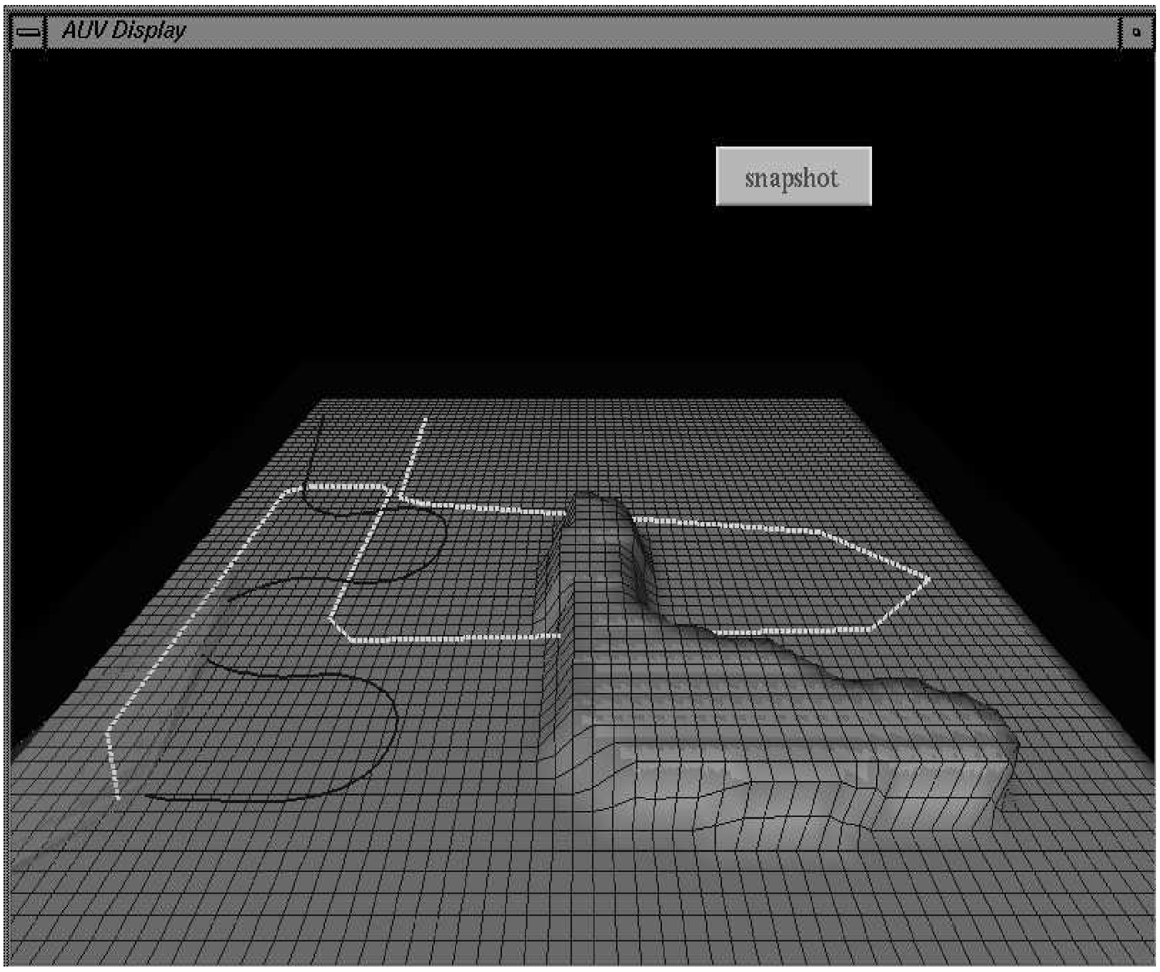


Figure 6.11: SGI simulation after Battery Failure has occurred, but before Sonar Failure evidence has been entered.

the replanner more extensively.

