TECHNICAL RESEARCH REPORT

SYSTEMS RESEARCH CENTER

*Supported by the National Science Foundation Engineering Research Center Program (NSFD CD 8803012), Industry and the University*

# Planning in Uncertain, Unpredictable, or Changing Environments

*by J. Hendler*

TR 90-45

# Planning in Uncertain, Unpredictable, or Changing Environments

(Working Notes of the 1990 AAAI Spring Symposium)

Edited by:
James Hendler
Systems Research Center
University of Maryland

Abstract:
This report is a compendium of the extended abstracts submitted by participants at the 1990 AAAI Spring Symposium entitled *Planning in Uncertain, Unpredictable or Changing environments.* The papers concentrate on extending planning systems and/or intelligent agent architectures for use in dynamic domains. Three main themes include: the integration of planning and machine learning, the use of probabilistic and decision-theoretic information during planning, and the integration of planning with reaction.

# Working Notes

# AAAI
# Spring Symposium Series

Symposium:
Planning in Uncertain, Unpredictable, or Changing Environments

Program Committee:
James Hendler, University of Maryland, Chair
Thomas Dean, Brown University
Charles Schmidt, Rutgers University
Alberto Segre, Cornell University

# TABLE OF CONTENTS

# Computation and Situated Activity

**Philip E. Agre**
Department of Computer Science
University of Chicago
1100 East 58th Street
Chicago, Illinois 60637

For the last five years, my work has been aimed toward a computational understanding of the situated activity of embodied agents, human and otherwise [Agre in preparation]. I have explored the idea that the organization of purposive activity is an emergent property of interactions between sensible agents and an orderly world. In doing so, I have worked to provide an alternative to the idea that activity is organized through the execution of plans. The alternative view might not be the most useful in all environments, but it does have a clear appeal in environments that are uncertain, unpredictable, or changing. The challenge is to bring some technical substance to the notion of the emergent organization of activity. To this end, this paper briefly describes a series of projects.

1. First I will review the ideas about routine activity that lie behind the running argument system.
2. Building on my experience with this system, I then summarize the ideas about embodiment, indexicality, and perception that lie behind the Pengi system.
3. The third section then discusses a project currently under way to investigate empirically the evolution of routine forms of activity.
4. The final section discusses some of the motivations behind a new project that seeks a deeper computational understanding of the culturally organized worlds within which the situated activity of both people and computers occurs.

Each of these projects has developed from the earlier ones by interpreting the lessons of engineering experience with the help of ideas and conceptual frameworks from social theory. Our social environment is uncertain, unpredictable, and changing, but nonetheless it orderly in a way that permits our interactions with it to be coherently organized.

## Running arguments

The agents constructed with classical planning techniques organized their activity through the construction and execution of plans. If something went wrong, as detected by the failure of a monitored precondition of a plan step, local repairs could be initiated or control could be returned to the plan-construction device. This approach was first described by Miller, Galanter, and Pribram [1960] and was first implemented in Strips [Fikes, Hart, and Nilsson 1972]. Classical planning works best in domains with two properties:

1. It is computationally and epistemologically tractable to construct plans whose execution is unlikely to encounter too much trouble.
2. An executive can readily make rational decisions about whether to continue executing the current plan.

Some real-world domains are like this. I hypothesize that others, like the world of everyday life, are not. The notion of a running argument was an attempt to formulate a different model of action [Agre 1985, in preparation]. Whereas a system based on plan execution only makes periodic decisions about action, an agent engaged in a running argument continually redecides what to do next. Deciding what to do can be computationally expensive, so that designing an architecture to support continual redecision is a challenge. The running argument system addresses this challenge by conjoining a fairly conventional rule language, a simplified version of Amord [de Kleer et al 1977], with an optimized and stripped-down dependency system [Doyle 1979].

The running argument system closes a tight loop with its (simulated) environment. On each tick of the clock, perceptions are entered into the system's database, the rules run, and the chosen actions are retrieved from the database. The system supports the fiction that the entire rule base has forward-chained to exhaustion on every tick of the clock. In reality, the complex rule-firing process must only be applied to those rules which have not run before. Each rule-firing causes a new patch to be added to the system's dependency network, thus permitting future such firings to occur very efficiently through a simple, highly parallel token-passing scheme that might be likened to the operation of a combinational logic circuit. Nilsson [1989] has investigated the formal properties of this kind of scheme in more detail.

This kind of system makes a strong assumption about the kind of activity in which it is to participate. Specifically, it requires that this activity be *mostly routine*, so that almost all of what an agent is doing at any given time is something it has done before. If the agent learns, it will be principally through the incremental evolution of relatively settled forms of activity, with a lesser role assigned to substantial innovations based on extensive simulations of possible future courses of action. In a domain in which activity is mostly routine, the running argument system will settle down to a style of processing in which the dependency network does most of the work, supplemented by a background level of rule-firing that can be sustained without unreasonably complicated or expensive machinery.

## Deictic representation and visual routines

The running argument system has two major shortcomings. First, since it followed technical convention in giving constant symbols (A, JOHN, BLOCK23) a central place in its representation scheme, the dependency records did not generalize away from the particulars of a given scene. Second, since it also followed convention in assuming that its vision system could maintain an up-to-date set of world-model propositions in its database, it had no real theory of the connection between perception, reasoning, and action.

The solution to these problems started from a fuller awareness of the architectural consequences of an agent's having a body and the situated character of its practical reasoning [Rosenschein and Kaelbling 1986, Smith 1986]. In its practical activities, an agent is always involved with particular other entities, and it understands these entities in terms their relation to the activity itself. This is the intuition behind deictic representation [Agre in preparation], which displaces constant symbols from their central role and replaces them with the interactional notion of entities, such as *the-keyboard-on-which-I-am-typing*. Deictic representation provides a novel theory of abstraction from situation particulars, one based not on variables but on what Barwise and Perry [1983] have called the "efficiency" of indexical representations: their ability to refer to different relevant individuals in different situations.

Deictic representation is an interactional notion in that a deictic entity is not a model in an agent's head but rather a sustained pattern of interaction with something in the world. In building an architecture to support this idea, one must co-design the central system (which subserves the reasoning narrowly construed) and the peripheral systems (which subserve perception and motor control), guaranteeing that they can enter into complex but orderly forms of interaction amongst themselves and with the outside world. What is required is an account of visual processing in which perception is not entirely mediated by a world model but instead participates in a more active engagement with the environment [Ballard 1989, Horswill 1988]. An account of visual-system architecture that is suggestive in this regard is Ullman's [1984] concept of visual routines, according to which the visual system supplies a repertoire of visual operations that the central system can apply to focalized regions of the visual image.

David Chapman and I designed the Pengi system [Agre and Chapman 1987] to illustrate these ideas. Pengi combines a central system made of combinational logic and a visual system based loosely on Ullman's ideas (all in simulation) to engage in a continual, flexible interaction with a video game called Pengo. (See [Agre in preparation] for more details.) Starting from an initial, coarse understanding of the dynamics of Pengo-playing, we built Pengi through a process of iterative refinement, using our observations of the system's interactions with the game to direct the search for deepened understandings of the game's dynamics and then revising our ideas about the appropriate set of entities and reasoning processes accordingly.

Our design process was guided in part by a collection of scenarios describing typical courses of interaction between a player and the game. In practice, though, it is unusual for the system to participate in one of these scenarios straight through without interruptions, interpolations, repetitions, rearrangements, or the like. In no useful sense, then, could Pengi be said to be executing plans that we wrote for it. The process of wiring up central systems for agents such as Pengi is rather involved but with a good debugging environment it is comparable in difficulty to any other kind of programming. In the particular domain of video games, most of the difficulty comes from the heavy requirement for rapid and artfully organized scanning of the game board.

The aim of Pengi was to demonstrate a way in which complex, orderly, goal-directed forms of activity can arise as emergent properties of the interaction between a sensible agent and its familiar environment. In particular, we intended Pengi to illustrate the point that careful attention to the orderly properties of improvised activity can lead to simpler architectures. Some confusion has arisen on the point, though, so for the sake of subsequent discussions let us take a moment here to clarify what these themes do and do not mean in the context of Pengi.

1. Although Pengi does not employ any sort of symbolic representations, the point is not to deny the existence of natural language and other semiotic phenomena but simply to refuse a central role in practical reasoning to an account of naming and abstraction based on variables and constants (as in first-order logic).
2. Although Pengi does not make or use plans, the point is not a rejection of every possible concept of plans.

In [Agre and Chapman in press] we sketch an alternative theory of plans that ascribes different properties to them and gives them a smaller role than in the classical planning view.

3. Although Pengi does not employ central system state, the point is not a rejection of state as such but rather a search for alternatives to world models, understood as internal symbolic structures which stand in a systematic, objective correspondence to states of affairs in the world.
4. Although Pengi uses combinational logic to decide upon its actions, the point is not to assert the sufficiency of combinational logic for all problems but rather to stress the possibility of greatly reducing the complexity of an agent's machinery through careful consideration of the nature of its intended patterns of interaction with its world.
5. Although Pengi's combinational logic can be construed as a kind of "table lookup" mapping from "situations" to "actions" [Ginsberg 1989], this way of looking at Pengi's machinery both neglects the complex visual processing that mediates between Pengi and its environment and overlooks the central point of the exercise by presupposing, contrary to our intention, that rationality inheres in single isolated actions rather than in time-extended emergent patterns of interaction. (In any event, as Mel [1989] among others points out, table lookup schemes have an important place in motor learning and one should not write them off too quickly.)
6. Although Pengi employs a fixed circuit to play Pengo, the point is not to assert the sufficiency of fixed circuits for all tasks. Nor do we mean to imply that any program could synthesize such circuits all of a piece. We believe that circuits such as Pengi's could arise through a continual incremental evolution during the course of a system's interactions with its environment. The running arguments work offered a sense of how this might occur.

We have offered Pengi as a provisional illustration of a view of situated agency that locates the important properties of a device not in its machinery but in its orderly patterns of interaction with the world. The engineered artifact is not the device itself, removed from its typical settings in its particular world, but rather these emergent patterns of interaction themselves. The challenge we have set ourselves is not to build devices that can engage in increasingly complex forms of abstract, detached reasoning, but rather to build devices that can engage in increasingly complex forms of concrete, situated activity. This alternative focus requires a redefinition and rethinking of many of the field's central questions, a process that I hope can lead to a renewed awareness of the assumptions underlying various approaches to these questions.

## Routine evolution

To get a fuller sense of how routine forms of activity might evolve in the course of everyday life, Jeff Shrager and I are, as part of an ongoing project, analyzing a videotape of an experimental subject performing a highly routine photocopying task which required making three copies of a seventeen-page-long article from a bound book. (In accord with accepted experimental procedures, the subject was taped secretly and then debriefed and offered the opportunity to have the tape erased.) We plan to present this material fully in a forthcoming article. My purpose here is only to sketch its possible significance for computational research on action.

Even though in a general sense she does roughly the same things to copy each page of the article, the complexity of the subject's performance appears incompatible with the hypothesis that she might have been repeatedly executing a fixed plan. Instead, her relationship to the machine involves a very dense system of continual fine adjustments whose details and extents and combinations vary from page to page within a generally fixed repertoire. The organized character of the activity appears not predetermined but rather an emergent property of an unfolding pattern of interaction [Lave 1988].

We can, though, observe some definite trends in the course of the task. She anticipates events (such as the final flash of the copier) with successively greater accuracy, she omits (by increments) operations that prove not to be entirely necessary (such as closing the copier's cover for each copy), and she finds things to do with the dead times that regularly occur when waiting for various events (again, such as the copier's flashing). These modifications arise not through revolutionary reinventions of her routine but rather through discrete mutations that appear and then, with subsequent iterations, work their way into the fabric of the routine.

We have begun tracking each of these various observed mutations, attempting to understand their origins and dynamics. Our hypothesis is that the mutations arise through the transfer of simple observations or explanations (so that they might be viewed as restricted cases of explanation-based learning) that correspond to small, readily recognizable aspects of the ongoing situation. In a subsequent phase of this research, we hope to match the dynamics of this kind of routine evolution to the forms of machinery that can support it. We believe that our time spent understanding these dynamics will lead us to machinery that, like Pengi's, is simple and elegant and well-suited for the task of that kind of learning.

## Computation as culture

Throughout this research, the leading principle has been that the organization of purposive activity is an emergent property of interactions between sensible agents and orderly worlds. In keeping with computational theory's traditional focus on internal states of individual agents, it has remained obscure what it is about the world that facilitates these organized forms of interaction. Answers to this question will no doubt vary to some extent across species of organisms and robots. Insects (organic or robotic) such as described by Brooks [1986], for example, live in an evolutionarily constituted "niche" in a particular ecosystem. It would be of great interest to understand what these niches consist in and how one might design creatures to inhabit them.

Though my own interest is in the organization of human activities, the issues that arise in the course of computational study of organized activity are equally relevant to the engineering of autonomous agents and the like. In work that is just beginning, I have trying to derive computational ideas about the organization of situated activity from ideas available in social theory, that is, in anthropology, sociology, and developmental social psychology. These fields are unanimous in insisting that the world that human beings inhabit is, in various senses, a *social* world. This means three things:

1. We all live in a densely structured field of human institutions and relationships, with all their local constraints and customs and rituals and other conventional interactional forms.
2. Much of the lived environment is the product of human construction; most of the rest is subject to human domestication, regulation, surveillance, and technical ordering.
3. Our understandings of and actions within the world are mediated by a very large network of culturally organized practices and categories with which we become competent as part of our induction into our societies.

Given these properties of the social world, an important challenge for computational research is to understand what it would mean for an artifact, such as an autonomous robotic agent, to participate in such a world [Gasser 1986].

Social theory has produced a number of ways of understanding the social world. Among those that are of immediate relevance to the issues at hand, I have been influenced principally by two, ethnomethodology and practice theory. Ethnomethodology [Garfinkel 1967, Heritage 1984, Sacks 1964-72] focuses on the intricate and seemingly highly improvised methods through which people, in their organized face-to-face interactions, "construct" the social world. Practice theory [Bourdieu 1977, Ortner 1984] focuses on the ways in which the habits and built forms of a culture aid in producing and reproducing its social structures. Suchman [1987] has outlined some of the relevance of ethnomethodology to computational theory. Lave [1988] has done the same for practice theory.

That action can only be effectively organized in an orderly world is well demonstrated by the capacities and limitations of classical planning techniques. Since these techniques are very sensitive to uncertainty, unpredictability, and change, they will work best in highly regulated environments, such as factory floors, in which possibilities are limited and the planner can be protected from unexpected changes. It is highly likely that classical planning techniques can find profitable application in such an environment [Wilkins 1989].

As a "world," the factory floor exhibits a particular kind of orderliness that facilitates certain ways of organizing an agent's activity. Other worlds, though, possess other kinds of orderliness. In a face-to-face conversation, for example, each party's actions must continually depend on what the other party is doing, if only to nod approval, say "uh-huh," and coordinate adjustments in posture [Goodwin 1981]. As a result, every conversation is a joint improvisation of formidable complexity.

The orderliness of the "world" of conversational interaction, then, is to a considerable extent a contingent product of the speakers' joint efforts. In interacting with other people, we start from an assumption that we share a common understanding of the world and of the currently ongoing situation [Heidegger 1961]. The full complexity of this shared background comes to bear on the details of our interactions in a massive and remarkably detailed way. This is another kind of orderliness in our activities, a source of both guidance and constraint throughout our everyday interactions.

Ideas about the dynamics of conversational interaction can lead to effective computational models. In current work, Chapman [1990] is applying the technology of Pengi in investigating the contribution of shared understandings of an ongoing activity's context to the interpretation of simple instructions. As a computational matter, this process can be remarkably efficient under the assumption that speaker and hearer share deictic representations of the shared situation. Suitably extended, it may also be able to capture some of the important dynamics of routine collaborative activities.

Computational research has much to gain from social theory. The idea that the organization of activity is a product of situated improvisation, for example, is central to ethnomethodology and contributed to early attempts to formulate alternatives to the view of plans employed in classical planning. But now a further point is that many of the salient features of the world participate in a cultural order. The cultural ordering of the world provides, among many other things, support for cognition [Hutchins 1987, Norman 1988]. As members of a culture, our ways of understanding the world are geared to the kind of world our culture organizes. Likewise, our ways of acting in the world are geared to reproducing the kind of world that we can understand. One might thus hope that inquiry into the social world can provide insight into the close interrelationships between the forms of human cognition and the forms of the human environment.

## Bibliography

[Agre 1985] Philip E. Agre, Routines, AI Memo 828, MIT Artificial Intelligence Laboratory, 1985.

[Agre and Chapman 1987] Philip E. Agre and David Chapman, Pengi: An implementation of a theory of activity, *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, 1987, pages 196-201.

[Agre and Chapman in press] Philip E. Agre and David Chapman, What are plans for?, in Pattie Maes, ed., *New Architectures for Autonomous Agents: Task-level Decomposition and Emergent Functionality*, MIT Press, in press.

[Agre in preparation] Philip E. Agre, *The Dynamic Structure of Everyday Life*, Cambridge University Press, in preparation.

[Ballard 1989] Dana H. Ballard, Reference frames for animate vision, *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, 1989, 1635-1641.

[Barwise and Perry 1983] Jon Barwise and John Perry, *Situations and Attitudes*, MIT Press, 1983.

[Bourdieu 1977] Pierre Bourdieu, *Outline of a Theory of Practice*, Cambridge University Press, 1977.

[Brooks 1986] Rodney A. Brooks, A robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation* 2(1), 1986, pages 14-23.

[Chapman 1990] David Chapman, *Instruction Use in Situated Activity*, Technical Report 1204, MIT Artificial Intelligence Laboratory, 1990.

[de Kleer *et al* 1977] Johan de Kleer, Jon Doyle, Guy L. Steele, Jr., and Gerald Jay Sussman, Explicit control of reasoning, *Proceedings of the ACM Symposium on Artificial Intelligence and Programming Languages*, Rochester, New York, 1977.

[Doyle 1979] Jon Doyle, A truth maintenance system, *Artificial Intelligence* 12(3), 1979, pages 231-272.

[Fikes, Hart, and Nilsson 1972] Richard E. Fikes, Peter E. Hart, and Nils J. Nilsson, Learning and executing generalized robot plans, *Artificial Intelligence* 3(4), 1972, pages 251-288.

[Garfinkel 1984] Harold Garfinkel, *Studies in Ethnomethodology*, Polity Press, 1984. Originally published in 1967.

[Gasser 1986] Les Gasser, The integration of computing and routine work, *ACM Transactions on Office Information Systems* 4(3), 1986, pages 205-225.

[Ginsberg 1989] Matthew Ginsberg, Universal plans: An (almost) universally bad idea, *AI Magazine* 10(4), 1989, pages 40-44.

[Goodwin 1981] Charles Goodwin, *Conversational Organization: Interaction Between Speakers and Hearers*, Academic Press, 1981.

[Heidegger 1961] Martin Heidegger, *Being and Time*, translated by John Macquarrie and Edward Robinson, Harper and Row, 1961. Originally published in German in 1927.

[Heritage 1984] John Heritage, *Garfinkel and Ethnomethodology*, Polity Press, 1984.

[Horswill 1988] Ian D. Horswill, *Reactive Navigation for Mobile Robots*, Master's thesis, MIT Department of Electical Engineering and Computer Science, 1988.