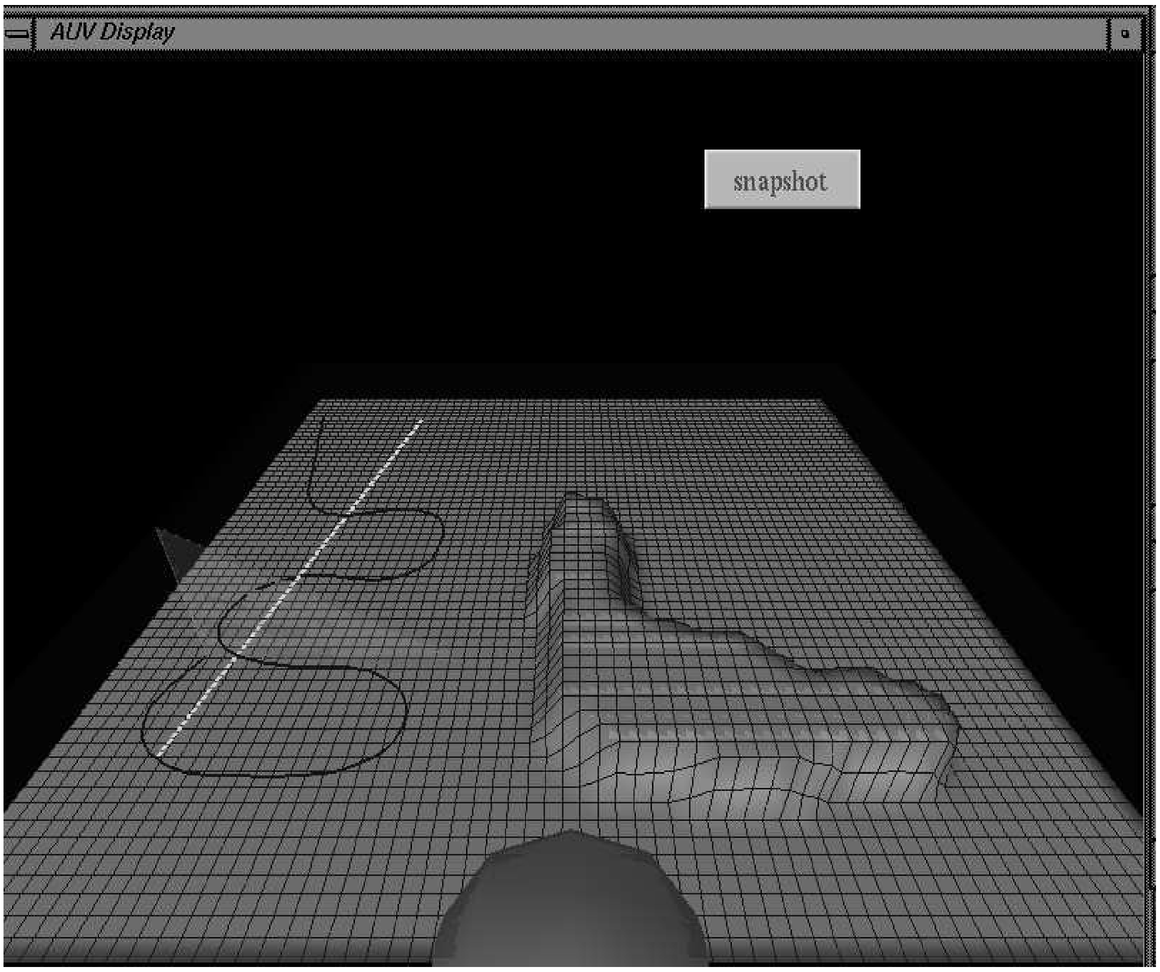


Figure 6.12: SGI simulation after Sonar Failure has occured. The new plan is to immediately return for pickup.

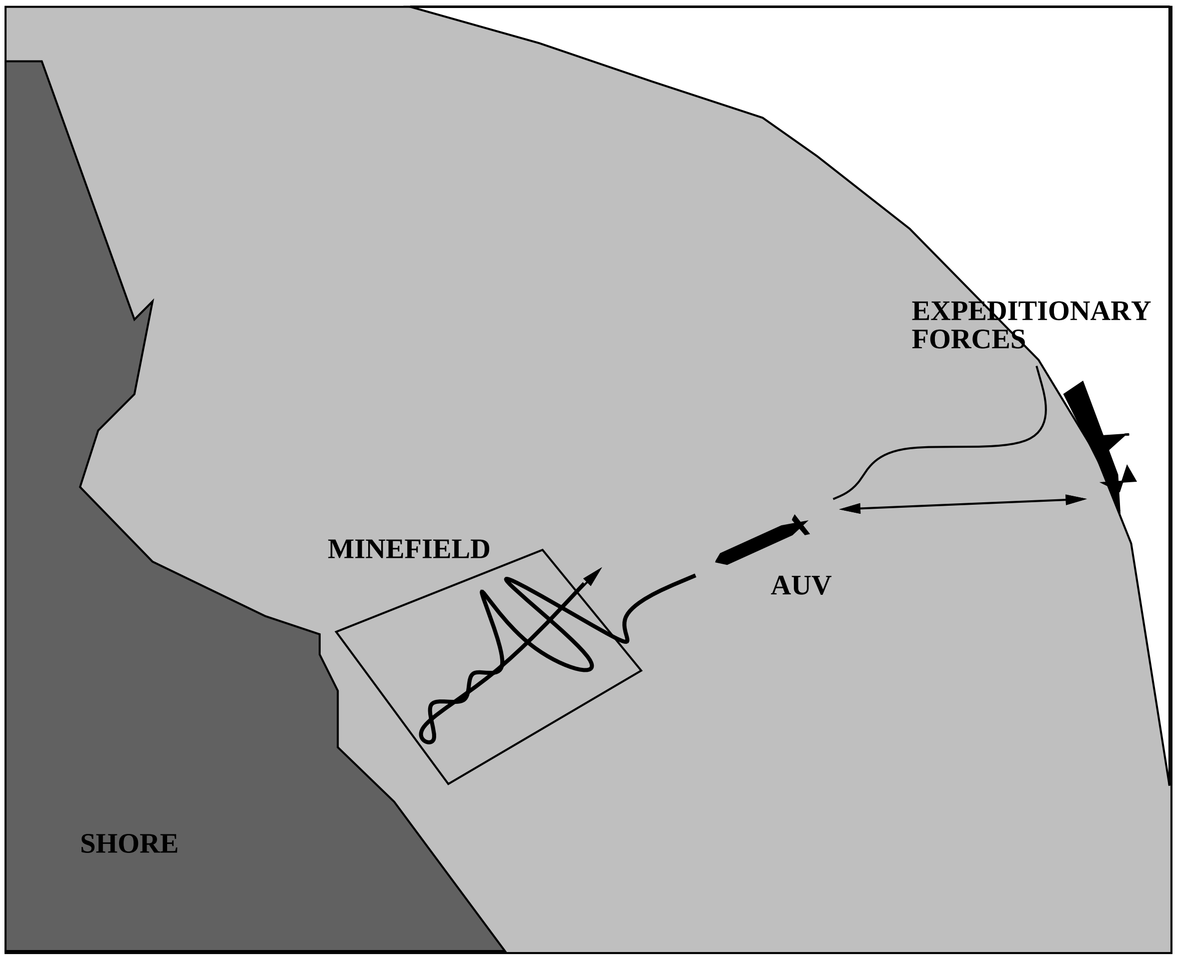


Figure 6.13: The mine mapping domain.