[Hutchins 1987] Edwin Hutchins, Mediation and automatization, ICS Report 8704, Institute for Cognitive Science, University of California at San Diego, 1987.

[Lave 1988] Jean Lave, *Cognition in Practice: Mind, Mathematics, and Culture in Everyday Life*, Cambridge University Press, 1988.

[Mel 1989] Bartlett W. Mel, MURPHY: A neurally-inspired connectionist approach to learning and performance in vision-based robot motion planning, Technical Report CCSR-89-17A, Center for Complex Systems Research, University of Illinois, 1989.

[Miller, Galanter, and Pribram 1960] George A. Miller, Eugene Galanter, and Karl H. Pribram, *Plans and the Structure of Behavior*, Henry Holt and Company, 1960.

[Nilsson 1989] Nils J. Nilsson, Action networks, *Proceedings from the Rochester Planning Workshop: From Formal Systems to Practical Systems*, Rochester, New York, 1989.

[Norman 1988] Donald A. Norman, *The Psychology of Everyday Things*, Basic Books, 1988.

[Ortner 1984] Sherry B. Ortner, Theory in anthropology since the sixties, *Comparative Studies in Society and History*, 26(1), pages 126-166, 1984.

[Rosenschein and Kaelbling 1986] Stanley J. Rosenschein and Leslie Pack Kaelbling, The synthesis of digital machines with provable epistemic properties, in Joseph Halpern, ed, *Proceedings of the Conference on Theoretical Aspects of Reasoning About Knowledge*, Monterey, CA, 1986.

[Sacks 1964-72] Harvey Sacks, Unpublished transcribed lectures, University of California, Irvine, 1964-72. Transcribed and indexed by Gail Jefferson.

[Smith 1986] Brian Cantwell Smith, The correspondence continuum, *Proceedings of the Sixth Canadian AI Conference*, Montreal, 1986.

[Suchman 1987] Lucy Suchman, *Plans and Situated Action*, Cambridge University Press, 1987.

[Ullman 1984] Shimon Ullman, Visual routines, *Cognition* 18, 1984, pages 97-159.

[Wilkins 1989] David E. Wilkins, Can AI planners solve practical problems?, Technical Report 468, SRI International Artifical Intelligence Center, 1989.

# Organizing Memory for Probabilistic Search Control

**John A. Allen**
Sterling Federal Systems
AI Research Branch, Mail Stop: 244-17
NASA Ames Research Center
Moffett Field, CA 94035
Allen@ptolemy.arc.nasa.gov

## Abstract

Planning researchers spend much of their effort trying to reduce the amount of search required to solve problems in their domain. One possible solution is to use machine learning techniques to learn the domain-dependent information that would allow for efficient search. This paper describes DÆDALUS an implemented system that uses an incremental conceptual clustering algorithm to learn search control. We suggest modifications that would allow DÆDALUS to work in uncertain, unpredictable, or changing domains.

## 1 Introduction

In the general case, planning has been demonstrated to be intractable (Chapman, 1987); no single planning algorithm is capable of performing in a reasonable amount of time on all given problems in all given domains. This intractability is exacerbated when the domain being reasoned about is uncertain, unpredictable, or changes due to events caused by actions other than those made by the agent. Several techniques, such as hierarchical planning (Sacerdoti, 1974), have been introduced in an attempt to reduce the combinatorial search that plagues planning problems. Although useful, these techniques often require the implementer to introduce domain-dependent information, and acquiring such information is difficult.

A more recent approach is for the system to use machine learning algorithms in hopes of having the system learn the domain-dependent information itself. This can cut down the search, as well as obviating the problem of obtaining the information by hand. Researchers in machine learning have focused primarily on three methods for reducing search in planning. The first is to create macro-operators (Fikes, Hart & Nilsson, 1971; Iba, 1989) which give access to nodes in the search tree with a single operator application that previously required two or more operator applications. This method tries to reduce the search complexity from $h^b$ (where $h$ is the height of the search tree and $b$ is the average branching factor) into $1^b$ in the ideal case. The second is to acquire useful intermediate goals (Ruby & Kibler, 1989), breaking up the search into a series of $n$ short planning problems, which reduces the search complexity to $\sum_1^n (\frac{h}{n})^b$. The third method is to learn search-control knowledge (Laird et al., 1986; Minton, 1988; Allen & Langley, 1989), which reduces search by pursuing the correct path at each choice point. This method changes the computational complexity from $h^b$ to $h^1$.

In this paper we present an approach to planning, implemented as the program DÆDALUS, which uses an inductive learning technique to acquire search-control knowledge. DÆDALUS currently makes use of a simple variant of means-ends analysis to construct plans. The resulting plans are then broken up and incorporated into a probabilistic concept hierarchy, indexing them by the differences they reduce and the states to which they can be applied. Upon encountering a previously unseen problem, it retrieves a relevant plan segment and uses it to select operators for the new task. In the following sections, we discuss DÆDALUS' representation, its planning and learning components, preliminary results, and future work.

## 2 Representation and Planning in DÆDALUS

DÆDALUS acts on data structures of three types: states, problems, and operators. In general, a *state* consists of some description of the world, possibly including the internal state of the agent. In the current system, we use a simple STRIPS-like state representation (Fikes, Hart & Nilsson, 1971), with each state described as a set of objects and symbolic relations that hold among them.

A *problem* consists of an initial state and a desired state the agent wants to achieve. Each state may contain only partial descriptions of the world. One can also describe a problem in terms of the *differences* between the initial and desired state.

Most work on planning assumes that domain-operators have known preconditions and effects, and that they should be reasonably efficient (i.e. require no search to apply); DÆDALUS is no different. In the current system, operators are STRIPS-like, having preconditions, add-lists, and delete-lists. However, from this information one can derive a set of *differences* that exist between states before and after application, giving a description similar to that used for problems.

The performance system of DÆDALUS is a goal-oriented planner that draws heavily from the means-ends analysis used in GPS (Newell, Shaw, & Simon, 1960) and STRIPS (Fikes et al., 1971). The planner takes as input a problem represented as an initial state, $I$, and a description of the goal state, $G$. The goal state need not be completely defined and usually specifies a class of acceptable goal states. Having received this input, DÆDALUS computes the differences, $D$, between $I$ and $G$. The set of differences is used to encode the current goals the system needs to achieve, and, with the initial state, is used to retrieve both an operator and a partial set of bindings (described in section 3). Next the planner will attempt to apply the operator to the current problem. If the preconditions of the operator are not met, the planner recursively calls itself, passing $I$ as the initial state and the preconditions of the operator as the goal state. However, if the preconditions are matched, the operator is applied to $I$, generating a new state $N$. The planner then recursively calls itself, passing $N$ as the initial state and $G$ as the goal state. The base case of recursion occurs when $I$ satisfies all the requirements of $G$.

One major difference of our strategy from earlier methods is that it places an *ordering* on operators, rather than dividing them into relevant and irrelevant sets. One result is that it prefers operators that reduce multiple differences in the current problem, which makes it more selective than traditional techniques. More important, although DÆDALUS prefers operators that reduce problem differences, it is not restricted to this set. If none of the 'relevant' operators are successful, it falls back on operators that match none of the current differences. This gives it the potential to break out of impasses that can occur on 'trick problems'.

The planning system produces a *derivational trace* (Carbonell, 1986) that stores the reasons for each step in the operator sequence. This trace consists of a binary tree of problems and subproblems, with the original task as the top node and with trivial (one-step) subproblems as the terminal nodes. Each node (problem) in the derivational trace is described by differences between its initial and final state, by the initial state, and by the instantiated operator that was selected to solve the problem. It is the derivational trace that is analyzed by the learning mechanism to improve future performance.

## 3 Memory and Learning in DÆDALUS

The organization of memory and learning mechanism is based on Fisher's COBWEB (1986). In this section, we start out with a basic description of COBWEB, describe our additions to the basic mechanism, and relate how the memory influences the planning system.

### 3.1 A Review of COBWEB

COBWEB represents each instance as a set of nominal attribute-value pairs, and it summarizes these instances in a hierarchy of *probabilistic concepts* (Smith & Medin, 1981). Each concept $C_k$ is described as a set of attributes $A_i$ and their possible values $V_{ij}$, along with the conditional probability $P(A_i = V_{ij}|C_k)$ that a value will occur in an instance of a concept. The system also stores the overall probability of each concept, $P(C_k)$. COBWEB uses this information in its evaluation function – *category utility* (Gluck & Corter, 1985) – which favors high intra-class similarity and high inter-class differences.

COBWEB integrates classification and learning, sorting each instance through its concept hierarchy and simultaneously updating memory. Upon encountering a new instance $I$, the system incorporates it into the root of the existing hierarchy and then recursively compares the instance with each new partition as it descends the tree. At a node $N$, it considers incorporating the instance into each child of $N$, as well as creating a new singleton class, and evaluates each resulting partition with category utility. If the evaluation function prefers adding the instance to an existing concept, COBWEB modifies the concept's probability and the conditional probabilities for its attribute values and then recurses to the children of that concept. If the system decides to place the instance into a new class, it creates a new child of the current parent node, and the classification process halts. COBWEB also incorporates two bidirectional operators, splitting (which destroys an existing class) and merging (which creates a new class out of two existing classes), to mitigate sensitivities to instance orderings.

### 3.2 Organization and Retrieval in DÆDALUS

In DÆDALUS, the nodes in the concept hierarchy consist of two parts: *predicates* and *operators*. The predicates of the node correspond to the attributes of a COBWEB hierarchy. They are made up of the same predicates the planner uses to describe the states and the differences. Each predicate has two values associated with it, *present* or *absent*. The value *present* refers to the number of situations that have passed through that node which exhibit that predicate. Conversely, *absent* refers to the number of situations that have passed through that node which do not exhibit that predicate. Note that the negation of a predicate is not the same as its absence, negated predicates are needed to describe the preconditions of some operators, and consequently will have present and absent probabilities associated with it as well. The operators of a node correspond to the domain-operators; the arguments of which correspond to the pattern-matching variables found in the node's predicates.

Initially, the DÆDALUS is given a hierarchy, like the one depicted in figure 1, which contains the domain operators with the differences they reduce and a partial

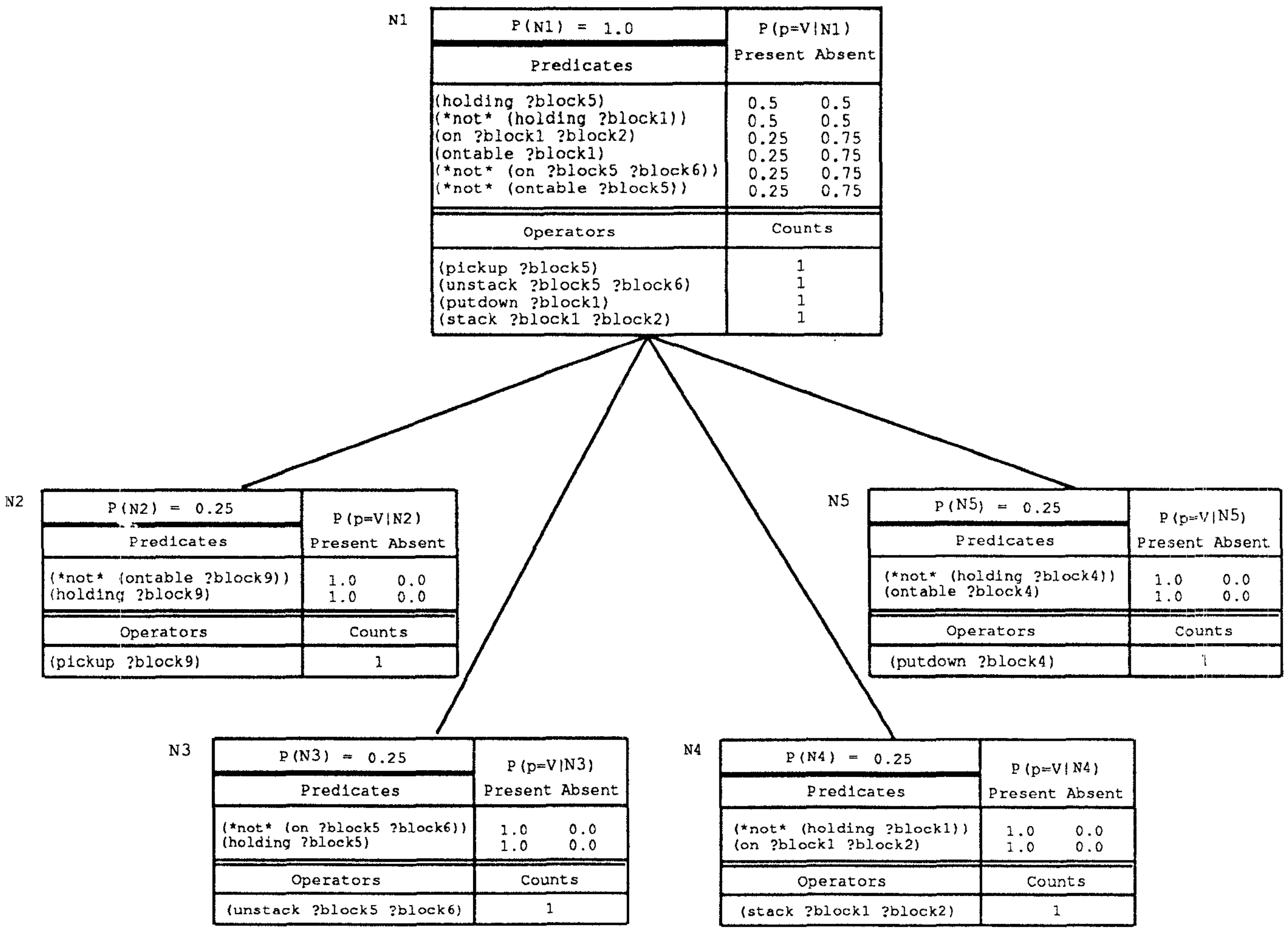

N1
P(N1) = 1.0
P(p=V|N1)
Predicates
Present Absent
(holding ?block5) 0.5 0.5
(*not* (holding ?block1)) 0.5 0.5
(on ?block1 ?block2) 0.25 0.75
(ontable ?block1) 0.25 0.75
(*not* (on ?block5 ?block6)) 0.25 0.75
(*not* (ontable ?block5)) 0.25 0.75
Operators
Counts
(pickup ?block5) 1
(unstack ?block5 ?block6) 1
(putdown ?block1) 1
(stack ?block1 ?block2) 1
N2
P(N2) = 0.25
P(p=V|N2)
Predicates
Present Absent
(*not* (ontable ?block9)) 1.0 0.0
(holding ?block9) 1.0 0.0
Operators
Counts
(pickup ?block9) 1
N5
P(N5) = 0.25
P(p=V|N5)
Predicates
Present Absent
(*not* (holding ?block4)) 1.0 0.0
(ontable ?block4) 1.0 0.0
Operators
Counts
(putdown ?block4) 1
N3
P(N3) = 0.25
P(p=V|N3)
Predicates
Present Absent
(*not* (on ?block5 ?block6)) 1.0 0.0
(holding ?block5) 1.0 0.0
Operators
Counts
(unstack ?block5 ?block6) 1
N4
P(N4) = 0.25
P(p=V|N4)
Predicates
Present Absent
(*not* (holding ?block1)) 1.0 0.0
(on ?block1 ?block2) 1.0 0.0
Operators
Counts
(stack ?block1 ?block2) 1

*Figure 1:* Initial hierarchy containing four operators

state description representing the class of states to which they can be applied (e.g., the operator's preconditions)[1]. The memory system builds a hierarchy in which each of the terminal nodes contains the name of an operator, its preconditions, and its differences. The internal nodes correspond to classes of operators that have some overlap in their differences, preconditions, or both. The end result is something that plays a role similar to that of a difference table (Newell et al., 1960).

When DÆDALUS is running, the planning system passes the memory system initial state $I$ and differences $D$. The memory system takes these two inputs and combines them, forming a structure called a *situation*. The concept hierarchy is then used to determine which class of operator is most applicable to the current situation. In other words, it tries to find the node in the hierarchy whose predicates have the greatest amount of overlap with the situation. The memory system uses category utility to determine the most applicable operator, but it does not change the concept hierarchy in any way — no learning occurs at this stage of the process.

Once a node is found, the memory returns the operator associated with that node and the binding generated by matching the situation with the predicates of the node. If the node selected happens to be an internal node, the most frequent operator found in that class is returned, else it is just the operator associated with the node.

Since the planner may backtrack and ask the memory system for a different operator instance, the process of incorporating situations into nodes excludes those incorporation which would give rise to an operator instantiation that the planner has already rejected. It is in this way that the memory system imparts an ordering on operator instances that includes not only operators relevant to the current differences, but also those that have no apparent relation to the current differences.

[1]For simplicity of presentation, we have shown only the differences each operator reduces and excluded the preconditions.

### 3.3 Storing Successful Plans

Section 2 mentioned that the planner returns its plans in a structure called a derivational trace, which records the operators, the situations, and the sub-problems generated by each operator. The learning system uses this example of a successful plan to improve the accuracy of operator prediction. The derivational trace is first broken up into its situation/operator pairs. Each of these pairs is then reclassified and incorporated into the concept hierarchy, and the hierarchy is updated and modified by this incorporation process. The situation/operator pairs are used to reinforce existing classes, as well as creating new operator classes. The reinforced classes reflect those situations where the hierarchy correctly predicted the right operator. The new operator classes reflect the correct choice in those situations where the planner, in creating the plan, had made and error and had to backtrack. The hierarchy has an encoding of the correct operator application. This is how the DÆDALUS learns search control knowledge.

In most planning systems, the order in which the the system resolves the goals of a problem have significant effect on the amount of search performed. Nonlinear planning was developed to address this problem. DÆDALUS has the unusual property of being a linear planner that is unaffected by goal ordering. If DÆDALUS is presented with such a problem, it will solve it only after extensive search. Once it has solved the problem, it will not have trouble with it again, even if the goals are presented in a different order or the correct operator has no relation to any of the goals. This is a direct result of the learning system used.

## 4 Preliminary Results

We have carried an initial experiment to evaluate DÆDALUS learning. Figures 2 and 3 present results averaged over 5 trials, each trial containing 10 problems. The problems come from the blocks-world domain and consist of a number of blocks in a randomly determined initial state, with one or two randomly chosen conditions in the goal state. The problems are filtered to insure that the initial state does not satisfy the goal conditions. At the start of each learning trial, DÆDALUS is presented with the concept hierarchy presented in figure 1. The system incorporates the derivational trace of the solution of each problem into the concept hierarchy before solving the next problem.

The independent variable of both graphs tell the number of problems solved, which is a rough measure on the amount of information DÆDALUS has had to learn. The dependent variable is a ratio of the number of nodes in the returned solution path, over the number of nodes searched in finding the solution. The best value for this ratio is 1.0 — the case in which the correct path was taken at each choice point in the search tree. The worst value is an asymptotic approach to 0 — the case in which the least correct path was taken at each choice point in the search tree.