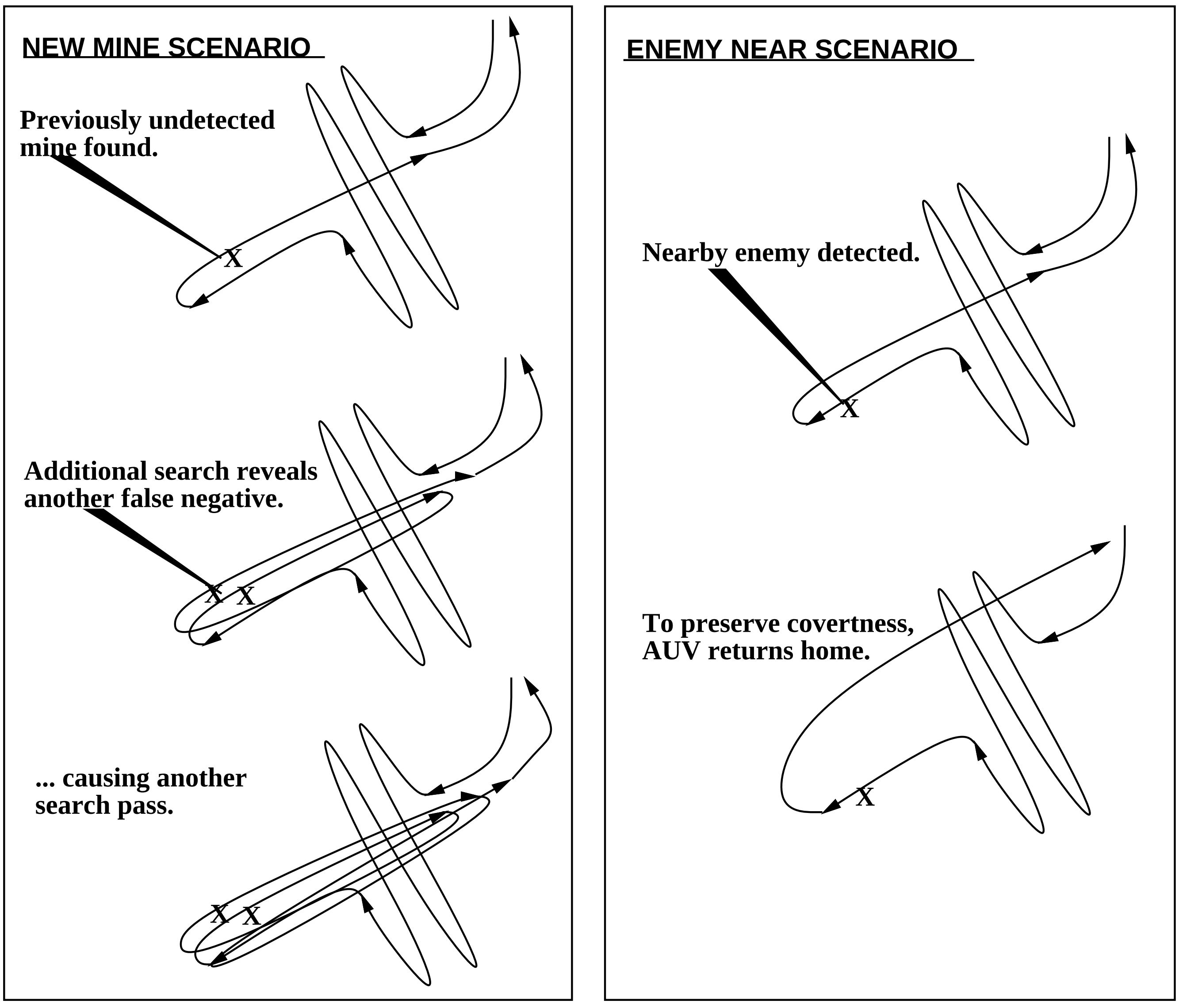

NEW MINE SCENARIO
Previously undetected mine found.
X
Additional search reveals another false negative.
X X
... causing another search pass.
X X
ENEMY NEAR SCENARIO
Nearby enemy detected.
X
To preserve covertness, AUV returns home.
X

Figure 6.14: Two cases of replanning for the mine mapping domain.

Chapter 7

Conclusion

To conclude, this chapter reviews the contributions and points to the limitations and further work of this research program.

7.1 Contributions

The research of this dissertation has joined and built upon a long history of results in problem-solving, planning, and probabilistic and decision-theoretic reasoning. The fundamental problem of decision making has been a topic of interest for thousands of years and approached from many different fields, e.g., philosophy, psychology, mathematics. To classify the goals of this research project in the category of "artificial intelligence" is rather limiting, since the idea of a normative theory of decision making is interdisciplinary and general. However, the assumption of computational resources and the acceptance of basic axioms of decision theory are distinctive to this approach.

The main thesis of this dissertation is that the use of a decision-theoretic knowledge representation system, *multiple execution architectures*, and metalevel control of computation supports decision-making in complex, real-time domains where standard systems falter.

Specific areas of contribution are

- **A new knowledge representation framework.** The need for the *multiple execution architectures* framework has been motivated by the requirement of flexibility when operating in complex domains. By partitioning the space of knowledge relevant to a decision problem, we have identified an intuitive and useful modularization of the possible types of knowledge. The multiple EAs framework also resolves the "reactive" vs. "deliberative" debate that has occupied a lot of attention in research on agent architectures. This was done by showing how reactive and deliberative systems are merely points on a spectrum of possible EAs whose performance can be compared and traded-off.

- **A decision-theoretic representation system.** To represent multiple EAs, *extended influence diagrams* were defined. The "flexibility" advantage of having multiple EAs can then be precisely defined as expected utilities of decisions given a particular amount of computation time. Using extended influence diagrams is one method to deal with the problem of simultaneously maintaining representational accuracy and inferential tractability.

- **A knowledge compilation strategy.** By providing a method to convert knowledge of one type into another type that is more compiled, we have facilitated the general strategy of acquiring knowledge in whichever type is most apt and then converting it via *knowledge compilation* into an efficient, run-time system. This contribution is important because it allows for backward compatibility with existing expert systems that use "if-then" condition-action rules and also forward compatibility to new, more complex domains that should still be efficiently executable.

- **Planning to handle uncertainty and multiple objectives.** A view of *decision-theoretic planning* was presented. Decision theory is specifically designed to handle world uncertainty and multiple objectives. However, because of the complexity of acquiring the necessary probabilities and utilities and the intractability of solving the required equations, decision theory has generally been used only for one-shot, single decision situations (e.g., medical treatment decisions). By providing *metalevel control of planning*, we have made decision-theoretic planning tractable by trading off the utility of replanning and additional time computing.

- **Temporally extensible reasoning.** Exploring the use of *dynamic influence diagrams* that can adjust as time passes, we have implemented a planning system that can operate for indefinite periods of time as it expands and contracts the current planning window. This capability has allowed replanning behavior like the increased number of mine sweeps of the underwater vehicle in the mine mapping mission.

- **Implemented autonomous systems.** The *autonomous underwater vehicle* (AUV) control systems that have been developed at Lockheed Missiles and

Space Company [71] satisfy real needs and requirements of the US Navy customer. The control systems are not refined enough to be fielded on at-sea vehicles, but that is the intent for 1995-6.

7.2 Limitations and Further Work

The goal of real-time decision-making is clear, and the research described in this dissertation towards that end has also clarified what needs to be accomplished to attain better performance. These limitations are the subject of further work, or at the very least, further serious consideration.

- **More complex planning issues.** Classical AI planning techniques have identified several useful concepts that are not exploited in the planning system described here. For example, *subgoals* play an integral role in focusing on what a planner should work. Our ideas on integrating subgoals are not yet presentable. The benefits of *abstraction* have been used to some extent here, but a more complete theory of generalization in the decision-theoretic framework is necessary. *Temporal projection* was accomplished by using a planning window with a fixed lookahead. Any such approach can be plagued by the "horizon" problem where a significant outcome state is just beyond the current window. Methods to use variable-amount and focused temporal projection need to be developed.

- **Learning.** The relative advantages of learning of the respective knowledge types given different domain types is a well-contained research topic that needs to be addressed. For example, given a utility function for time cost and a particular type of feedback available from the environment, how does the learning of utilities of outcome states compare to the learning of q-values (utilities of state-action pairs). Of course, learning systems that can be integrated with this agent architecture would also be useful.

- **Knowledge acquisition bottleneck.** Learning systems would be useful to alleviate the difficulty in acquiring the requisite knowledge for the agent. Learning from large databases and observations is necessary to achieve greater autonomy.

- **Approximate decision models.** We have only looked at knowledge compilation in the behaviorally equivalent sense. The problem should also be examined in the case where the compilation is approximate, and there is a real trade-off between the quality of results and computational properties of execution architectures.

- **Additional applications.** The unsubstantiated claim is that the multiple EAs and metalevel control technology will work for any domain. But to date, we have only implemented two significant applications, both for an AUV. Other applications that are in progress are a decision-making system to control a automated automobile [48], and an integrated sensor and control management system for a Nomadic mobile robot [68].

- **Additional analysis.** Comparative analyses of several of the algorithms needs to done to completely justify their benefits. The metalevel control of planning needs to be compared to systems with different types of metalevel control. The difficulty is that there is no "gold-standard" planning system or domain with which to compare. The benefits of knowledge compilation also need to be analyzed. In this case, the method of comparison is clearer, but it still depends on major assumptions about the utility structure of the domain. Analysis of the appropriateness of the Markov state assumption where the next state is independent of the past given the current state needs to be done. It may be the case that the Markov assumption is appropriate for the automated underwater vehicle domain, but not appropriate for a human planning his health care choices, for example. Methods to relax the Markov assumption will directly impact the updating algorithm for the dynamic influence diagrams and will probably impact any planning algorithm.

I expect real progress on real applications to be incremental. Each real problem teaches different lessons on what theory/hacking/bureaucratic problem needs to be solved next. But the fundamental theory remains sound, the useful problems are there to be solved, and the hope of AI persists.

Bibliography

[1] P. E. Agre and D. Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence*. Morgan Kaufmann Publishers, 1987.

[2] J. Allen, J. Hendler, and A. Tate. *Readings in Planning*. Morgan Kaufmann Publishers, San Mateo, CA, 1990.

[3] S. K. Andersen, K. G. Olesen, F. V. Jensen, and F. Jensen. HUGIN— a shell for building belief universes for expert systems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*. Morgan Kaufmann Publishers, 1989.

[4] J. R. Anderson. Knowledge compilation: The general learning mechanism. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume 2. Morgan Kaufmann Publishers, Los Altos, CA, 1986.

[5] J. S. Breese. Construction of belief and decision networks. *Computational Intelligence*, 1992.

[6] J. S. Breese and M. R. Fehling. Control of problem solving: principles and architecture. In *Proceedings of the Fourth Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers, 1988.