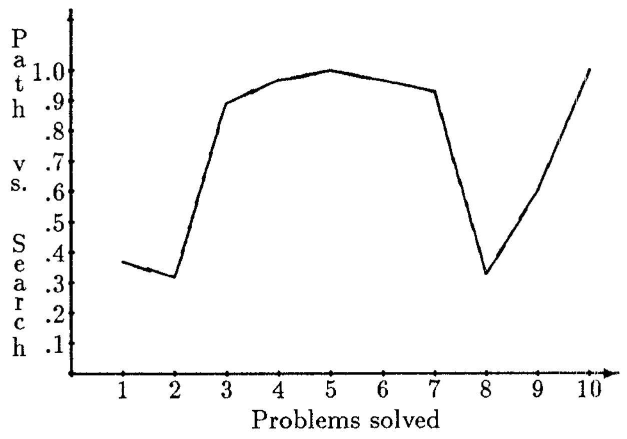


*Figure 2:* Learning 2-block problems

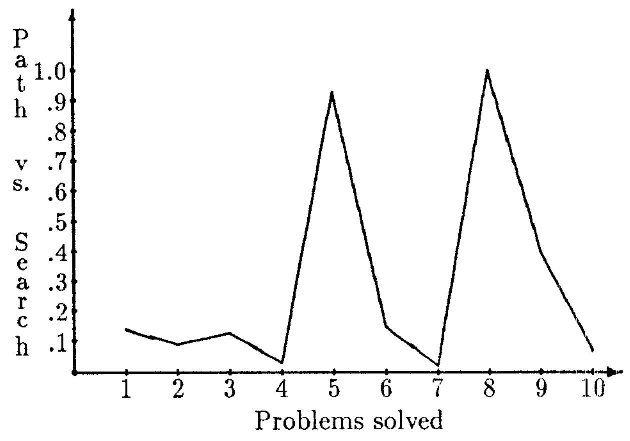


*Figure 3:* Learning 3-block problems

Figure 2 shows DÆDALUS' performance on two-block problems. Here, DÆDALUS quickly learns most of the knowledge necessary to solve most of the problems presented. The dip at problem 8 was caused by an outlier, with the other values hovering between .9 and 1.0.

Figure 3 graphs DÆDALUS's performance on three-block problems. DÆDALUS had considerably more trouble learning these problems. We believe this is due to a combination of factors. First, the planner we have implemented is not very efficient, and will return sub-optimal plans even in the three-blocks scenario. The system incorporates these non-optimal plans which effects the performance on later problems. The hierarchy is particularly vulnerable to this type of problem early on. However, the control structure of the memory hierarchy should allow it recover after seeing future examples. Another option, one used by Minton (1988), is to train DÆDALUS on easy problems first, and once it has mastered the simple problems, train it on progressively harder problems.

DÆDALUS' current implementation also suffers from one aspect of the *utility problem* (Minton, 1988). Since the system organizes its knowledge in a hierarchy, the amount of learned information it must search should be proportional its log of situations stored. However, the new knowledge does require matching to be accessed, and it is this added match time that is slowing DÆDALUS down. We are currently looking into strategies to minimize the match required.

## 5 Future Work

We intend to use DÆDALUS as the controlling system of an autonomous agent, but to realize this goal we must first address several issues. Currently, DÆDALUS merely plans, lacking any mechanism for executing its plans. To have the system create a plan and then send it to an executor seems somewhat unsatisfying — an interleaving of the two seems preferable, and means-ends analysis seems particularly amenable to an interleaving approach. Whenever DÆDALUS has an operator whose preconditions are met, it would apply it in the external world, and would sit and plan if it had to satisfy the preconditions of some operator. Since the system creates differences with each invocation of the algorithm, it should be able to correct its own plans — if an applied operator affects the world in an unpredicted manner or if some external event should change the world. The system would query the world to get its information about the new state, and not depend on the effects of the operator on the internal representation.

In addition, the probabilistic nature of the memory structure should be advantagious in unpredictable domains. For example, suppose that a in an unpredictable domain, operator $A$ reduces a particular difference $D$ 70% of the time, and operator $B$ reduces $D$ 30% of the time. The learning mechanism would be able to learn that $A$ was the better guess for reducing $D$ and modify its predictions accordingly. The planner would then receive the optimal ordering of $A$ and $B$. Furthermore, if the relative effectiveness of $A$ and $B$ at reducing $D$ should change, the concept hierarchy would change to reflect this, and once again provide the optimal ordering of $A$ and $B$.

## References

Allen, J. A., & Langley, P. (1989). Using concept hierarchies to organize plan knowledge. *Proceedings of the sixth international workshop on machine learning* (pp. 229–231). Morgan Kaufmann: Ithaca, NY.

Carbonell, J. G. (1986). Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2). Los Altos, CA: Morgan Kaufmann.

Chapman, D. (1987). Planning for conjunctive goals. *Artificial Intelligence*, *32*, 333-377.

Fikes, R. E., Hart, P. E., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *2*, 189–208.

Fisher, D. H. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, *2*, 139–172.

Gluck, M., & Corter, J. (1985). Information, uncertainty and the utility of categories. *Proceedings of the Seventh Annual Conference of the Cognitive Science Society* (pp. 283–287). Irvine, CA: Lawrence Erlbaum.

Iba, G. A. (1989). A heuristic approach to the discovery of macro-operators. *Machine Learning*, *3*, 285–318.

Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986). Chunking in SOAR: The anatomy of a general learning mechanism. *Machine Learning*, *1*, 11–46.

Minton, S. (1988). Quantitative results concerning the utility of explanation-based learning. *Seventh National Conference on Artificial Intelligence* (pp. 564–569). Morgan Kaufmann: St. Paul, MN.

Newell, A., Shaw, J. C., & Simon, H. A., (1960). A variety of intelligent learning in a general problem solver. In Yovits & Cameron (Eds.), *Self organizing systems*. Pergamon Press: New York.

Ruby, D., & Kibler, D. (1989). Learning to plan in complex domains. *Proceedings or the sixth international workshop on machine learning* (pp. 180–182). Morgan Kaufmann: Ithaca, NY.

Sacerdoti, E. D. (1974) Planning in a Hierarchy of Abstraction Spaces. *Artificial Intelligence*, *5* 115-135.

Smith, E., & Medin, D. (1981). *Categories and concepts.* Cambridge, MA: Harvard University Press.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, *3*, 10-43.

# Planning and Understanding: Revisited[*][†]

**Richard Alterman**
**Roland Zito-Wolf**
Computer Science Department
Brandeis University

## Introduction

Much recent planning research has been motivated by the recognition that planners that rely on the traditional assumptions of complete knowledge and certainty of prediction do not scale up to significantly large or complex domains. This is due to the complexity of the computation (Chapman 1987), the unboundedness of potentially relevant world knowledge, and the problems of unanticipatable events. Broadly, there have been two approaches to these issues:

1. Re-planning or partial planning. Planning and acting can be interleaved (McDermott 1978). The plan may be partially or completely elaborated before execution begins. Planning may be resumed under a variety of guises: plan elaboration (Georgeff & Lansky 1987, Firby 1987), plan refitting and adaptation (Alterman 88, Kambhampati & Hendler 1989), plan repair (Wilkins 1988, Hammond 1987, Simmons 1987), experimentation (Shrager 1987), or opportunistic action (Hammond 1989, Birnbaum 1986, Hayes-Roth & Hayes-Roth 1979).
2. Reactivity. Knowledge can be organized so that planning is not required; sufficient information will be available at run time to select the appropriate action. Agre and Chapman (1987, Agre 1988) develop a theory of *situated activity* which exploits regularities in the agent's interaction with the world. Schoppers (1987) converts the plan into a *universal plan* – a set of situation-action rules – that covers the (anticipatable) contingencies. Brooks (1986) and Kaebling (1986) have demonstrated systems that implement complex behavior using simple reactive machinery.

While these approaches remove the artificial separation of planning from acting, there are other factors that play a critical role in building a mechanism that is reactive and can deal with uncertainty. In this paper we argue that understanding plays a crucial role in dealing with uncertainty and, in some planning domains, is the primary source of improvisation.

## The role of understanding

The tradition in symbolic AI is to treat understanding as a problem of representation. A program is said to have 'understood' an input, whether it be experiential, perceptual, or textual, to the degree to which it can reason about that representation/understanding. Two properties of understanding/representation that are of interest here are correspondence and coherence. Correspondence means that there must be mappings from concepts in semantic memory to the input stream. Coherence means that the program assigns internally consistent chunks of semantic memory to sets of correspondences. In this paper we treat 'interpretation' as a species of understanding, in which understanding is framed from a particular vantage point.

Previous work on the relationship between planning and understanding has emphasized the following:

**Shared Knowledge** Wilensky argued that planning and understanding share knowledge (Wilensky 1983). This includes not only first-order knowledge of particular plans, or a hierarchy of plans, but also second-order knowledge about the planning process itself. His focus was on the development of various kinds of 'meta-goals' and 'meta-plans'; from this work began to emerge a cognitive model of the planner as a rational agent.

**Framing and Elaborating the Problem** In domains of subjective planning the understanding frames or elaborates the planning process (e.g. MEDIATOR: Kolodner & Simpson 1989; HYPO: Ashley & Rissland 1987; POLITICS: Carbonell 1981). For example, HYPO, given a case description, posts several interpretations (elaborations) of the case, from which it builds an argument. The cases that HYPO

[*] *Planning and Understanding:* Wilensky, 1983.

[†]This work was supported in part by the Defense Advanced Research Projects Agency, administered by the U.S. Air Force Office of Scientific Research under contract #F49620-88-C-0058.

retrieves gives it a framework for deciding what are the important legal facets of the new case. The elaboration that comes from understanding the situation can also be a source of improvisation: I am walking down the street in front of the liquor store on Main Street. I see a pedestrian hit by a car and interpret the liquor store as a place that has a telephone from which an ambulance can be called.

**Plan Recognition** The recognition of the plans of other agents is a form of understanding. Plan recognition techniques have been developed in the context of discourse (e.g. Mayfield 1989, Allen 1983), text understanding (e.g., Wilensky 1978), and user modeling (e.g. Genesereth 1982).

Our position is that in some cases understanding not only supports the planning process, but drives it.

> A planner 'plans by understanding' if its actions, during the period of engagement, are driven by the assignment of coherent meanings to the elements of the situation.

Under certain circumstances the planner can assume that the situation will be understandable — because it has been designed to be so — and can therefore use its understanding as the source of improvisation. Examples of problems where these circumstances arise are in various commonsense planning situations (e.g. riding public transportation) or in the usage of mechanical devices (e.g. setting the time on a new VCR).

Planning as situation understanding is a workable strategy of improvisation under circumstances where a large community of agents, who share a set of concepts, must communicate procedures to one another through situations or artifacts. In these kinds of situations, the agent can reasonably assume that the procedure to be detected was *designed* to be accessible , i.e., it is structured to minimize the conscious mental processing involved in performing it (Norman 1988, pp. 124-127).

The criteria of 'understandability' also acknowledges the fact that most designs do not arise *de novo*, but are derived from the designer's own library of known procedures. For example, in the last ten years large numbers of videotape-rental stores have opened, with varying procedures for tape storage, display, checkout, return, and theft control. These procedures presumably arose from an interpretation of the store from the perspective of other, already worked out, procedures such as checking out a book from a library. These procedures have gradually been modified over time: tapes are stored differently, membership alternatives change, and so forth. Yet whenever I go to a VCR store that uses unfamiliar procedures I can understand and act in this situation because those procedures were ultimately derived from other procedures that are shared in the culture.

## FLOABN and Instructions

FLOABN (For Lack Of A Better Name) is an implementation in progress of a commonsense planner that reasons about the usage of mechanical devices by constructing an interpretation. When action and interpretation break down, FLOABN reads instructions. The current domain of FLOABN is to learn to use a series of telephone devices, gradually building up its memory of procedures after each encounter. We are also using FLOABN to explore the domain of time setting devices. Some examples of problems that FLOABN works on are making a telephone call from a phone on an airplane and figuring out how to set the time on a VCR.

Some central features of FLOABN are:

**memory** FLOABN includes both semantic and case memory. For the telephone domain, its case memory contains routines for using telephones in differing contexts (at home, at work, pay phone). Semantic memory includes generalizations over these routines, the steps of the routines, and ways of chunking the steps. It also includes generalizations over the device and its components.

**operational level** Rather than working from an abstract level and refining, the planner starts with a concrete routine as its 'reference point' (Lakoff 1987, Rosch 1981) and backs away from the details only when the situation merits, i.e., FLOABN is a case-based planner (Hammond 1986, Alterman 1986, Kolodner, Simpson & Sycara-Cyranski 1985, Carbonell 1983).

**correspondence and coherence** Planning and adaptation are guided by the twin constraints of correspondence and coherence. FLOABN draws on two earlier models of correspondence and coherence: NEXUS (Alterman 1989), which used a spreading activation-like mechanism to determine coherency, and PLEXUS (Alterman 1988), which used a case as an anchor to guide the construction of coherent correspondences in the service of adaptation.

**engagement** By delaying situation matching until actual engagement, FLOABN has access to features that could not have been anticipated and a richer situation matrix to drive interpretation.

The heart of the system is the adaptive planner (Alterman 1988). The adaptive planner works by retrieving from its memory of plans a routine that seems to match the situation-at-hand. It then adapts that plan (improvises) during the period of engagement, as a function of the interpretation it constructs of the situation. When adaptation fails, FLOABN falls back on instructions. In the domain of mechanical devices, some form of instructions are usually available. They may be printed on the device (as on a pay phone) or collected separately (as in an instruction manual)

Instructions are difficult to understand in the abstract because they are *schematic*. Outside of a context of use, the planner can grasp only the general sense of what the instructions mean and the operations they depict. Therefore FLOABN does not plan primarily from the instructions (it might skim them) but first engages in the activity. When a situation arises to which it can not adapt, FLOABN refers to the instructions. (This is in accordance with human instruction usage (LeFevre and Dixon 1986).) The situation of difficulty provides a context, a backdrop, against which the instructions can be made concrete. FLOABN's understanding/representation provides the detail necessary to operationalize (Mostow 1981) the instructions. For FLOABN, operationalization occurs as a result of an interpretive process.

For example, FLOABN, in a simulated situation, attempts to make a telephone call from an airplane for the first time. FLOABN initially interprets the situation from the context of its pay-phone routine and during the course of action it gradually adapts that interpretation, maintaining correspondence and coherence, as it works way through the situation. When FLOABN gets to the step INSERT-COIN in its pay-telephone routine, and it can find no coin slot in the airplane telephone, it forms a request for an instruction related to the concept of 'payment'. An instruction is found: *Insert credit card face up with card name to the right.* This instruction contains three kinds of information. One relates the instruction to the planner's ongoing understanding (the reference to credit card), a second selects a procedure associated with cards (insert), and the third qualifies the action (face up with card name to the right). In this case, FLOABN attaches the first statement to its ongoing understanding, because credit card is coherent with payment, and determines that payment is to be made with a credit card. The action INSERT is attached to actions associated with credit cards, and the procedure of insertion of cards is qualified, in this context, according to constraints provided by the instruction.

After a planning episode, a learner, from the framework of a given interpretation, can explain the episode, and hence acquire a new procedure (e.g. EGGS, Mooney 1988). In the case of FLOABN, after a successful planning episode, the interpretation that FLOABN constructed is used to generalize the base plan. Adaptations are incorporated into memory by annotating the old plan with discrimination points which indicate aspects of the plan about which run-time decisions must be made. Various concepts associated with the plan may be generalized as well. For example, having successfully interpreted a touch-tone phone as a dial-phone, with a keyboard replacing the rotary dial, FLOABN generalizes its existing telephone-object category into a conjunctive category encompassing both device types, and chunks the subprocedures associated with the two different device features (touch-tone and rotary dial).

## Discussion

Suchman suggests the following example of a situated activity (Suchman 1987, p. 52):

> So, for example, in planning to run a series of rapids in a canoe, one is very likely to sit for a while above the falls and plan one's descent.

Her claim is that however detailed the plan is it falls short of the various contingencies that arise as you navigate through the rapids. The planning you did at the outset provided orientation but the bulk of the activity was composed of reactive embodied skills.

An extreme position might argue that situated activities are entirely reactive, and that any planning or understanding that seemed to be associated with the activity was actually *post hoc*. The idea is that what from the vantage point of the actor is a sequence of situated reactions, could be interpreted by an observer or in retrospect as being planned.

The understanding process seems to be keyed to those situations where reactivity breaks down and other forms of improvisation become needed. For commonsense domains, such as the usage of mechanical devices, 'understanding' is an important mode of improvisation if reaction breaks down. When routine proves insufficient, one tries to improvise by understanding the situation, looking for new sources of guidance.

We take the prevalence of instructions as one piece of evidence that in many cases understanding occurs before action (and is not merely imposed *post hoc*). Most instructions (written or verbal) omit an enormous amount of presumably shared detail, but since the planner shares general background and an understanding of the situation up to that point, it can fill them in itself. To do so, to operationalize the instructions, the correspondence between the situation and instructions needs to be elaborated: the situation needs to be understood in terms of the instructions before performance can resume.

## References

[1] Philip E. Agre. The dynamic structure of everyday life. Technical Report TR 1085, MIT Artificial Intelligence Laboratory, 1988.

[2] Philip E. Agre and Davidd Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of AAAI-87*, pages 268–272, 1987.

[3] James Allen. Recognizing intentions from natural language utterances. In Michael Brady and Robert C. Berwick, editors, *Computational Models of Discourse*, pages 107–166. MIT Press, 1983.

[4] Richard Alterman. An adaptive planner. In *Proceedings of AAAI-86*, pages 65–69, 1986.

[5] Richard Alterman. Adaptive planning. *Cognitive Science Journal*, 12:393–421, 1988.

[6] Richard Alterman. Event concept coherence. In David Waltz, editor, *Advances in Natural Language Processing*, pages 57–87. Lawerence Erlbaum Associates, 1989.

[7] Kenneth D. Ashley and Edwina L. Rissland. Compare and contrast, a test of expertise. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 273–278, 1987.

[8] Lawrence Birnbaum. Integrated processes in planning and understanding. Technical Report CSD/RR 489, Yale University, 1986.

[9] Rodney A. Brooks. A robust layered control system for a mobile robot. Technical Report AI Memo 864, MIT Artificial Intelligence Laboratory, 1985.

[10] Jaime G. Carbonell. Counterplanning: A strategy-based model of adversary planning in real-world situations. *Artificial Intelligence*, 16:295–329, 1981.

[11] Jamie Carbonell. Derivational analogy and its role in problem solving. In *Proceedings of AAAI-83*, 1983.

[12] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–377, 1987.

[13] R. James Firby. An investigation into reactive planning in complex domains. In *Proceedings of AAAI-87*, pages 202–206, 1987.

[14] Michael R. Genesereth. The role of plans in intelligent teaching systems. In D. Sleeman and J. S. Brown, editors, *Intelligent Tutoring Systems*, pages 137–155. Academic Press, 1982.

[15] Kristian J. Hammond. Chef: A model of case-based planning. In *Proceedings of AAAI-86*, pages 267–271, 1986.

[16] Kristian J. Hammond. Explaining and repairing plans that fail. In *Proceedings of IJCAI-87*, pages 109–114, 1987.

[17] Kristian J. Hammond. Opportunistic memory. In *Proceedings of IJCAI-89*, pages 504–510, 1989.

[18] B. Hayes-Roth and F. Hayes-Roth. A cognitive model of planning. *Cognitive Science*, pages 275–310, 1979.