[7] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1), 1986.

[8] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32, 1987.

[9] P. Cheeseman. In defense of probability. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 1002–1009. Morgan Kaufmann Publishers, 1985.

[10] G. F. Cooper. A method for using belief networks as influence diagrams. In *Proceedings of the Fourth Conference on Uncertainty in Artificial Intelligence*, pages 55–63, 1988.

[11] G. F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42:393–405, 1990.

[12] R. T. Cox. Of inference and inquiry– an essay in inductive logic. In Levine and Tribus, editors, *The Maximum Entropy Formalism.* MIT Press, 1979.

[13] P. Dagum, A. Galper, and E. Horvitz. Dynamic network models for forecasting. In *Proceedings of the Eighth Conference on Uncertainty in Artificial Intelligence*, pages 41–48, 1992.

[14] P. Dagum and M. Luby. Approximating probabilistic inference in belief networks is NP-hard. *Artificial Intelligence*, 1993.

[15] B. D'Ambrosio. Local expression languages for probabilistic dependence: a preliminary report. In *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence.* Morgan Kaufmann Publishers, 1991.

[16] B. D'Ambrosio. Incremental probabilistic inference. In *Proceedings of the Ninth Conference on Uncertainty in Artificial Intelligence*, pages 301–308, 1993.

[17] T. Dean, K. Basye, R. Chekaluk, S. Hyun, M. Lejter, and M. Randazza. Coping with uncertainty in a control system for navigation and exploration. In *Proceedings of the Eighth National Conference on Artificial Intelligence.* Morgan Kaufmann Publishers, 1990.

[18] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence.* Morgan Kaufmann Publishers, 1988.

[19] M. H. DeGroot. *Probability and Statistics, Second Edition.* Addison-Wesley, Reading, MA, 1986.

[20] M. desJardins. *PAGODA: A model for autonomous learning in probabilistic domains.* PhD thesis, UC Berkeley, Computer Science Division, 1992.

[21] J. Doyle. Rationality and its roles in reasoning. In *Proceedings of the Eighth National Conference on Artificial Intelligence.* Morgan Kaufmann Publishers, 1990.

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

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

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

[25] R. J. Firby. An investigation into reactive planning in complex domains. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 202–207. Morgan Kaufmann Publishers, 1987.

[26] N. S. Flann and T. G. Dietterich. A study of explanation-based methods for inductive learning. *Machine Learning*, 2:139–172, 1989.

[27] A. M. Flynn, R. A. Brooks, and L. S. Tavrow. Twilight zones and cornerstones: A gnat robot double feature. Technical Report MIT AI Memo 1126, Computer Science Department, 1989.

[28] R. Fung and K. Chang. Weighing and integrating evidence for stochastic simulation in belief nets. In M. Henrion, R. D. Shachter, L. N. Kanal, and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence*, volume 5, pages 209–219. Elsevier Science Publishers B. V., 1990.

[29] R. P. Goldman and E. Charniak. Dynamic construction of belief networks. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, pages 171–184, 1990.

[30] R. Goodwin and R. Simmons. Rational handling of multiple goals for mobile robots. In *Proceedings of the First International Conference on AI Planning Systems*, pages 70–77, 1992.

[31] K. J. Hammond. Explaining and repairing plans that fail. *Artificial Intelligence*, 45:173–227, 1990.

[32] S. Hanks. Practical temporal projection. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 158–163, 1990.

[33] O. Hansson, A. Mayer, and S. Russell. Decision-theoretic planning in BPS. In *Proceedings AAAI Spring Symposium on Planning in Uncertain Environments*, 1990.

[34] D. E. Heckerman. A tractable inference algorithm for diagnosing multiple diseases. In L. N. Kanal and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence*, volume 5, pages 163–172. North-Holland Publishers, Amsterdam, 1990.

[35] D. E. Heckerman, J. S. Breese, and E. J. Horvitz. The compilation of decision models. In *Proceedings of the Fifth Conference on Uncertainty in Artificial Intelligence*, 1989.

[36] D. E. Heckerman, E. J. Horvitz, and B. Middleton. An approximate nonmyopic computation for value of information. In *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers, 1991.

[37] M. Henrion. Propagating uncertainty in bayesian networks by probabilistic logic sampling. In L. N. Kanal and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence*, volume 2. Elsevier Science Publishers B. V., 1988.

[38] M. Henrion. Towards efficient probabilistic diagnosis in multiply connected networks. In R. M. Oliver and J. Q. Smith, editors, *Influence Diagrams, Belief Nets and Decision Analysis*, pages 385–407. John Wiley & Sons, Chichester, UK, 1990.