[19] L. P. Kaelbling. An architecture for intelligent reactive systems. In *Reasoning About Actions and Plans: Proceedings of the 1986 Conference*. Morgan Kaufmann, Los Altos, California, 1987.

[20] Subbarao Kambhampati and James A. Hendler. Control of refitting furing plan reuse. In *Proceedings of IJCAI-89*, pages 943–948, 1989.

[21] Janet Kolodner, R. Simpson, and K. Sycara-Cyranski. A process model of case-based reasoning in problem solving. In *Proceedings of IJCAI-85*, pages 284–290, 1985.

[22] Janet L. Kolodner and Robert L. Simpson. The MEDIATOR: Analysis of an early case-based problem solver. *Cognitive Science*, 13:507–549, 1989.

[23] George Lakoff. *Women, Fire, and Dangerous Things*. University of Chicago Press, 1987.

[24] Jo-Anne LeFevre and Peter Dixon. Do written instructions need examples? *Cognition and Instruction*, 3(1):1–30, 1986.

[25] James Mayfield. Goal analysis: Plan recognition in dialogue systems. Technical Report UCB 89/521, University of California at Berkeley, 1989.

[26] Drew McDermott. Planning and acting. *Cognitive Science*, 2:71–109, 1978.

[27] Raymond Mooney. A general explanation-based learning mechanism and its application to narrative understanding. Technical Report Technical Report AITR88-66, University of Texas at Austin, 1988.

[28] David Jack Mostow. Machine transformation of advice into heuristic search procedure. In Ryszard Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning, Volume 1*, pages 367–403. Tioga Publishing Company, 1983.

[29] Donald A. Norman. *The Psychology of Everyday Things*. Basic Books, 1988.

[30] Michael P.Georgeff and Amy K. Lansky. Reactive reasoning and planning. In *Proceedings of AAAI-87*, pages 677–682, 1987.

[31] E. Rosch. Prototype classification and logical classification: The two systems. Paper presented at a meeting of the Jean Piaget Sciety, Philadelphia, 1981.

[32] M. J. Schoppers. Universal plans for reactive robots in unpredictable environments. In *Proceedings of IJCAI-87*, pages 1039–1046, 1987.

[33] Jeff Shrager. Theory change via view application in instructionless learning. *Machine Learning*, 2:247–276, 1987.

[34] Reid Simmons and Randall Davis. Generate, test and debug: Comnbining associational rules and causal models. In *Proceedings of IJCAI-87*, pages 1071–1078, 1987.

[35] Lucy A. Suchman. *Plans and Situated Actions*. Cambridge University Press, 1987.

[36] Robert Wilensky. *Understanding goal-based stories*. PhD thesis, Yale University, 1978.

[37] Robert Wilensky. *Planning and Understanding*. Addison-Wesley, 1983.

[38] David E. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm*. Morgan Kaufmann, 1988.

# Learning Approximation–based Uncertainty–tolerant Plans

**Scott W. Bennett**

Beckman Institute for Advanced Science and Technology, University of Illinois at Urbana–Champaign
405 North Mathews Avenue, Urbana, IL 61801
Internet: bennett@rtd2.cs.uiuc.edu

## 1. Introduction

Most learning and planning systems to date have functioned in isolation from the real world. They work with a simplified representation for the world, usually measuring success by the ability to efficiently produce plans which function well under the assumptions of that representation. This work has produced many invaluable techniques for use in learning and planning. However, one component missing from most of these systems is a mechanism for reconciling their performance in the simplified world with the real world. Any model of real–world behavior will necessarily have discrepancies with how the real world actually behaves.

Explanation–based learning has shown promise in robotics. In Segre's ARMS system, knowledge about the control of a robotic manipulator in conjunction with geometric object knowledge permitted learning assembly plans from observation [Segre87]. Explanation–based learning permits general plans to be learned from a single example [DeJong86, Mitchell86]. In robotics, a sequence of robot control primitives is used as the example. Attempting to use ARMS to control a real robotic manipulator often resulted in failures due to discrepancies with the real world. This highlighted the need for a mechanism which can adapt a system's model of the world to the real–world environment.

We are currently developing a system called GRASPER to illustrate the use of explanation–based learning in the real world [Bennett89a]. It seeks to learn and execute real–world plans tractably in the presence of uncertainty. The system is being tested in a robotic grasping domain where it can act on the world through control of a robotic manipulator and can acquire data with a visual system and position and force sensors associated with the manipulator. We are testing system performance at grasping relatively flat pieces from a puzzle for young children. Grasping complex shapes such as these is a difficult ongoing robotics research area and provides an ideal testbed for our learning algorithms. The system employs explicit approximations in the domain theory. It is the tuning of these approximations with experience which is the key to the system. This paper focuses on how approximate rules are tuned over time to increase the uncertainty tolerance of learned plans. First, our approximation terminology is introduced. The approximation tuning algorithm is presented next. The algorithm is then employed on a robotics grasping example.

## 2. Approximations

An approximation has two important defining features:

*Assumability*

An approximation must make some statement about the world based not on logical proof but on conjecture.

*Tunability*

The approximation must provide a method by which it can be tuned as the system acquires new knowledge and/or its goals change. The tuning method should allow adjustment of continuous parameters of the approximation to decrease its error with respect to the true world situation. The tuning method may accept as input new knowledge (obtained from sensor readings) which facilitate this adjustment.

Assumability gives an approximation its efficiency and tractability advantage. This provides a justification that further reasoning need not be done. Tunability indicates that our use of the term approximation requires that they be a function of continuously tunable parameters. By making approximations explicit, rather than implicit as in many AI systems, failures resulting from the approximations can be traced back to the inadequate approximations. In much of the work on approximation, the focus is on how to make the approximations initially. This is an important issue in using approximations to improve the efficiency of one's knowledge. But in using approximations to deal with real–world uncertainties, the goal is to improve the uncertainty tolerance of one's knowledge. The unfortunate reality is that everything a system senses from the outside world is in some sense approximate already. To put this in perspective, in the following two subsections, we discuss and distinguish between *data approximations* and *rule approximations.*

### 2.1. Data Approximations

Data approximations involve representations for measures sensed from the real world. The raw sensor readings obtained by the system are *external approximations.* That is, they can only be tuned through interaction with the world. If a range sensor returned the distance to an object, that distance would be externally approximate. However, one could imagine performing some further actions in the world, such as using tactile sensors to feel first contact with the object, so as to tune that initial approximation for distance to the object. In contrast, an *internal approximation* can be tuned by the system's own reasoning alone without acting on the world. A common type of internal data approximation employed by our system is to reduce the complexity of visually sensed 2–D object contours. Such contours involve hundreds of points and make it difficult and slow to devise grasping points. The internal data approximation currently employed reduces the contour to a n–gon with n much less than the number of sensed contour points.

Data approximations, both external and internal, are currently pre–selected for the domain. The tuning of such approximations from their initial values is the key problem here. Our use of internal data approximations is for efficiency not uncertainty tolerance.

## 2.2. Rule Approximations

Rule approximations are employed when the system plans how it can act on the world to achieve some goal. They affect the way the system interacts with the world. Consequently, these approximations are useful for building uncertainty tolerance into a plan. They are always internal approximations, capable of being controlled by the system. Approximation techniques, such as those in use by Keller [Keller87], which drop rule preconditions, are like our rule approximations but in a discrete, not a continuous, sense. Their approach to improving efficiency of rules through approximations is complementary to ours as both efficiency and uncertainty tolerance are important aspects of a system's real–world performance.[1]

This paper focuses on rule approximations for the purpose of improving the uncertainty tolerance of a plan. Here, the rule approximations involve a choice for continuous parameters whose adaptability to the environment is desired. The initial approximate rules include a declaration of these continuous approximate quantities as well as a set of predicates (antecedents to the approximate rules) which calculate the initial values. These approximate rules are pre–defined as part of the domain knowledge. We present an algorithm for recognizing which approximations, among the declared rule approximations, to tune on failure and how to tune them.

## 3. A General Tuning Algorithm

Since rule approximations involve a selection of values for various continuous parameters, what is needed is a way of reasoning about how quantities influence each other within an explanation–based rule. The major innovation in our tuning algorithm is conversion of the quantitative portion of an EBL rule into a qualitative model of quantity influences. Qualitative tuning explanations can then be produced from the qualitative model as to how to increase the probability of success of a certain predicate, given its explanation, through modification of tunable quantities from the rule approximations.

| Step 1 . Produce the failing explanation for which a remedy is sought |
|---|

In the real–world version of the GRASPER system, monitored actions executed as part of the plan have explanations associated with them as to why their expectations should hold. Therefore, at the time of an expectation failure, it is that explanation which is the starting point for approximation tuning. However, we seek to reduce this explanation further by eliminating those aspects which didn't specifically relate to the observed failure. For example, a monitored move command may justify its expectation to sense no external forces during the move by an explanation which guarantees there will be no collisions with nearby objects. Should a failure occur, indicating that the no–collision explanation must have some error in it, we would reduce it to the no–collision explanation dealing with the object most likely to have collided with the gripper. In this way, the complexity of the tuning explanation is reduced. It would be inefficient to entertain other possible approximation tunings to prevent collisions with the other objects when they likely didn't cause the observed failure.

1. For a model of real–world plan operationality see [Bennett89b].

| Step 2 . Establish the existing qualitative relationships between quantities in the failing explanation |
|---|

To make analysis of the failing explanation more straight–forward, we form the general rule from the failing explanation using the EGGS algorithm [Mooney86]. This rule is guaranteed to express all the same relationships between variable quantities as were expressed in the original failed explanation. In order to reason about the influences of the adjustment of various quantities on the probability of success of the failed explanation, the possible relationships between quantities in the rule need to be asserted. All rule antecedents and consequents which deal with quantities are asserted as qualitative relations. These declarations in combination with qualitative predicate definitions, given in a set of domain independent qualitative rules, yield a set of qualitative proportionalities. For example, suppose the predicate *(dif ?a27 ?b32 ?c15)* were an antecedent to the rule being analyzed. This would be asserted as the qualitative relation *(qrelation (dif a27 b32 c15))*. That is, the general variables *?a27*, *?b32*, and *?c15* will be treated as quantities. One of the several qualitative definition rules for *dif*, the system's subtraction predicate, is shown below. This qualitative definition rule can be used to show

```
(rule  :form
       (Q+ ?r ?q1)
       :ants
       (qrelation (dif ?q1 ?q2 ?r))
```

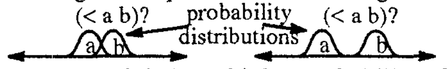
*(Q+ c15 a27)*, that the magnitude of the quantity *a27* has a direct positive influence on *c15*, the result of the subtraction operation. A set of these rules for each predicate employed by the system provides a mechanism for establishing relations between quantities in an explanation.

Often the system will not be able to conclude all the necessary qualitative proportionalities purely from qualitative predicate definitions. For example, it may not be possible to establish that a certainty quantity is greater than another one in general in the rule. This relationship may be needed to show a qualitative proportionality which has a bearing on the ability to produce a tuning explanation. In these cases, the specific binding list for the original explanation is consulted and if the relationship holds there, it is asserted as a qualitative relation between the two quantities.

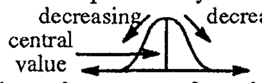
Among the quantities identified in the rule are two special types: *data approximate quantities* which are part of data approximations and *tunable quantities* which come from rule approximations. Both types of special quantities are identified through use of domain specific approximation declarations. Data approximate quantities are not controllable by the system as are the tunable quantities. They represent various measured values about which there exists a certain amount of uncertainty.

| Step 3 . Produce an explanation for how to positively influence the probability of success of the root predicate to the failing explanation |
|---|

Quantitative predicates employed by the system have one of two basic intents. Either they are *calculation predicates*, whose purpose is to compute some value (e.g. the *dif* function discussed earlier), or they are *test predicates*, which are designed to fail for certain sets of inputs (e.g. the *less–than* function). There is no way to vary the probability of success of a calculation predicate since they always succeed. A test predicate's probability of success, is sensitive to the probability distribution of its argument quantities. In the diagram below, the



less–than test on the right has a higher probability of succeeding given the illustrated probability distributions for its arguments. While probability distributions are difficult to define and work with, there is a much simpler qualitative view of the probability distribution: probability density decreases as one



moves either higher or lower away from the central value. The general definition for a data approximation embodies this principle. The measured quantity is taken to be the central value. Some distribution is present because of the uncertainty involved. Without knowing any details about the distribution, the definition for a data approximation states that the probability of encountering the actual value for the measured quantity decreases as we get farther from the measured approximate value. One of the rules regarding data approximate quantities is shown below.

```
(rule  :form
        (PQ– (< ?test ?loc) ?test)
        :ants
        (data–approx–quantity ?loc2)
        (IQ+ ?loc ?loc2))
```

This translates to:

> *if a less–than is being performed between ?test and a quantity ?loc which is indirectly or directly qualitatively proportional to a data approximate quantity, the probability of the less–than succeeding is inversely proportional to the magnitude of the ?test quantity*

Rules like this effectively translate goals to increase the probability of success of a predicate into goals to increase or decrease quantities.

In general, an explanation for positively influencing the probability of a predicate proceeds so as to:

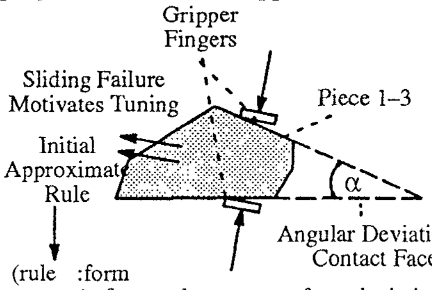
1 ) relate the probability of the failing predicate to that of a test predicate involving approximate quantities

2 ) use the definition of a data approximation to relate the probability of success of a test predicate with the magnitude of a tunable quantity

To guarantee that the probability of the failing predicate will increase, all the test predicates in the rule antecedents must be examined. At least one must show an increasing probability of success and the others must be non–decreasing.

| Step 4 . Use the quantity tunings required by the tuning explanation of step 3 to modify the necessary approximate rules |
|---|

The tuning explanation serves to identify the quantities to be tuned and suggests the direction. It does not, however, specify the new value that the tunable quantity should take on. The rule–approximation gives calculable upper and lower bounds on the values of the tunable quantity. The approach currently taken is to tune to the extreme in the absence of other information about tuning that quantity in that rule approximation. If the tuning explanation suggests that a value be increased, it is increased to its maximum value. Once a quantity in a rule approximation has been tuned once, constraints associated with the rule approximation begin to be posted. These constraints serve to identify tuning tradeoffs. A tuning constraint tells the direction in which the tunable quantity should be tuned, gives the specific numeric value from which the tuning was to take place, and provides a set of predicates which calculate that value in general. Consider the rule approximation illustrated in Figure 1 which seeks to determine the maximum angle by which two contact surfaces of a piece may deviate before slipping occurs. The initial approximate rule led to the object slip-


Gripper Fingers
Sliding Failure Motivates Tuning
Piece 1–3
Initial Approximate Rule
α
Angular Deviation of Contact Faces

```
(rule  :form
        (safe–angular–contact–face–deviation ?object ?a)
        :ants
        (max–angular–face–deviation ?object ?a))
```

Figure 1. A Rule Approximation for Maximum Contact Face Deviation

ping out of grasp as force was applied by the gripper. The first time the approximate quantity, ?a in this case, is tuned a decrease would be suggested by the tuning explanation. The decrease would be from the numeric value returned by the predicate *max–angular–face–deviation.* The direction, numeric value, and predicate are saved as constraints on future tuning of the approximation. In this case, the initially approximate rule states that any two faces are acceptable for the grasp by allowing the two faces chosen to deviate by as much as the two faces which deviate most. On the first failure diagnosed as a slipping failure, the suggest tuning would be to the other extreme: choosing the two faces which deviate the least. This fixes the immediate problem but will likely require further tuning as it over–constrains other important properties of the grasp such as distance to center of gravity. That is, a possible tuning of another approximation may require the re–tuning of this approximation. This is triggered by the presence of inter–approximation constraints. Distance to center of gravity and face angle are so constrained through choice of contact faces.

This phase of approximation tuning continually constrains the tunable values. Tradeoffs are currently settled using equal weights for the various constraints. A set of successive constraints for an approximation appear graphically as shown in Figure 2 for a rule approximation whose tunable quantity is the proper opening width of a gripper for grasping.

The constraining phase of approximation tuning may eventually arrive at a set of constraints which it is not possible to resolve (e.g. an increasing constraint at a higher numeric value than a decreasing constraint) or it may be determined that the quality of plan is being undermined by repeated constraints (e.g. objects tend to be very close together so that continually tuning the opening width won't pay off). The former can easi-

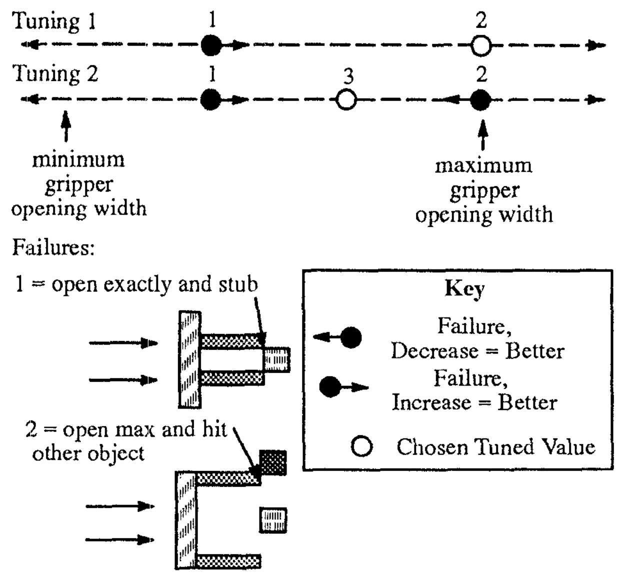

Tuning 1
1
2
Tuning 2
1
3
2
minimum gripper opening width
maximum gripper opening width
Failures:
1 = open exactly and stub
2 = open max and hit other object
Key
Failure, Decrease = Better
Failure, Increase = Better
Chosen Tuned Value

Figure 2. Managing Tradeoffs During the Constraining Phase

ly be detected. The latter can be handled by choosing a limit for how constrained a tunable quantity is allowed to get and/or how fragmented the range of possible values is becoming.

Once the limit for constraining a tunable quantity, within a single approximate rule, has been reached, multiple approximate rules are formed. The original rule will be split into two rules distinguished on the context of the failure. For this purpose, the failing explanation is converted into a series of preconditions for a new approximate rule. The set of preconditions for the new approximate rule capture the same reasoning which inferred which specific failure had occurred. Therefore, having formed the new approximate rule, the tuning for the failure at hand is the initial tuning for this new rule and is subject to no past intra-approximation tuning constraints.

## 4. An Example Illustrating Use of the Algorithm

Figure 3 shows the raw contour information acquired by the

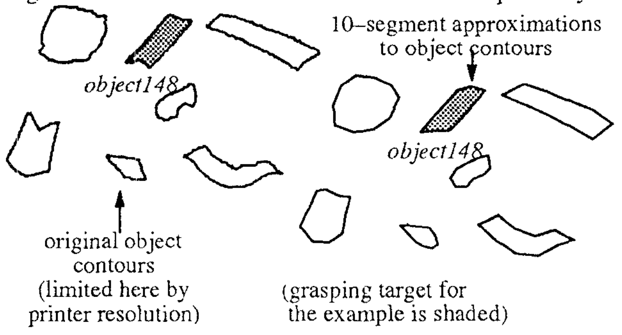

10-segment approximations to object contours
object148
object148
original object contours (limited here by printer resolution)
(grasping target for the example is shaded)

Figure 3. Initial View of Work Space and Initial Internal Data Approximations

system on the left and the initial internal data approximations for the contours on the right. The shaded object has been selected as the target of the grasp for this example. The system then constructs an explanation for how to achieve the goal *(grasp gripper1 object148)*. The explanation is then generalized (using the EGGS algorithm) and packaged into a rule for accomplishing the goal. The rule includes an operator sequence which is a declaratively specified set of monitored actions. A monitored action monitors various expected sensor values and also includes an explanation justifying those expected sensor values. To illustrate the rule approximation tuning algorithm we will be analyzing a common initial failure of the first approximate plan. The initial approximate rule below involves how widely to open the gripper. This rule indicates

```
(rule    :form
         (determine-opening-amount ?gripper ?object
         ?angle ?chosen-width)
         :ants
         (max-radius-at-orientation ?object ?angle ?radius)
         (prod ?radius 2 ?chosen-width))
```

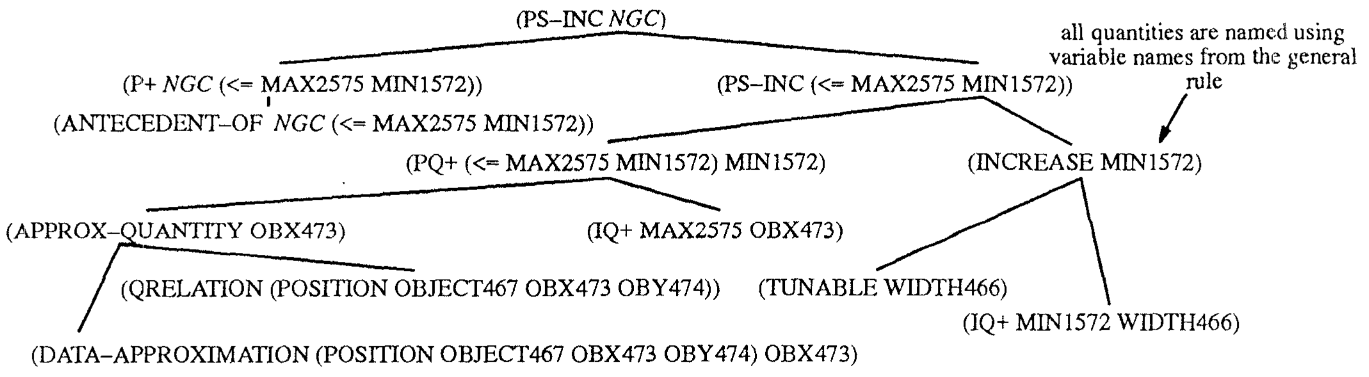
the gripper should be opened enough to surround the perceived width of the object. After the gripper has moved above the object's geometric center, the fingers are opened to the designated width. The gripper is then moved down to surround the object. The monitored move of the gripper expects to feel no forces until the table is reached because in the approximate model of the world there would be no collision between the gripper and any other objects. However, a common initial failure is stubbing one or both of the fingers on the edge of the object. This shows up as an expectation violation and triggers approximation tuning. The expectations were that there would be no gripper collisions with other objects during the down move. In step 1 of the algorithm, the explanation for this expectation is reduced to the one shown in Figure 4 which in-

```
(NO-GRIPPER-COLLISION-OBJECT GRIPPER1 FINGER1
  263 180 0 40 OBJECT148)
   (LEFT-FINGER-OF GRIPPER1 FINGER1)
   (NON-INTERSECTING-GRIPPER-FINGER-OBJECT
    GRIPPER1 FINGER1 263 180 0 40 OBJECT148)
      Subproof for translating finger to appropriate opening
       width (6 facts, 8 built-ins)
      Subproof for counter-rotating object center for clipping
       against finger (8 built-ins)
      Subproof for calculating extents and checking for
       overlap (7 built-ins)
   (RIGHT-FINGER-OF GRIPPER1 FINGER2)
   (NON-INTERSECTING-GRIPPER-FINGER-OBJECT
    GRIPPER1 FINGER2 263 180 0 40 OBJECT148)
      Subproof for translating finger to appropriate opening
       width (6 facts, 8 built-ins)
      SHARED Subproof for counter-rotating object center
       for clipping against finger (8 built-ins)
      Subproof for calculating extents and checking for
       overlap (7 built-ins)
```

Figure 4. Explanation Specific to Failure

dicates that no collision should have occurred between the gripper and object148. This is rated as the most plausible failure among the possible collision failures. This is because the test predicates employed by this explanation, inequality tests which perform the final geometric collision checks between the data approximate gripper and object, are most likely to have failed. This is because object148 was the closest one to the gripper fingers causing the two quantities in the inequality to be very close in value.

Next, the failure explanation is generalized into an EBL rule and qualitative relations are asserted for the antecedents. Tunable quantities from rule approximations as well as data-approximate quantities are also identified. Then, an explanation


(PS–INC *NGC*)
all quantities are named using variable names from the general rule
(P+ *NGC* (<= MAX2575 MIN1572))
(ANTECEDENT–OF *NGC* (<= MAX2575 MIN1572))
(PS–INC (<= MAX2575 MIN1572))
(PQ+ (<= MAX2575 MIN1572) MIN1572)
(INCREASE MIN1572)
(APPROX–QUANTITY OBX473)
(IQ+ MAX2575 OBX473)
(QRELATION (POSITION OBJECT467 OBX473 OBY474))
(TUNABLE WIDTH466)
(IQ+ MIN1572 WIDTH466)
(DATA–APPROXIMATION (POSITION OBJECT467 OBX473 OBY474) OBX473)

Where *NGC* represents the failing predicate:

(NO–GRIPPER–COLLISION–OBJECT GRIPPER461 X463 Y464 ANGLE465 WIDTH466 OBJECT467)

Figure 5. A Qualitative Tuning Explanation

is produced for how to increase the probability of success that no gripper collision will occur with the target object. Figure 5 shows the qualitative explanation for how opening the gripper (increasing the opening–width tunable quantity) positively influences the probability that there will be no collision between the first gripper finger and the object. Table 1 gives the

| | |
|---|---|
| (PS–INC ?pred) | *the probability of success of ?pred is influenced positively* |
| (P+ ?pred1 ?pred2) | *the probability of success of ?pred2 influences the probability of success of ?pred1 positively* |
| (ANTECEDENT–OF ?pred1 ?pred2) | *?pred2 is an antecedent of ?pred1 in the rule being analyzed* |
| (PQ+ ?pred ?quant) | *the magnitude of the quantity ?quant influences the probability of success of ?pred positively* |
| (INCREASE ?quant) | *the magnitude of the quantity ?quant is influenced positively* |
| (APPROX–QUANTITY ?quant) | *?qant is an approximate quantity from a data approximation (not controllable by the system)* |
| (IQ+ ?q1 ?q2) | *the magnitude of quantity ?q2 indirectly influences the magnitude of quantity ?q1 positively* |
| (TUNABLE ?quant) | *the magnitude of quantity ?quant is a tunable quantity from a rule approximation (controllable by the system)* |

Table 1. Predicates Employed in the Tuning Explanation of Figure 5

semantics for the predicates employed in the explanation. The topmost left–hand subtree establishes that the probability of the <= test predicate can influence the probability of the *no–gripper–collision* goal because it is an antecedent of the rule. The right–hand subtree establishes that the probability of the <= can be positively influenced through an increase in the opening width. The IQ+ predicate is a built–in predicate for establishing transitive relations between quantities. It consults the body of qualitative proportionalities which hold in the current situation. There is a similar supporting explanation for the other finger, which moves oppositely while opening. Together, the two subproofs cover all the test predicates employed in the original explanation and thus guarantee that opening the gripper wider decreases the chance of this failure (with the target object) happening in the future.

## 5. Conclusion

Any system which hopes to manage a model of the world in conjunction with actions and values sensed in the real world has to have some approximation mechanism. The characterization of data and rule approximations provides a good framework from which to explore how to manage approximations. Tunable approximate quantities are sufficiently powerful to introduce uncertainty tolerance into plans in an on–demand manner. The approximation tuning method presented offers a general way of discovering the relationships between the tunable approximate quantities, data approximate quantities measured from the world, and ultimately the probability of success of a goal.

## 6. Acknowledgements

I would like to thank my advisor, Gerald DeJong, for his many helpful discussions and insightful comments. This research was supported by the Office of Naval Research under grant ONR N00014–86–K–0309.

## 7. References

[Bennett89a] S. W. Bennett, "Learning Approximate Plans for Use in the Real World," *Proceedings of the Sixth International Conference on Machine Learning*, Ithaca, NY, June 1989, pp. 224–228.

[Bennett89b] S. W. Bennett, "Learning Uncertainty Tolerant Plans Through Approximation in Complex Domains," M.S. Thesis, ECE, University of Illinois, Urbana, Il., January 1989.

[DeJong86] G. F. DeJong and R. J. Mooney, "Explanation–Based Learning: An Alternative View," *Machine Learning 1*, 2 (April 1986), pp. 145–176.

[Keller87] R. M. Keller, "The Role of Explicit Contextual Knowledge in Learning Concepts to Improve Performance," Ph.D. Thesis, Department of Computer Science, Rutgers University, New Brunswick, NJ, January 1987.

[Mitchell86] T. M. Mitchell, R. Keller and S. Kedar–Cabelli, "Explanation–Based Generalization: A Unifying View," *Machine Learning 1*, 1 (January 1986), pp. 47–80.

[Mooney86] R. J. Mooney and S. W. Bennett, "A Domain Independent Explanation–Based Generalizer," *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp. 551–555.

[Segre87] A. M. Segre, "Explanation–Based Learning of Generalized Robot Assembly Tasks," Ph.D. Thesis, Department of Electrical and Computer Engineering, University of Illinois, Urbana, IL, January 1987.

# Controlling Decision-Theoretic Inference

Mark Boddy and Keiji Kanazawa*

Department of Computer Science
Brown University
Box 1910,
Providence, RI 02912

## 1 Introduction

We are interested in the development of frameworks for decision-making under uncertainty in time-critical situations. In the past, we have separately investigated (i) *deliberation scheduling* [Boddy and Dean, 1989], in which decision-making computation is scheduled explicitly to make use of the time available, and (ii) *probabilistic planning* [Kanazawa and Dean, 1989, Dean and Kanazawa, 1987]: the application of normative probabilistic and decision-theoretic reasoning to planning. In this paper, we explore the applicability of deliberation-scheduling methods to probabilistic and decision-theoretic reasoning. In time-critical situations, it may not be feasible to perform probabilistic inference using conventional techniques or algorithms. In order to address such problems, we analyze the tradeoff between the time cost of constructing and solving problem models and the expected utility of the decisions made using those models. In order to rate the success of this approach, we determine the expected gain in utility over the use of conventional methods.

This work addresses the problem of planning under uncertainty in two distinct ways. First, object-level uncertainty in the world is directly addressed by the model of decision making that we adopt: *influence diagrams* [Howard and Matheson, 1984]. Second, using deliberation-scheduling methods allows us to reason explicitly about uncertainty in the performance of the decision procedures we employ, and in the range of problems that our system must contend with.

*This work was supported in part by an IBM Graduate Fellowship, and by the National Science Foundation under grant IRI-8612644 and by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Air Force Office of Scientific Research under Contract No. F49620-88-C-0132.

## 2 Specifics

Suppose that we have a system faced with the problem of making a decision, where that decision will be of no utility at all if it is not available in the next $t$ time units. If the system uses a fixed-time decision procedure taking, say, $s$ time units to run, then for $t < s$ the answer will not be available in time, while for $t >> s$ the system will have used much less time that was available, and may not make a very good decision. We use a decision procedure that can be tailored to use the time available. This is not an *anytime algorithm* as defined in [Dean and Boddy, 1988]. Once the customized decision procedure is constructed, we must still wait a fixed period of time for an answer. In Section 3, we discuss the construction of an anytime decision procedure for cases where the time available is known only approximately.

The algorithm we have developed involves iteratively constructing a partial influence diagram by adding arcs (and nodes if needed), such that the resulting influence diagrams are all subsets of the full decision model for the class of problems that the system can address.[1] By compiling expectations on the length of time needed to solve these partial decision models (influence diagrams), we can make use of the time available by continuing to construct larger influence diagrams as long as the probability of their evaluation completing in the remaining time remains sufficiently high (see below).

Partial influence diagrams are built according to the procedure in Figure 1. The parameter $n$ is chosen to maximize the expected utility of the answer returned. Given $t_1$ = time to evaluate a given di-

[1]This is distinct from Breese's approach [Breese, 1987] in that we do not start with a large database and construct the appropriate decision model. The problem we handle is smaller: we already have the decision model, and the question is how best to employ it in the time available.

```
Procedure Build_diagram($n$)
    for $i$ = 1 to $n$
        begin
            Choose an arc
            Add it to the current diagram.
            Add the node at the tail of the
             arc, if needed.
        end
```

Figure 1: Building partial influence diagrams

agram, $tc$ = the time required to add one arc, and $\mu_1$ = the utility of the answer returned, the expected utility $E(\mu)$ is calculated as follows:

$$E(\mu) = p(t_1 < t - n * tc) * E(\mu_1)$$

We assume that $t_1$ and $\mu_1$ are conditionally independent, given $n$. Constructing the required distributions for $t_1$ and $\mu_1$ is a straightforward exercise in statistical inference.

The procedure **Build_diagram** may not be a simple one; "choosing an arc" could be a very complicated procedure. The order in which arcs should be added so as to maximize the quality of the decisions returned by the resulting diagrams is dependent on the evidence available. Unfortunately, constructing an *optimal* partial diagram of a given size will require either compiling a table of arcs to add given a particular configuration of evidence, or doing the sensitivity analysis as the arcs are constructed. The former requires an exponential amount of memory, while the latter requires repeatedly solving partial influence diagrams on-line. For now, we add arcs in a fixed order. This order can be set to maximize the expected utility of the decisions made, given expectations on the evidence and amount of time available over the situations encountered by the agent.

We evaluate the influence diagrams constructed by **Build_diagram** using the *clique junction tree algorithm* [Jensen *et al.*, 1989]. The expected time required to evaluate influence diagrams is small, and is polynomial in the size of the diagram and exponential in the size of the largest clique.

So the overall algorithm we use is

1. Run **Build_diagram**, choosing $n$ as described above.
2. Evaluate the resulting influence diagram.

We can compare this algorithm and a fixed-time algorithm by calculating the expected utility of each over the distribution of problem instances. We assume that $n$ and the type of evidence available are independent.
For the fixed-time algorithm:

$$E(\mu_{fixed}) = p(m < n) * E(\mu_1)$$

where $\mu_1$ is the expected utility of the answer exclusive of the cost of being late.
For our algorithm:

$$E(\mu) = p(t_1 < t - n * tc) * E(\mu_1)$$

as defined above. So the expected benefit to being able to tailor decision-theoretic inference in this way is $E(E(\mu) - E(\mu_{fixed}))$, calculated over the distribution of values for $n$.

## 3 Discussion

The algorithm developed in Section 2 is not an anytime algorithm. If $n$ is known and the variance on how long it takes to evaluate a partial diagram is small (or the time itself is small), it is possible to tailor an algorithm in this way, and not expect to lose much utility to either being late or wasting large amounts of time. If $n$ is not known, or the time needed to evaluate a diagram cannot be closely predicted, than an anytime algorithm would be more useful. Such an algorithm might consist of using Cooper's methods to construct a partial belief net corresponding to a partial influence diagram [Cooper, 1988], and then using any of several approximation algorthms on the belief net, e.g., [Horvitz *et al.*, 1989, Cooper, 1984, Henrion, 1986].

The algorithms and calculations in Section 2 provide a basis for addressing a wide range of other problems. The expectations we compile regarding the quality of the decisions made by a partial model can be used to schedule computation on several competing problems, much in the style of the *time-dependent planning* discussed in [Dean and Boddy, 1988]. In this case, "observations" corresponding to the availability of evidence mark the beginning of the time over which computation on a particular problem can proceed.

As another example, suppose we have a system, again faced with a decision problem, but in this case required to solve the same problem repeatedly, where some observations may change from instance to instance. Shachter [Shachter, 1986] defines an algorithm for "absorbing" chance and decision nodes in an influence diagram. Preprocessing the influence diagram by absorbing chance nodes for which we do not expect new information may be useful—the cost of this reduction can be amortized over the number of times the reduced diagram is used. The tradeoff we are concerned with in this case is: should we spend time further reducing the current diagram, update the diagram to reflect recent observations (which may require undoing some or all of the reduction we've done), or just keep using the current diagram?

The reduced diagram may still contain chance nodes for which observations can be made—some features of the problem may change too rapidly for it ever to be a good idea to remove the corresponding chance nodes from the diagram. So there is a more complex tradeoff involved: a smaller diagram is easier to solve, but may need to be re-computed more often (keeping the old diagram as new observations arrive may mean making sub-optimal decisions).

To decide how best to spend our time, we need to calculate the expected cost due to using outdated information, and the expected cost of the delay required to solve a given diagram. We also need expectations for the time required to reduce a diagram (per node removed, or per time-unit decrease in expected time to solve it), and for the time required to solve a given diagram. The expected time required to solve a diagram could be conditioned upon arbitrary features of the individual diagram, but an easy starting point is to condition only on the diagram's size.

## 4 Conclusion

This abstract presents our preliminary investigation of a new approach to controlling probabilistic and decision-theoretic inference in time-critical situations. Initial experimentation has been encouraging. Existing techniques for manipulating and evaluating influence diagrams can be adopted or adapted without much effort, allowing us to build on previous work. These initial explorations seem to indicate that this is a promising area for additional work. Planned extensions include adding more complex models for time cost, better methods for constructing partial influence diagrams, and further use of expectations on the system's performance for controlling inference.

## References

[Boddy and Dean, 1989] Mark Boddy and Thomas Dean. Solving time-dependent planning problems. In *IJCAI89*, 1989.

[Breese, 1987] John S. Breese. Knowledge representation and inference in intelligent decision systems. Technical Report 2, Rockwell International Science Center, 1987.

[Cooper, 1984] Gregory F. Cooper. *NESTOR: A computer-based medical diagnostic aid that integrates causal and probabilistic knowledge.* PhD thesis, Stanford University, November 1984.

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

[Dean and Boddy, 1988] Thomas Dean and Mark Boddy. An analysis of time-dependent planning. In *Proceedings AAAI-88*, pages 49–54. AAAI, 1988.

[Dean and Kanazawa, 1987] Thomas Dean and Keiji Kanazawa. Persistence and probabilistic inference. Technical Report CS-87-23, Brown University Department of Computer Science, 1987.

[Henrion, 1986] M. Henrion. Propagating uncertainty by logic sampling in bayes' networks. In *Proceedings of the Second Workshop on Uncertainty in Artificial Intelligence*, 1986.

[Horvitz *et al.*, 1989] Eric J. Horvitz, Gregory F. Cooper, and David E. Heckerman. Reflection and action under scarce resources: Theoretical principles and empirical study. In *Proceedings IJCAI-89*, Detroit, Michigan, 1989.