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

[40] E. Herskovits and G. Cooper. Kutato: An entropy-driven system for construction of probabilistic expert systems from databases. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, pages 117–125, 1990.

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

[42] E. J. Horvitz. Reasoning under varying and uncertain resource constraints. In *Proceedings of the Seventh National Conference on Artificial Intelligence*. Morgan Kaufmann Publishers, 1988.

[43] E. J. Horvitz, G. F. Cooper, and D. E. Heckerman. Reflection and action under scarce resources: Theoretical principles and empirical study. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*. Morgan Kaufmann Publishers, 1989.

[44] E. J. Horvitz, D. E. Heckerman, and C. P. Langlotz. A framework for comparing alternative formalisms for plausible reasoning. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 210–215. Morgan Kaufmann Publishers, 1986.

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

[46] R. A. Howard. Decision analysis in systems engineering. In R. A. Howard and J. E. Matheson, editors, *Readings on the Principles and Applications of Decision Analysis*, volume 1, pages 57–94. Strategic Decisions Group, Menlo Park, CA, 1984.

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

[48] T. Huang, G. Ogasawara, and S. Russell. Symbolic traffic scene analysis using belief networks. In *Working Notes of the AAAI-93 Workshop on AI in Intelligent Vehicle Highway Systems*, 1993.

[49] K. Kanazawa and T. Dean. A model for projection and action. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*. Morgan Kaufmann Publishers, 1989.

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

[51] U. Kjaerulff. A computational scheme for reasoning in dynamic probabilistic belief networks. In *Proceedings of the Eighth Conference on Uncertainty in Artificial Intelligence*, pages 121–129, 1992.

[52] U. Kjaerulff. dHUGIN API reference manual. Technical report, Dept. of Math. and Computer Science, Aalborg University, DK, 1993.

[53] C. Knoblock. Search reduction in hierarchical problem solving. In *Proceedings of the Ninth National Conference on Artificial Intelligence*. Morgan Kaufmann Publishers, 1991.

[54] J. E. Laird, A. Newell, and P. S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33, 1987.

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

[56] S. L. Lauritzen and D. S. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, B*, 50:157–224, 1988.

[57] P. E. Lehner and A. Sadigh. Two procedures for compiling influence diagrams. In *Proceedings of the Ninth Conference on Uncertainty in Artificial Intelligence*, pages 335–342, 1993.

[58] P. Maes and R. A. Brooks. Learning to coordinate behaviors. In *Proceedings of the Eighth National Conference on Artificial Intelligence*. Morgan Kaufmann Publishers, 1990.

[59] J. E. Matheson. Using influence diagrams to value information and control. In R. M. Oliver and J. Q. Smith, editors, *Influence Diagrams, Belief Nets and Decision Analysis*, pages 25–48. John Wiley & Sons, Chichester, UK, 1990.

[60] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4, 1969.

[61] S. Minton. Quantitative results concerning the utility of explanation-based learning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*. Morgan Kaufmann Publishers, 1988.

[62] T. M. Mitchell, J. Allen, P. Chalasani, J. Cheng, O. Etzioni, M. Ringuette, and J. C. Schlimmer. Theo: a framework for self-improving systems. In K. VanLehn, editor, *Architectures for Intelligence*, pages 323–355. Lawrence Erlbaum Associates, Hillsdale, NJ, 1991.

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

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

[65] A. E. Nicholson. Qualitative monitoring of a robot vehicle using dynamic belief networks. In *Proceedings of the Eighth Conference on Uncertainty in Artificial Intelligence*, 1992.

[66] D. W. North. A tutorial introduction to decision theory. *IEEE Transactions on Systems Science and Cybernetics*, SSC-4(3), 1968.

[67] G. H. Ogasawara. A decision-theoretic control system for a mobile robot. Master's thesis, University of California, Berkeley, Computer Science Division, 1990.

[68] G. H. Ogasawara. Control of sensing and planning using decision analysis. In *AAAI-91 Fall Symposium on Sensory Aspects of Robotic Intelligence*, 1991.

[69] G. H. Ogasawara. A distributed, decision-theoretic control system for a mobile robot. *SIGART Bulletin*, 2(4):140–145, 1991.

[70] G. H. Ogasawara, T. Omata, and T. Sato. Multiple movers using distributed, decision-theoretic control. In *Proceedings of the 1992 Japan-USA Symposium on Flexible Automation*, 1992.