[Howard and Matheson, 1984] Ron A. Howard and James E. Matheson. Influence diagrams. In Ron A. Howard and James E. Matheson, editors, *The Principles and Applications of Decision Analysis.* Strategic Decisions Group, Menlo Park, CA 94025, 1984.

[Jensen *et al.*, 1989] Finn V. Jensen, Steffen L. Lauritzen, and Kristian G. Olesen. Bayesian updating in recursive graphical models by local computations. R- 89-15, Institute for Electronic Systems, Aalborg University, Aalborg, Denmark, 1989.

[Kanazawa and Dean, 1989] Keiji Kanazawa and Thomas Dean. A model for projection and action. In *Proceedings IJCAI-89*, Detroit, Michigan, 1989.

[Shachter, 1986] Ross D. Shachter. Evaluating influence diagrams. *Operations Research*, 34(6):871–882, November/December 1986.

# Integrating Planning and Reaction
A Preliminary Report

**John Bresina** and **Mark Drummond**[*]
Sterling Federal Systems
NASA Ames Research Center
Mail Stop: 244-17
Moffett Field, CA 94035

## Abstract

This paper is a preliminary report on the Entropy Reduction Engine architecture for integrating planning, scheduling, and control. The architecture is motivated through a NASA mission scenario and a brief list of design goals. The main body of the paper presents an overview of the Entropy Reduction Engine architecture by describing its major components, their interactions, and the way in which these interacting components satisfy the design goals.

## 1 Motivation

NASA has plans to send a rover to Mars sometime this decade. Let's consider two extreme design scenarios for such a mission.

In the first scenario, let's assume that in advance of the rover's deployment, all relevant facts are known by the design team; for example, soil surface characteristics, surface topography, and location of all areas which could be hazardous to the rover. With all this foreknowledge, the designers can specify desired rover behavior for all situations the rover will encounter. The designers can produce a control system which enables the rover to achieve all scientific goals under Martian operating conditions.

Now consider a second scenario in which the design team has limited foreknowledge of the relevant facts needed to produce a rover control system. In this case, the rover must be capable of performing, on Mars, some of the activities that the designers could not complete due to lack of knowledge. For example, since the possible situations and goals will be unknown to the designers, the rover must be capable of determining, at runtime, a response appropriate to a novel situation-goal pair. This determination may involve synthesizing a complex behavior and evaluating it before acting.

[*]This work has been partially supported by the Air Force Office of Scientific Research, Artificial Intelligence Research Program.

These two scenarios vary only in the amount of foreknowledge possed by the rover designers. Most realistic mission scenarios will fall somewhere between these two – some parameters will be known in advance, and it will be necessary to determine some others at runtime. In any scenario there is a role for automated tools that reason about goals, that select actions relevant to those goals, that schedule selected actions, and that do temporal projection to determine possible consequences of behaviors. These tools can be useful as knowledge compilers in advance or as reactive systems at run time, or both to some degree, depending on the designers' foreknowledge and other mission constraints. Our research goal is to analyze, implement, and integrate such tools. The Entropy Reduction Engine (ERE) architecture is our developing body of theory in this endeavor.

## 2 Design Goals

The primary design goal for ERE has been to integrate planning (goal reasoning and action selection), scheduling (action sequencing and resource allocation), and control (monitoring of and adapting to a dynamic environment). This overall goal can be decomposed into the following design subgoals.

**Manage goals with temporal extent.** Standard planning goals of simple conjunctive achievement are not particularly useful in realistic situations. We want to be able to express behavioral constraints of maintenance and prevention over intervals of time.

**Schedule actions in terms of metric time and metric resources.** Most realistic applications for tools which manage time and actions involve a significant scheduling component. Planning and scheduling must be functionally integrated.

**Synthesize plans.** Scheduling a predetermined set of actions is not enough – many applications require that the set of actions be selected automatically.

**Act without plans.** It is not always possible to produce a plan for a problem in the time available. Unplanned action must be possible.

**Manage disjunctive plans.** The system must be able to represent and synthesize disjunctive plans. A disjunctive plan is more robust; that is, it increases the likelihood of successful execution.

**Reason about parallel actions.** Parallelism is rife in realistic applications. Both possible and necessary parallelism must be handled in terms of representation and temporal projection capability.

**Analyze plan execution as a control theory problem.** A reaction plan can be viewed as a specification of how to react to a set of situation–goal pairs. Versions of this idea can be found in modern discrete event control theory (Ramadge and Wonham, 1989). These ideas from AI and control theory must be integrated and extended.

**Encode problem solving strategies when available.** Problem solving strategies for a domain or set of problems are often known by domain experts. We want to capture and exploit such expert knowledge so as to make search more efficient when possible.

**Plan while things are changing.** The world will often change while planning is going on. The plan formation process must be able to deal with changing situations.

**Plan synthesis must have anytime, incremental characteritics.** It should be possible to stop a plan synthesis algorithm at any time during its execution and expect useful results. One should also expect the "quality" of the results to improve continuously as a function of time. (Refer to Dean and Boddy, 1988 for more details.)

## 3 ERE Architecture Overview

This section gives a guided tour of our architecture and explains how it addresses each of the previous section's design goals. The ERE architecture includes the following components.

1. The *reactor* produces reactive behavior in the environment.
2. The *projector* explores possible futures and provides advice about appropriate behaviors to the reactor.
3. The *reductor* reasons about behavioral constraints and provides search control advice to the projector.

This architecture is organized around the *Principle of Independent Ability*, which is as follows: *each component must have the basic ability to perform its assigned task.* In no way does independent ability guarantee good performance; in fact, a component in isolation will typically exhibit poor performance and will improve only through interactions with other components.

For a concrete example consider the reactor and projector components. The reactor is able, in principle, to realize all the behaviors that are possible in a given domain. However, without any advice, the reactor is myopic – it does not know the future consequences of its behavior nor does it know whether its behavior will satisfy the given behavioral constraints. The performance level of the reactor is increased through interactions with the projector. The projector considers consequences of the various possible behaviors and advises the reactor on which particular behavior best satisfies the given behavioral constraints.

The reductor–projector interface is similar. Forward chronological search performed by the projector is inherently myopic; the projector does not have a "global picture" of the search space and as a result does not know which behaviors to project and which others to ignore. Of course projection can be done – it is just not very efficient. The projector aspires to efficiency by accepting search control guidance from the reductor. The reductor uses domain-specific planning expertise to recursively decompose the given problem into a conjunction of simpler (and more localized) subproblems. The conjunction represents a strategy for solving the overall problem and is used to provide global advice to the projector.

In both the projector–reactor and reductor–projector interactions, the input from one component simply serves to control an existing ability and does not serve to define that ability. This approach differs from that taken in classical "plan execution systems". A traditional plan executor (Wilkins, 1984) has nothing to do if it has no plan. In contrast, our reactor can always do something: the existence of a "plan" simply serves to increase the goal-achieving properties of the reactor. Similarly, the projector can consider possible futures without reference to some developing plan – search guidance from the reductor serves to control the projection when such advice is available, but such advice is not strictly necessary.

The principle of independent ability fits cleanly with the idea of an anytime algorithm. By decoupling the system into reduction, projection, and reaction, the ERE architecture can exploit each component's anytime characteristics. For example, the projector can give guidance to the reactor once it has found *a single* behavior satisfying the given constraints, and can incrementally augment this guidance with descriptions of other satisfactory behaviors as these are discovered in the projection.

The reductor has similar anytime characteristics. Initially, all behaviors which do not necessarily violate the overall behavioral constraint are allowed according to the reductor's first-cut problem solving strategy. Successive applications of reduction operators serve to refine the problem solving strategy providing search guidance that grows increasingly detailed and accurate over time, thus restricting the projector to ever fewer of the myriad possible behaviors.

The following three sections explain, in more detail, the functions of the ERE components and the nature of their decoupled anytime interaction.

### 3.1 The Reactor

The reactor accepts a specification of the environment's dynamics represented as a *plan net* (Drummond, 1985, 1986). A plan net defines the events that are possible in the environment in terms of each event's preconditions and situation-dependent effects. Each event is represented by a single operator in the plan net. From the point of view of the reactor, a plan net can be characterized by a set of operators and the two functions given below, where $S$ is the domain's set of possible situations, $O$ is the set of plan net operators, and $\Pi(O)$ denotes the power set of $O$ (note that this is a slight simplification of the full formalism explained in Drummond, 1989).

- $executancy : O \mapsto \{true, false\}$
- $enabled : S \mapsto \Pi(\Pi(O))$

The function `executancy` distinguishes between external events and agent-based actions. That is, `executancy(o)` indicates whether the reactor has control over the execution of the action denoted by operator o or whether o denotes an event whose occurrence is determined by the environment.

The function `enabled(s)` returns a set of operator sets in the plan net, where each of the operator sets returned can be performed in parallel in situation `s`. The reactor is only concerned with those operators that are enabled according to its current "world model". It needs to find a set of operators enabled in its world model for which it has executancy. The reactor interprets the plan net as a nondeterministic program, choosing and executing possible actions in an undefined order.

Control over the execution process is achieved by the use of Situated Control Rules, or SCRs (Drummond, 1989). An SCR is an if-then rule, where the antecedent refers to elements of the reactor's current world model and the current behavioral constraint, and where the consequent contains a set of possible operator sets to execute. Essentially, the consequent of an SCR for a situation $s$ and behavioral constraint $B$ contains those operator sets whose execution defines a prefix to a behavior which satisfies $B$. This means that the SCR's consequent is a subset of `enabled(s)`, since the operators that satisfy posted behavioral constraints will include some (but typically not all) of those operators that are enabled in $s$. The synthesis of these SCRs is discussed in more detail in Drummond (1989), and the next section provides a brief overview of the process.

The reactor always checks to see if any SCRs exist that are appropriate to the current situation and given behavioral constraints. If so, the SCRs' advice about what to do next is heeded. If there are no appropriate SCRs, unplanned execution is still possible. Without reference to the SCR input from the projector, the reactor simply selects and attempts to execute any enabled operator in the plan net. The results of such nondeterministic execution are (of course) unpredictable.

For the fully autonomous extreme of the rover example considered in section 1, the plan net given to the reactor would contain a specification of all actions the rover could perform, as well as all relevant external events which could affect the success of the rover's mission. For instance, an action for the rover could be `aim-laser-range-finder`, and an external event could be `rock-slips-from-gripper`. A background set of SCRs would be provided to give the rover essential reactions to situations demanding immediate response (*e.g.*, those needed for self-preservation). Other SCRs can be synthesized dynamically by the projector.

## 3.2 The Projector

The projection process considers the effects of events under the system's control and external events caused by the environment or other agents (*cf* Dean and McDermott, 1987). Projection is simply a search through the space of possible event sequences. A projection path represents a possible behavior. Considering all possible future behaviors is typically impossible.

The projector needs to view the plan net as a causal theory and so requires the following extra function which describes the effects of a set of operators $o$ in a situation $s$. The function is defined $\forall o \subseteq O, s \in S$.

$$apply(o, s) = \begin{cases} s' \in S & \text{if } o \in enabled(s) \\ undefined & \text{otherwise} \end{cases}$$

Projection associates a duration with each set of operators applied and uses this to calculate a time stamp for each new situation. Currently, operator durations are integers and can be a function of the situation in which the operators are applied; situation time stamps are also integers.

Behavioral constraints are conjunctions and disjunctions of the following two forms.

- $(maintain\, \phi\, t_1\, t_2)$ is true of a projection path iff wff $\phi$ is true from time point $t_1$ through time point $t_2$ in the path.
- $(prevent\ \phi\ t_1\ t_2)$ is true of a projection path iff wff $\phi$ is false from time point $t_1$ through time point $t_2$ in the path.

A wff is a conjunction or disjunction of grounded predicates. Time points refer to situational time stamps and can be integers or variables; the domain of each variable is the integers. Arithmetic constraints on time point variables are allowed in the language. This language might appear quite simple but it allows us to express behavioral constraints that are more complicated than most planning systems can handle.

For example, the language allows the following:

```
(and  (maintain (memory 3 6) 1 5)
      (prevent (battery low) 2 7)
      (maintain (image taken) ?t ?t))
```

where `(memory 3 6)` indicates in our rover domain that the amount of memory available is between three and six megabytes, `(battery low)` indicates the battery's status, and `(image taken)` is true when a picture from the rover's camera has been taken. This constraint requires that the first predicate be true from time 1 through time 5 and that the second predicate be true from time 2 through time 7. The third conjunct in the constraint corresponds to a traditional goal of achievement, where the predicate must be true at an arbitrary but single point in time, here indicated by the variable `?t`.

Our approach calls for two phases of temporal projection. First, we find a single projection path that satisfies all given constraints. The search method used is based on likelihood (how probable is a candidate partial path; *cf* Hanks, 1990) and utility (how well does a candidate partial path satisfy the given constraints). The projection path is compiled into SCRs, giving the reactor a single correct behavior. The result of this first phase is somewhat like a triangle table (Nilsson, 1984) insofar as the reactor has information regarding what to do for any situation in a defined sequence. Our second phase of operation attempts to make this first solution more robust by strengthening probabilistically "weak" sections of the behavior. This two-phase approach gives the SCR synthesis anytime characteristics; details are explained by Drummond and Bresina (1990).

For a projection example let's look to our ongoing Mars rover scenario. There are limited resources on board, and given goals will often compete for these resources (e.g., the goals of obtaining a sample and of ensuring rover safety). Provided that an appropriate plan net and behavioral constraints are given to the on-board executive system, competing possible behaviors can be considered in terms of their likelihood and the degree to which each satisfies the given constraints. Projection will produce appropriate SCRs to be used by the reactor when the relevant situations arise.

The initial behavioral constraints will rarely provide enough control over the temporal projection search due to their scope: behavioral constraints are typically global, and temporal projection, while it eventually constructs a behavior with this global scope, does so incrementally through a series of single operator applications. Our problem of search control in this context is not new. All "goal-oriented" systems require a mechanism that can translate a computationally non-effective goal into a computationally effective means for controlling the search for a solution which satisfies the goal.

We expect the reductor to translate "global non-effective" behavioral constraints into ones that are "local" and "computationally-effective" to control temporal projection. The basic idea behind this translation process is the topic of the next section.

## 3.3 The Reductor

Standard problem reduction operates by applying nonterminal reduction rules to recursively decompose problems (situation–goal pairs) into conjunctions of "simpler" subproblems until "primitive" problems are recognized by terminal reduction rules which return their "obvious" solutions (Nilsson, 1971). A *complete* reduction trace is represented as an And tree whose root node represents the initial problem and whose leaves represent solved subproblems. The trace of a search through the reduction space is represented as an And/Or graph.

The ERE reductor is based on the REAPPR system (Bresina, 1988; Bresina, *et al.*, 1987) which extends this standard approach in a number of ways. REAPPR enables the encoding and effective utilization of domain specific and problem specific planning expertise. In order to fulfill its role in the ERE architecture, REAPPR is undergoing customizations and extensions.

In the ERE context, a problem is a pair consisting of a situation and a behavioral constraint. Nonterminal reductions can decompose a behavioral constraint based on its logical structure, its temporal extent, the logical structure of its formulae, or the semantics associated with the formulae's predicates.

For instance, in terms of the fully autonomous rover scenario, if a behavioral constraint requires that the distance to a nearby rock be precisely determined, then there might be two reductions giving more detailed behavioral constraints regarding how exactly this might be achieved. One reduction might specify that two visible light cameras should be used in conjunction with a calculation of binocular disparity; the other reduction might specify that the laser range finder should be used. The two alternative strategies have different costs and the reductions will indicate the situations under which each is appropriate.

The semantics of a nonterminal reduction is that satisfying the conjunction of behavioral constraints specified in the decomposition *implies* satisfaction of the original behavioral constraint. Furthermore, a nonterminal reduction represents the heuristic advice that satisfying the conjunctive subproblems is a *good strategy* for satisfying the original problem. By induction, given a partial reduction And tree, the set of leaf nodes represents a conjunction of subproblems whose satisfaction implies the satisfaction of the root node problem.

In accord with the standard approach, a terminal reduction applicable to a subproblem would return an action which is enabled in the subproblem's situation and satisfies the subproblem's behavioral constraints. Another use of terminal reductions is suggested by the following observation. Once a robust solution for a subproblem has been found by the projector and compiled into a set of SCRs, the projector no longer needs guidance from the reductor on solving subsequent occurrences of that particular subproblem. Hence, terminal reductions can be formed to recognize subproblems covered by existing SCRs, so the reductor will not waste time reasoning about them.

As the tree grows, the leaf subproblems become simpler and more localized; furthermore, they represent an increasingly accurate strategy for satisfying the initial problem. Hence, over time, the conjunctive set of leaf subproblems makes it increasingly easy to estimate the quality of a partial behavior in the projection and to estimate the likelihood that it can be extended to satisfy the overall constraints. The limit of this advice is a *complete* specification of all behaviors which satisfy the overall constraints. This limit is approached as more terminal reductions are applied.

## 4 Conclusion

We have implemented a temporal projection system based on the ideas outlined in this paper and have begun experiments in a domain loosely based around our autonomous Mars Rover scenario. This domain, *The Reactive TileWorld*, involves uncontrollable external events and the need to act before planning is complete. Behavioral constraints in the Reactive TileWorld are complex, typically involving the maintenance of conjunctions of predicates over intervals of time. We have implemented a subset of the goal language defined in this paper; in our language subset, if a variable is used to refer to the time points in a `maintain` or `prevent` statement, the same variable must be used for both the start point and end point. We have implemented the SCR compilation code defined in a previous paper (Drummond, 1989) and are currently developing a set of Reactive TileWorld benchmark experiments. The REAPPR system is being integrated with our temporal projection code.

How does our evolving architecture measure up in terms of our declared design goals? The architecture allows us to schedule actions in terms of metric time and metric resources by considering the situation-dependent effects of actions during projection. It also allows for synthesizing plans by selecting actions, for acting without plans, and for the management of disjunctive plans. The ERE architecture also supports reasoning about parallel actions in temporal projection. The reductor makes it possible to encode domain- and problem-specific strategies when such knowledge is available. All the components of our architecture have incremental, anytime properties. And what of our goal to plan while things are changing? We're working towards that by developing notions of situational *coverage* and overall system *robustness* in an effort to connect our work with modern discrete event control theory. Results will be reported in a forthcoming paper (Drummond and Bresina, 1990).

### Acknowledgements

Thanks to other members of the ERE group for technical inspiration and for help with the implementation – Nancy Sliwa, Rich Levinson, and Andy Philips. Thanks also to Amy Lansky, Smadar Kedar, John Allen, and Ann Reid for helping with the paper and its production.

## References

[1] Bresina, J. 1988. REAPPR – An Expert System Shell for Planning. Technical report LCSR-TR-119, LCSR, Rutgers University, February.

[2] Bresina, J., Marsella, S., and Schmidt, C. 1987 *Predicting Subproblem Interactions.* Rept. LCSR-TR-92, LCSR, Rutgers University, February.

[3] Bresina, J., Marsella, S., and Schmidt, C. 1986. REAPPR – Improving Planning Efficiency via Expertise and Reformulation. Rept. LCSR-TR-82, LCSR, Rutgers University, June.

[4] Dean, T., and Boddy, M. 1988. An Analysis of Time-Dependent Planning. In proc. of *AAAI-88*. pp. 49–54.

[5] Dean, T., and McDermott, D. 1987. Temporal Database Management. AI Journal, Vol. 32(1). pp. 1–55.

[6] Drummond, M. 1989. Situated Control Rules. *Proceedings of Conference on Principles of Knowledge Representation and Reasoning*, Toronto, Canada.

[7] Drummond, M. 1986. A Representation of Action and Belief for Automatic Planning Systems. *Reasoning About Actions and Plans*, M. Georgeff and A. Lansky, Eds., Morgan Kauffman. pp. 189–211.

[8] Drummond, M. 1985. Refining and Extending the Procedural Net. *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, Los Angeles, CA.

[9] Drummond, M., and Bresina, J. 1990. An anytime temporal projection algorithm for maximizing expected run-time robustness. (In preparation.)

[10] Hanks, S. 1990. Projecting Plans for Uncertain Worlds. Yale University, CS Department, YALE/CSD/RR#756.

[11] Ramadge, P.J.G., and Wonham, W.M. 1989. The Control of Discrete Event Systems. *Proceedings of the IEEE.* Vol. 77, No. 1 (January). pp. 81–98.

[12] Nilsson, N. 1971. Problem Solving Methods in Artificial Intelligence. McGraw Hill, N.Y.

[13] Nilsson, N. (ed). 1984. Shakey the Robot. Technical Note 323, Stanford Research Institute.

[14] Wilkins, D. 1984. Domain Independent Planning: Representation and Plan Generation. *Artificial Intelligence*, Vol. 22, pages 269–301.

# Towards Intelligent Real-Time Cooperative Systems

**Edmund H. Durfee**
Department of Electrical Engineering and Computer Science
University of Michigan

## Introduction

In dynamically changing worlds, intelligent decisionmaking cannot be divorced from time: The best decision can lead to disaster if the world has changed substantially by the time the decision is enacted. For example, if I see no cars coming, I might decide to cross the street. Although this is a correct decision given the initial situation, if I spend too much time making this decision or putting it into action, I might still get hit.

We are exploring issues in timely decisionmaking, or, more generally, real-time AI. By real-time, we mean that a system must carry out its actions before the environment has a chance to change substantially. Put another way, a system must act on its environment more quickly than its environment can unpredictably act on it. If we can measure the expected (or minimum) amount of time that the environment needs to change substantially, then we can place hard real-time deadlines on a system [Stankovic and Ramamritham, 1987].

Guaranteed real-time performance demands that the behavior of the system, or at least its worst-case behavior, is predictable. Intelligence, on the other hand, demands some degree of unpredictability—of creativity—without which the system's behavior appears mechanical and unintelligent. Real-time AI systems are complicated, in part, because they must combine these diametrically opposed perspectives.

Many problems, including real-time perception, temporal and commonsense reasoning, planning, execution, recovery, communication, and synchronization, must be solved before we know how to build real-time intelligent systems. Our current emphasis is on studying issues in real-time planning, execution, and communication in the context of simulated and actual robots in a shared workspace. Specifically, we are concerned with developing interacting real-time and AI subsystems, and with investigating the practicality of using communication to synchronize robot actions.

## Motivation

### Real-Time AI

Research in real-time AI has taken several directions. One direction has been to engineer AI systems to meet real-time needs [Laffey *et al.*, 1988]. Typically, this means simplifying a system's knowledge-base and inference mechanism so that the system will respond to all expected inputs within some maximum time. Unfortunately, while these systems might retain some of the languages and algorithms of AI, whatever intelligence they began with has been engineered out in order to guarantee predictable real-time responses.

Another direction has been to develop AI systems that use iterative improvement algorithms, so that at any given time the system can return some approximation of the desired response [Dean and Boddy, 1988, Horvitz, 1987]. Systems that use this approach attain goals within real-time, but this approach is limited to applications that admit to successive-refinement algorithms. In many applications, successfully meeting time constraints might mean that the system generates a useful but unexpected result, rather than an approximation of the expected result. For example, when navigating a vehicle through a congested area, an approximation such as "turn 90 degrees, plus or minus 45 degrees" might lead to disaster, while a completely different response such as "honk your horn and slam on the brakes" might be better.

Both of these directions have viewed real-time AI as *embedding* an AI system within a real-time system. As part of the real-time processing, any AI reasoning must also return a response within a deadline. This view can be contrasted with the view in which real-time (reactive) capabilities are *embedded* within an AI system. For example, Cohen [Cohen *et al.*, 1989] describes an AI architecture which includes a real-time component to detect and respond to time-critical situations. A more unified approach, such as Soar [Laird *et al.*, 1987], encodes reactive knowledge just like any other knowledge, with the stipulation that, when it is applicable, the reactive knowledge should take priority.

A third view is possible, in which neither the real-time component nor the AI component is embedded in the other. Running concurrently, the real-time and AI subsystems exchange appropriate information so that each can acceptably perform its own functions. In a sense, each subsystem is an individual with its own goals and restrictions on behavior; the challenge is to get the individuals to cooperate so that real-time intelligent behavior emerges. Building systems from "cooperating experts" has been an active area of research [Durfee *et al.*, 1989, Smith and Broadwell, 1987], but cooperation between such diverse systems as a "planner" and "real-time executor" will require substantial improvements in technology.

## Planning

An approach to planning can be purely strategic (planning an entire sequence of actions to achieve a goal by assuming that the world is completely certain and closed), purely reactive (planning only the next action because the world is assumed to be extremely uncertain and unpredictably changing), or somewhere in between. Toward the strategic end of this continuum are approaches such as contingency planning [Drummond *et al.*, 1987] and constraint-based planning [Stefik, 1981], while toward the reactive end are systems that invoke short partial plans in response to current circumstances and goals. We have studied a type of planning that falls somewhere in the middle of the continuum, called incremental planning, which sketches out an entire strategic plan at an abstract level, and reactively details specific actions for the plan as it pursues the plan [Durfee and Lesser, 1986].

Incremental planning has been used to control the problem-solving behavior of a blackboard system, and was extended to permit time-constrained problem solving [Durfee and Lesser, 1988]. It does this by providing the blackboard architecture with the ability to make rough predictions about the time needed to generate a solution, and then to replace planned applications of time-consuming knowledge sources (or sequences of knowledge sources) with less costly knowledge sources or none at all. A solution could thus be found more quickly to meet time constraints, but it would be less complete, less precise, less certain, or some combination of these, depending on how the planned sequence was modified. In the original work, the system was given fixed preferences for how to modify sequences, although in subsequent research [Lesser *et al.*, 1988] we proposed criteria for evaluating alternative preferences so that the system could dynamically decide on effective modifications of its plans.

The mechanisms for planning and prediction we developed were very effective in a blackboard system for solving interpretation problems, because interpretation problems often call for similar sequences of actions to be applied to numerous pieces of data. The system could store information about past problem-solving experience, and exploit the repetitious nature of the task by using these experiences to predict the time needs for processing similar data in the future. Our technique therefore depends on predictions; these predictions can be inaccurate, but lower accuracy leads to poorer real-time performance, either in the form of missed deadlines or failure to use all of the actual available time. Therefore, although our technique can be generalized to time-constrained problem solving in systems other than blackboard-based systems, it is most effective only in reasonably predictable problem-solving domains.

## Communication

Most multi-agent planning research has concentrated on conflict avoidance [Cammarata *et al.*, 1983, Corkill, 1979, Georgeff, 1983]. Conflicts arise when different agents need access to a non-sharable resource, such as agents that use the same doorway to move between two rooms. As in single-agent planning [Sacerdoti, 1975, Waldinger, 1977], the approach is to identify conflicts between subgoals, and enforce an ordering on interacting subgoals to avoid the conflicts. In multi-agent planning, this means that before an agent can begin pursuing one subgoal, it might have to wait until the agents pursuing potentially interacting subgoals have finished. While non-interacting subgoals can be carried out in parallel without synchronization between the agents, synchronization is needed during *critical regions* in plans where lack of synchronization can lead to conflict.

If agents are working in completely predictable environments, the steps in the multi-agent plan could be tied to specific start and end times that would enforce the constraints between subgoals. In practice, however, the time agents need to achieve subgoals is not precisely predictable, so synchronization is achieved through communication. Thus, when an agent has achieved a subgoal, it can alert another agent that is waiting to work on an interacting subgoal that it is now safe to begin.

Delays and errors in communicating messages in this model can lead to cases where the multi-agent performance is degraded, such as when agents sit idle because they are waiting for messages. In the worst case, a lost message could lead to the discontinuation of the multi-agent plan. Although this model guarantees that agents will not engage in conflicting actions during critical regions, it cannot guarantee the timely performance of the entire multi-agent plan, or even that the plan will indeed be completed at all.

Moreover, this simple model of communicated messages as "eventually" arriving cannot be used for synchronizing actions that must occur simultaneously. The different actions of multiple agents might need to all begin within a small window of time, such as when a tight formation of vehicles must change direction or when two robots that are supporting a long object must move to a new location. Synchronization messages between the agents must be exchanged reliably and within some maximum window of time for the agents to succeed. While much research in communication and coordination between intelligent agents has assumed that such communication capabilities exist and can be incorporated into AI systems, the validity of this assumption has not received sufficient attention.

### Negotiation

Deadlines have reasons. We often think of deadlines caused by the physical world. For example, if our robot is to avoid colliding with a wall, or is to recognize and return a ping-pong ball, or is to surround and extinguish a forest fire, then it must react within some time bounds. Physics and acts of nature are uncompromising.

Many of the deadlines that an intelligent system faces, however, are social in nature. For example, deadlines for submitting a paper, for building a house, or for supplying a product are often negotiable. A technique for meeting real-time constraints that we often use is to change the constraints—ask for extensions, renegotiate a contract, add in a little extra time. When planning a meeting with a friend across town, for example, we would negotiate a time to meet that would minimize the likelihood of difficult-to-meet deadlines. If I'm worried about traffic delays, then instead of agreeing to meet in 15 minutes and forcing myself to meet that deadline by driving dangerously, I'd instead try to arrange the meeting an hour from now so I can take my time. Although some deadlines are carved in stone, we should not ignore techniques for reasoning about how to change, as well as meet, deadlines.

## Current Research Directions

Our current research in planning and coordination in dynamic environments combines ideas from real-time computing, intelligent planning, real-time communication, and distributed AI. We are attempting to integrate these ideas in order to build simulated and (eventually) actual intelligent robots that can achieve their goals in the face of time constraints that are brought on both by the physical world and by the need to interact with each other.

### Real-Time Planning and Acting

One direction of our research is to use real-time scheduling techniques in conjunction with AI techniques to develop intelligent real-time robotic systems. In this work, a sophisticated AI planning system uses hierarchical goal decomposition, incremental planning, temporal reasoning, and goal interaction knowledge to plan actions that further all of a robot's goals, including its goals to avoid collisions and avoid running out of power. The planner sketches out an entire temporal plan at an abstract level, and details low-level actions for that plan as they are needed. As it generates low-level actions, it makes worst-case predictions as to their time needs and, if they are periodic, their frequency.

For example, let us say that our robot has picked up an object and now has a goal to carry it across the room. To achieve this goal successfully, it must avoid obstacles in its path and make sure it does not drop the object. Thus, along with executing an action to move in a particular direction, it might also need to execute periodic actions to look ahead for potential obstacles, check its gripper to make sure the object is still held, and look for a landmark indicating that it has arrived at its desired destination. Each of these periodic actions has a worst-case time requirement, as well as the frequency of the action. Can the robot guarantee that it can achieve all of these objectives?

The requested actions are passed to the real-time system's scheduler. The scheduler uses the information to decide whether all of these actions can be done together. If not, the scheduler informs the planner of this fact and the planner needs to scale back its expectations: It might move more slowly (reducing the frequency of some actions) or might entirely forget some actions (it might not recheck the gripper until it arrives at its destination). Thus, the intelligent system (planner) must cooperate with the real-time system's scheduler to guarantee performance of the most important actions (generally, the actions that will maintain the robot in a safe state). As the robot embarks on achieving new goals, the planner interacts with the real-time system to schedule the crucial actions for the new goals. Of course, the planner can also request actions that cannot be guaranteed, such that the real-time system will do them "if there is time." By appropriately ranking goals, deciding on actions, and guaranteeing the most important actions, the combined systems can ensure some minimum performance (collisions will be avoided) and can attempt to achieve other less critical but desirable goals (such as delivering parts to workstations). Note that this approach does not embed either system in the other; it allows them run concurrently and to cooperate.

### Real-Time Communication

A second direction that we are exploring involves real-time communication between interacting robots. When multiple agents need to communicate in order to synchronize their actions, they need knowledge about the underlying communication mechanisms, such as whether messages will ever be lost and how long communication takes in the worst (or average) case. To ensure timely communication, we might need to adopt a port-based communication architecture [Shin and Epstein, 1987], allowing different priority channels. In dynamically changing environments, where agents come and go over time, reasoning about messaging capabilities, needs, and priorities will be a complex problem.

As a simple example of the types of tasks we are concerned with, consider several mobile robots that are following each other in a line. If the robot in the front discovers that it must stop as soon as possible, what should it do? It could stop immediately and, at the same time, send messages to the robot behind it to halt. But if the message takes too long to arrive and process, the robot behind might crash into the leader. The robot behind also must ensure that the robot following it will not crash into it. To avoid a chain reaction of rear-end collisions, therefore, a robot that is being followed must decide how quickly the following robot can stop, and a crucial aspect of this decision is using knowledge about the communication channels. If we are to guarantee real-time responsiveness in dynamic domains, then the capabilities and the use of communication channels must be appropriate.

### Real-Time Coordination

The last direction that we are exploring is the role that reasoning about coordination plays in real-time AI. Although many deadlines an agent faces are based on aspects of the physical world that are beyond the agent's control, other deadlines are based on coordination decisions with other agents. For example, if two robots have arranged to pass a part from one to the other at a specific time and place, they have imposed deadlines on themselves for this rendezvous. If one robot is slowed by some unanticipated obstacles, it could try alternative means of meeting the deadline (such as increasing its speed) but this might have drawbacks (such as increasing the chances that it will be unable to avoid a collision). The robot could instead attempt to modify the deadline; it could ask for an extension.

We believe that reasoning about the timing of interactions between intelligent systems is a key aspect of intelligent behavior in dynamic domains. Our expectation is that real-time AI and distributed AI have many connections between them, and that studying these connections will lead to important insights and progress in both fields.

## Conclusion

In conclusion, we are studying several issues in real-time AI, attempting to combine ideas from both the real-time computing and AI fields. Our strategy is to couple separate real-time and AI components using a shared protocol, rather than embedding one component inside the other. In time, we hope that our increased understanding of the relationships between the components will allow us to more fully integrate them into a unified real-time AI system.

To evaluate this work, we are implementing our ideas in a real, physical system composed of several robots. However, we are also using a flexible testbed for simulating multi-agent domains [Durfee and Montgomery, 1989]. Our testbed allows us to simulate different constraints on the actions of and interactions between agents, so that we can more fully evaluate alternative mechanisms for coordinating the different agents. We would like to explore the possibility of extending this testbed for use in simulating real-time domains.

**Acknowledgements.** Many of the ideas in this paper arose out of discussions with Professor Kang G. Shin, David J. Musliner, and Tom Tsukada. Dave Musliner and Terry Weymouth supplied useful comments on this paper.

## References

[Cammarata *et al.*, 1983] Stephanie Cammarata, David McArthur, and Randall Steeb. Strategies of cooperation in distributed problem solving. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 767–770, Karlsruhe, Federal Republic of Germany, August 1983. (Also published in *Readings in Distributed Artificial Intelligence*, Alan H. Bond and Les Gasser, editors, pages 102–105, Morgan Kaufmann, 1988.).

[Cohen *et al.*, 1989] Paul R. Cohen, Michael L. Greenberg, David M. Hart, and Adele E. Howe. Trial by fire: Understanding the design requirements for agents in complex environments. *AI Magazine*, 10(3):32–48, Fall 1989.

[Corkill, 1979] Daniel D. Corkill. Hierarchical planning in a distributed environment. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, pages 168–175, Cambridge, Massachusetts, August 1979. (An extended version was published as Technical Report 79-13, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, February 1979.).

[Dean and Boddy, 1988] Thomas Dean and Mark Boddy. An analysis of time-dependent planning. In *Proceedings of the National Conference on Artificial Intelligence*, pages 49–54, St. Paul, Minnesota, August 1988.

[Drummond *et al.*, 1987] Mark Drummond, Ken Currie, and Austin Tate. Contingent plan structures for spacecraft. In *Proceedings of the Space Telerobotics Workshop*, January 1987.

[Durfee and Lesser, 1986] Edmund H. Durfee and Victor R. Lesser. Incremental planning to control a blackboard-based problem solver. In *Proceedings of the National Conference on Artificial Intelligence*, pages 58–64, Philadelphia, Pennsylvania, August 1986.

[Durfee and Lesser, 1988] Edmund H. Durfee and Victor R. Lesser. Incremental planning to control a time-constrained, blackboard-based problem solver. *IEEE Transactions on Aerospace and Electronics Systems*, 24(5):647–662, September 1988.

[Durfee and Montgomery, 1989] Edmund H. Durfee and Thomas A. Montgomery. MICE: A flexible testbed for intelligent coordination experiments. In *Proceedings of the 1989 Distributed AI Workshop*, pages 25–40, September 1989.

[Durfee *et al.*, 1989] Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill. Trends in cooperative distributed problem solving. *IEEE Transactions on Knowledge and Data Engineering*, 1(1):63–83, March 1989.

[Georgeff, 1983] Michael Georgeff. Communication and interaction in multi-agent planning. In *Proceedings of the National Conference on Artificial Intelligence*, pages 125–129, Washington, D.C., August 1983. (Also published in *Readings in Distributed Artificial Intelligence*, Alan H. Bond and Les Gasser, editors, pages 200–204, Morgan Kaufmann, 1988.).

[Horvitz, 1987] Eric J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In *Proceedings of the 1987 Workshop on Uncertainty in Artificial Intelligence*, 1987.

[Laffey *et al.*, 1988] Thomas J. Laffey, Preston A. Cox, James L. Schmidt, Simon M. Kao, and Jackson Y. Read. Real-time knowledge-based systems. *AI Magazine*, 9(1):27–45, Spring 1988.

[Laird *et al.*, 1987] John E. Laird, Allen Newell, and Paul S. Rosenbloom. SOAR: An architecture for general intelligence. *Artificial Intelligence*, pages 1–64, 1987.

[Lesser *et al.*, 1988] Victor R. Lesser, Jasmina Pavlin, and Edmund H. Durfee. Approximate processing in real-time problem solving. *AI Magazine*, 9(1):49–61, Spring 1988.

[Sacerdoti, 1975] Earl D. Sacerdoti. The nonlinear nature of plans. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, pages 206–214, Cambridge, Massachusetts, August 1975.

[Shin and Epstein, 1987] Kang G. Shin and Mark E. Epstein. Intertask communications in an integrated multirobot system. *IEEE Journal of Robotics and Automation*, RA3(2):90–100, April 1987.

[Smith and Broadwell, 1987] David Smith and Martin Broadwell. Plan coordination in support of expert systems. In *Proceedings of the DARPA Knowledge-based Planning Workshop*, Austin, Texas, December 1987.

[Stankovic and Ramamritham, 1987] J. Stankovic and K. Ramamritham. *Tutorial on Hard Real-Time Systems.* IEEE Computer Society Press, 1987.