[71] G. H. Ogasawara and S. J. Russell. Planning using multiple execution architectures. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, 1993.

[72] J. Pearl. Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, 29:241–288, 1986.

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

[74] M. A. Peot and D. E. Smith. Conditional nonlinear planning. In *Proceedings of the First International Conference on AI Planning Systems*, pages 189–197, 1992.

[75] S. J. Russell. Execution architectures and compilation. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence.* Morgan Kaufmann Publishers, 1989.

[76] S. J. Russell, D. Subramanian, and R. Parr. Provably bounded optimal agents. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, 1993.

[77] S. J. Russell and E. H. Wefald. On optimal game-tree search using rational metareasoning. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence.* Morgan Kaufmann Publishers, 1989.

[78] S. J. Russell and E. H. Wefald. Principles of metareasoning. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning.* Morgan Kaufmann Publishers, 1989.

[79] S. J. Russell and E.H. Wefald. *Do the Right Thing: Studies in Limited Rationality.* MIT Press, Cambridge, MA, 1991.

[80] S. J. Russell and S. Zilberstein. Composing real-time systems. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 212–217. Morgan Kaufmann Publishers, 1991.

[81] E. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

[82] L. J. Savage. *The Foundations of Statistics.* Dover Publishers, 1972.

[83] M. J. Schoppers. Universal plans for reactive robots in unpredictable domains. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence.* Morgan Kaufmann Publishers, 1987.

[84] R. D. Shachter. Evaluating influence diagrams. *Operations Research*, 34(6), 1986.

[85] R. D. Shachter, B. D'Ambrosio, and B. A. Del Favero. Symbolic probabilistic inference in belief networks. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 126–131. Morgan Kaufmann Publishers, 1990.

[86] R. D. Shachter and M. A. Peot. Simulation approaches to general probabilistic inference on belief networks. In M. Henrion, R. D. Shachter, L. N. Kanal, and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence*, volume 5, pages 221–231. Elsevier Science Publishers B. V., 1990.

[87] C. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41, 1950.

[88] J. Shavlik and T. G. Dietterich. *Readings in Machine Learning.* Morgan Kaufmann Publishers, San Mateo, CA, 1990.

[89] M. A. Shwe, B. Middleton, D. E. Heckerman, E. J. Horvitz, H. P. Lehmann, and G. F. Cooper. Probabilistic diagnosis using a reformulation of the INTERNIST-1/QMR knowledge base. *Methods of Information in Medicine*, 30:241–55, 1991.

[90] R. Simmons. A theory of debugging plans and interpretations. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 94–99. Morgan Kaufmann Publishers, 1988.

[91] H. A. Simon. *The Sciences of the Artificial.* MIT Press, Cambridge, MA, 1982.

[92] M. Stefik. Planning and meta-planning. *Artificial Intelligence*, 16(2):141–170, 1981.

[93] D. Subramanian and R. Feldman. The utility of EBL in recursive domain theories. In *Proceedings of the Eighth National Conference on Artificial Intelligence.* Morgan Kaufmann Publishers, 1990.

[94] D. Subramanian and J. Woodfill. Situation calculus made indexical. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence.* Morgan Kaufmann Publishers, 1989.

[95] P. Tadepalli. Lazy explanation-based learning: A solution to the intractable theory problem. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1989.

[96] M. Tambe and P. S. Rosenbloom. Eliminating expensive chunks by restricting expressiveness. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence.* Morgan Kaufmann Publishers, 1989.

[97] A. M. Turing. Computing machinery and intelligence. *Mind*, 59, 1950.

[98] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior.* Princeton University Press, 1944.

[99] D. H. D. Warren. Warplan: A system for generating plans. In J. Allen, J. Hendler, and A. Tate, editors, *Readings in Planning.* Morgan Kaufmann Publishers, 1990.

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

[101] M. P. Wellman and J. Doyle. Preferential semantics for goals. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 698–703, 1991.

[102] M. P. Wellman and J. Doyle. Modular utility representation for decision-theoretic planning. In *Proceedings of the First International Conference on AI Planning Systems*, pages 236–242, 1992.

[103] R. Wilensky. *Planning and Understanding.* Addison-Wesley, Reading, MA, 1983.

[104] S. Zilberstein. *Operational rationality through compilation of anytime algorithms.* PhD thesis, University of California, Berkeley, Computer Science Division, 1993.

[105] S. Zilberstein and S. J. Russell. Anytime sensing, planning and action: A practical model for robot control. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, 1993.