[Stefik, 1981] Mark Stefik. Planning with constraints (MOLGEN: Part 1). *Artificial Intelligence*, 16:111–140, 1981.

[Waldinger, 1977] Richard Waldinger. Achieving several goals simultaneously. In *Machine Intelligence 8*, pages 94–136, 1977.

# *Incremental, Approximate Planning: Abductive Default Reasoning*

Charles Elkan
Department of Computer Science
University of Toronto*

**ABSTRACT**: This paper presents an abductive strategy for discovering and revising plausible plans. Candidate plans are found quickly by allowing them to depend on unproved assumptions. The formalism used for specifying planning problems makes explicit which antecedents of rules have the status of default conditions, and they are the only ones that may be left unproved, so only plausible plans are produced. Candidate plans are refined incrementally by trying to justify the assumptions on which they depend. The new planning strategy has been implemented, and the first experimental results are encouraging.

## 1 Introduction

Because of uncertainty and because of the need to respond rapidly to events, the traditional view of planning (deriving from STRIPS [Fikes *et al.*, 1972] and culminating in TWEAK [Chapman, 1987]) must be revised drastically. That much is conventional wisdom nowadays. One point of view is that planning should be replaced by some form of improvisation [Brooks, 1987]. However improvising agents are doomed to choose actions whose optimality is only local. In many domains, goals can only be achieved by forecasting the consequences of actions, and choosing ones whose role in achieving a goal is indirect. Thus traditional planners must be improved, not discarded.

This paper addresses the issue of how to design a planner that is incremental and approximate. An approximate planner is one that can find a plausible candidate plan quickly. An incremental planner is one that can revise its preliminary plan if necessary, when allowed more time.

*For correspondence: Department of Computer Science, University of Toronto, Toronto M5S 1A4, Canada, (416) 978-7797, cpe@ai.toronto.edu.

It is not clear how existing planning strategies can be made approximate and incremental. We therefore first outline a strategy for finding guaranteed plans using a new formalism for describing planning problems, and then show how to extend this guaranteed strategy to make it approximate and incremental.

Our approach draws inspiration from work on abductive reasoning. A plan is an explanation of how a goal is achievable: a sequence of actions along with a proof that the sequence achieves the goal. An explanation is abductive (as opposed to purely deductive) if it depends on assumptions that are not known to be justified. We find approximate plans by allowing their proofs of correctness to depend on unproved assumptions. Our planner is incremental because, given more time, it refines and if necessary changes a candidate plan by trying to justify the assumptions on which the plan depends.

The critical issue in abductive reasoning is to find plausible explanations. Our planning calculus uses a nonmonotonic logic that makes explicit which antecedents of rules have the epistemological status of default conditions. The distinguishing property of a default condition is that it may plausibly be assumed. These antecedents are those that are allowed to be left unjustified in an approximate plan. Concretely, every default condition in the planning calculus expresses either a claim that an achieved property of the world persists in time, or that an unwanted property is not achieved. Thus the approximate planning strategy only proposes reasonable candidate plans.

Sections 2 and 3 below present the formalism for specifying planning problems and the strategy for finding guaranteed plans. In Section 4 the strategy is extended to become approximate and incremental. Section 5 contains experimental results, and finally Section 6 discusses related and future work.

## 2 The planning formalism

Different formal frameworks for stating planning problems vary widely in the complexity of the problems they can express. Using modal logics or reification, one can reason about multiple agents, about the temporal properties of actions, and about what agents know [Moore, 1985; Konolige, 1986; Cohen and Levesque, to appear in 1990]. On the other hand, the simplest planning problems can be solved by augmented finite state machines [Brooks *et al.*, 1988], whose behaviour can be specified in a propositional logic. The planning problems considered here are intermediate in complexity. They cannot be solved by an agent reacting immediately to its environment, because they require maintaining an internal theory of the world, in order to project the indirect consequences of actions. On the other hand, they involve a single agent, and they do not require reasoning about knowledge or time.

Our nonmonotonic first-order logic for specifying this type of planning problem is called the PERFLOG calculus.[1] The formal aspects of the calculus will be discussed elsewhere; for the purposes of this paper PERFLOG axioms can be understood intuitively as logic program rules, and we shall just use the Yale shooting problem [Hanks and McDermott, 1986] to introduce the calculus by example.

Two "laws of nature" are central. In the following rules, think of $S$ as denoting a state of the world, of $A$ as denoting an action, and of $do(S, A)$ as denoting the state resulting from performing the action $A$ in the initial state $S$. Finally, think of $P$ as denoting a contingent property that holds in certain states of the world: a fluent.

$$causes(A,S,P) \rightarrow holds(P,do(S,A)) \quad (1)$$

$$holds(P,S) \wedge \neg\, cancels(A,S,P) \rightarrow holds(P,do(S,A)). \quad (2)$$

Rule (1) captures the commonsense notion of causation, and rule (2) expresses the commonsense "law of inertia": a fluent $P$ holds after an action $A$ if it holds before the action, and the action does not cancel the fluent. Note that since in addition to $A$, one argument of *causes* and of *cancels* is $S$, the results of an action (that is, the fluents it causes and cancels) may depend on the state in which the action is performed, and not just on which action it is.

[1] PERFLOG is an abbreviation for "performance-oriented perfect model logic": the formal meaning of a set of PERFLOG axioms is its perfect model as defined in [Przymusiński, 1987].

Given rules (1) and (2), a particular planning domain is specified by writing axioms that mention the actions and fluents of the domain, and say which actions cause or cancel which fluents. In the world of the Yale shooting problem, there are three fluents, *loaded*, *alive*, and *dead*, and three actions, *load*, *wait*, and *shoot*. The relationships of these fluents and actions are specified by the following axioms:

$$causes(S, load, loaded) \quad (3)$$

$$holds(loaded, S) \rightarrow causes(shoot, S, dead) \quad (4)$$

$$holds(loaded, S) \rightarrow cancels(shoot, S, alive) \quad (5)$$

$$holds(loaded, S) \rightarrow cancels(shoot, S, loaded). \quad (6)$$

The initial state of the world $s_0$ is specified by saying which fluents are true in it:

$$holds(alive, s_0). \quad (7)$$

According to the nonmonotonic semantics of PERFLOG collections of rules,

$$holds(dead, do(do(do(s_0, load), wait), shoot))$$

is entailed by rules (1)–(7). The Yale shooting problem is thus solved.

## 3 Finding guaranteed plans

The previous section showed how to state the relationships between the actions and fluents of a planning domain as a PERFLOG set of axioms. This section describes a strategy for inventing plans using such a set of axioms; the next section extends the strategy to be approximate and incremental.

A PERFLOG set of axioms is a set of general logic program clauses, and the strategy presented here is in fact a general procedure for answering queries against a logic program.

*Iterative deepening.* The standard PROLOG query-answering strategy is depth-first exploration of the space of potential proofs of the query posed by the user. Depth-first search can be implemented many times more efficiently than other exploration patterns, but it is liable to get lost on infinite paths. Infinite paths can be cut off by imposing a depth bound. The idea of iterative deepening is to repeatedly explore the search space depth-first, each time with an increased depth bound [Stickel and Tyson, 1985].

Iterative deepening algorithms differ in how the depth of a node is defined. One depth measure that performs well, called conspiracy depth, is presented in [Elkan, 1989]. Informally, this measure says that a subgoal is unpromising if its truth is only useful in the event that many other subgoals are also true.

*Negation-as-failure.* Given a negated goal, the negation-as-failure idea is to attempt to prove the un-negated version of the goal. If this attempt succeeds, the negated goal is taken as false; otherwise, the negated goal is taken as true. Negation-as-failure is combined with iterative deepening by limiting the search for a proof of each un-negated notional subgoal. If this search terminates without finding a proof, then the original negated subgoal is taken as true. If a proof of the notional subgoal is found, then the negated subgoal is taken as false. If exploration of the possible proofs of the notional subgoal is cut off by the current depth bound, it remains unknown whether or not the notional subgoal is provable, so for soundness the actual negated subgoal must be taken as false.

Negation-as-failure is only correct on ground negated subgoals, so when a negated subgoal is encountered, it is postponed until finding answers for other subgoals makes it become ground. This process is called freezing [Naish, 1986]. If postponement is not sufficient to ground a negated subgoal, then an auxiliary subgoal is introduced to generate potential answers. This process is called constructive negation [Foo *et al.*, 1988].

The performance of the planning strategy just described could be improved significantly, notably by caching subgoals once they are proved or disproved [Elkan, 1989]. Nevertheless it is already quite usable.

## 4 Finding plausible plans

This section describes modifications to the strategy of the previous section that make it approximate and incremental. In the same way that the guaranteed planning strategy is in fact a general query-answering procedure, the incremental planning strategy is really a general procedure for forming and revising plausible explanations using a default theory.

Any planning strategy that produces plans relying on unproved assumptions is *ipso facto* unsound, but by its incremental nature the strategy below tends to soundness: with more time, candidate plans are either proved to be valid, or changed.

*Approximation.* The idea behind finding approximate plans is simple: an explanation is approximate if it depends on unproved assumptions. Strategies for forming approximate explanations can be distinguished according to the class of approximate explanations that each may generate. One way to define a class of approximate explanations is to fix a certain class of subgoals as the only ones that may be taken as assumptions. Looking at the PERFLOG formalism, there is an obvious choice of what subgoals to allow to be assumptions. Negated subgoals have the epistemological status of default conditions: the nonmonotonic semantics makes them true unless they are forced to be false. It is reasonable to assume that a default condition is true unless it it is provably false.

There is a second, procedural, reason to allow negated subgoals to be assumed, but not positive subgoals. Without constructive negation, negated subgoals can only be answered true or false. Negation-as-failure never provides an answer substitution for a negated subgoal. Therefore unproved negated subgoals in an explanation never leave "holes" in the answer substitution induced by the explanation. Concretely, a plan whose correctness proof depends on unproved default conditions will never change because those defaults are proved to hold.

*Incrementality.* An approximate explanation can be refined by trying to prove the assumptions it depends on. If an assumption is proved, the explanation thereby becomes "less approximate". As just mentioned, proving an assumption never causes a plan to change. On the other hand, if an assumption is disproved, the approximate plan is thereby revealed to be invalid, and it is necessary to search for a different plan.

Here are the details of the modifications made to the planning strategy of the previous section. When a negated subgoal becomes ground, the proof of its notional positive counterpart is attempted. If this attempt succeeds or fails within the current depth bound, the negated subgoal is taken as false or true, respectively, as before. However, if the depth bound is reached during the attempted proof, then the negated subgoal is given the status of an assumption.

Initially any negated subgoal is allowed to be assumed. Each iteration of iterative deepening takes place with an increased depth bound. For each particular (solvable) planning problem, there is a certain minimum depth bound at which one or more approximate plans can first be found. Each of these first approximate plans depends on a certain set of assumptions. In later iterations, only subsets of these sets are allowed to be assumed. This restriction has the effect of concentrating attention on either refining the already discovered approximate plans, or finding new approximate plans that depend on fewer assumptions.

## 5 Experimental results

Implementing the planning strategies described above is straightforward, because the PERFLOG calculus is based on definite clauses. In general, it is insufficiently realized how efficiently logics based on definite clauses, both monotonic and nonmonotonic, can be implemented. The state of the art in PROLOG implementation is about nine RISC cycles per logical inference [Mills, 1989]. Any PERFLOG theory could be compiled into a specialized incremental planner running at a comparable speed.

The experiment reported here uses a classical planning domain: a lion and a Christian in a stadium. The goal is for the lion to eat the Christian. Initially the lion is in its cage with its trainer, and the Christian is in the arena. The lion can jump from the cage into the arena only if it has eaten the trainer. The lion eats a person by pouncing, but it cannot pounce while it is already eating. The following PERFLOG theory describes this domain formally.

```
%
% rules for how the world evolves

holds(P,do(S,A)) :-
        causes(A,S,P).
holds(P,do(S,A)) :-
        holds(P,S), not(cancels(A,S,P)).

%
% the effects of actions

causes(pounce(lion,X),S,eats(lion,X)) :-
        can(S,pounce(lion,X)).
can(pounce(X,Y),S) :-
        holds(in(X,L),S), holds(in(Y,L),S),
        not(call(X = Y)),
        not(Z,holds(eats(X,Z),S)).
causes(jump(X),S,in(X,arena)) :-
        can(jump(X),S), holds(in(X,cage),S).
can(jump(lion),S) :-
        holds(eats(lion,trainer),S).
cancels(drop(X,Y),S,eats(X,Y)) :-
        can(drop(X,Y),S).
can(drop(X,Y),S) :-
        holds(eats(X,Y),S).
holds(in(X,H),S) :-
        holds(eats(lion,X),S), holds(in(lion,H),S).

%
% the initial state of the world

holds(in(christian,arena),s0).
holds(in(lion,cage),s0).
holds(in(trainer,cage),s0).
```

Using the guaranteed planning strategy of Section 3, the query `holds(eats(lion,christian),P)?` is first solved with conspiracy depth bound 19, in 4.75 seconds.[2] The plan found is

```
P = do(do(do(do(s0,pounce(lion,trainer)),
                  jump(lion)),
                  drop(lion,trainer)),
                  pounce(lion,christian)).
```

Using the approximate planning strategy of Section 4, the same query is solvable in 0.17 seconds, with conspiracy depth bound 17. The candidate plan found is

```
P = do(do(do(s0,pounce(lion,trainer)),
              jump(lion)),
              pounce(lion,christian)).
```

This plan depends on the assumption that no `Z` exists such that

```
holds(eats(lion,Z),do(do(s0,pounce(lion,trainer)),
                                  jump(lion))).
```

Although the assumption is false and the plan is not correct, it is plausible. Note also that the first two actions it prescribes are the same as those of the correct plan: the approximate plan is an excellent guide to immediate action.

## 6 Discussion

The work reported here ties together ideas from a number of different research areas.

*Approximate planning.* From a knowledge-level point of view, the strategy for finding plausible plans is searching in an abstraction space where the available actions are the same as in the base space, but they are stripped of their difficult-to-check preconditions. Compared to other abstraction spaces [Knoblock, 1989], this space has the advantage that the execution of a plan invented using it can be initiated without further elaboration, if immediate action is necessary.

*Incremental planning.* An incremental approximate planner is an "anytime algorithm" for planning in the sense of [Dean and Boddy, 1988]. Anytime planning algorithms have been proposed before, but not for problems of the traditional type treated in this paper. For example, the real-time route planner of [Korf, 1987] is a heuristic graph search algorithm, and the route improvement algorithm of [Boddy and Dean, 1989] relies on an initial plan that is guaranteed to be correct.

[2] All times are for an implementation in CProlog, running on a Silicon Graphics machine rated at 20 MIPS.

*Abductive reasoning.* Abduction mechanisms have been investigated a great deal for the task of plan recognition, not so much for the task of inventing plans, and not at all for the task of inventing plausible plans. These three different tasks lead to different choices of what facts may be assumed. In the work of [Shanahan, 1989] for example, properties of the initial state of the world may be assumed. In our work, the facts that may be assumed say either that an established property of the world persists, or that an unestablished property does not hold.

*Directions for future work.* One important problem is to quantify how an approximate plan is improved by allowing more time for its refinement. Another problem is to find a planning strategy that is focused as well as approximate and incremental. A focused strategy would be one that concentrated preferentially on finding the first step in a plan–what to do *next.*

## References

[Boddy and Dean, 1989] Mark Boddy and Thomas Dean. Solving time-dependent planning problems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 979–984, August 1989.

[Brooks *et al.*, 1988] Rodney A. Brooks, Jonathan H. Connell, and Peter Ning. Herbert: A second generation mobile robot. MIT AI Memo 1016, January 1988.

[Brooks, 1987] Rodney A. Brooks. Planning is just a way of avoiding figuring out what to do next. Technical Report 303, Artificial Intelligence Laboratory, MIT, September 1987.

[Chapman, 1987] David Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–377, 1987.

[Cohen and Levesque, to appear in 1990] Philip R. Cohen and Hector J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, to appear in 1990.

[Dean and Boddy, 1988] Thomas Dean and Mark Boddy. An analysis of time-dependent planning. In *Proceedings of the National Conference on Artificial Intelligence*, pages 49–54, August 1988.

[Elkan, 1989] Charles Elkan. Conspiracy numbers and caching for searching and/or trees and theorem-proving. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 341–346, August 1989.

[Fikes *et al.*, 1972] Richard E. Fikes, Peter E. Hart, and Nils J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972.

[Foo *et al.*, 1988] Norman Y. Foo, Anand S. Rao, Andrew Taylor, and Adrian Walker. Deduced relevant types and constructive negation. In Kenneth Bowen and Robert Kowalski, editors, *Fifth International Conference and Symposium on Logic Programming*, volume 1, pages 126–139, Seattle, Washington, August 1988. MIT Press.

[Hanks and McDermott, 1986] Steve Hanks and Drew McDermott. Default reasoning, nonmonotonic logics, and the frame problem. In *Proceedings of the National Conference on Artificial Intelligence*, pages 328–333, August 1986.

[Knoblock, 1989] Craig A. Knoblock. Learning hierarchies of abstraction spaces. In *Proceedings of the Sixth International Workshop on Machine Learning*, pages 241–245. Morgan Kaufmann Publishers, Inc., 1989.

[Konolige, 1986] Kurt Konolige. *A Deduction Model of Belief.* Pitman, 1986.

[Korf, 1987] Richard E. Korf. Real-time path planning. In *Proceedings of the DARPA Knowledge-Based Planning Workshop*, 1987.

[Mills, 1989] Jonathan W. Mills. A pipelined architecture for logic programming with a complex but single-cycle instruction set. In *Proceedings of the IEEE First International Tools for AI Workshop*, September 1989.

[Moore, 1985] Robert C. Moore. *A Formal Theory of Knowledge and Action.* Ablex, 1985.

[Naish, 1986] Lee Naish. *Negation and Control in* PROLOG. Number 238 in Lecture Notes in Computer Science. Springer Verlag, 1986.

[Przymusiński, 1987] Teodor C. Przymusiński. On the declarative semantics of stratified deductive databases and logic programs. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216, Los Altos, California, 1987. Morgan Kaufmann Publishers, Inc.

[Shanahan, 1989] Murray Shanahan. Prediction is deduction but explanation is abduction. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1055–1060, 1989.

[Stickel and Tyson, 1985] Mark E. Stickel and W. M. Tyson. An analysis of consecutively bounded depth-first search with applications in automated deduction. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 1073–1075, August 1985.

# Integrating Planning and Acting in a Case-Based Framework.

**Kristian J. Hammond**
The University of Chicago
Department of Computer Science
Artificial Intelligence Laboratory
1100 East 58th Street
Chicago, IL 60637

**Tim Converse**
The University of Chicago
Department of Computer Science
Artificial Intelligence Laboratory
1100 East 58th Street
Chicago, IL 60637

## Planning as understanding

Research in planning has recently made a dramatic change in course. Planning researchers have begun to acknowledge that the world is too complex and uncertain to allow a planner to exhaustively plan for a set of goals prior to execution (Chapman, 1985). More and more, the study of planning is being cast as the broader study of planning, action and understanding (Agre and Chapman, 1987, and Alterman, 1986). The particular cast of this relationship that we have been studying is a view of planning as embedded within an understanding system connected to the environment. The power of this approach lies in the fact that it allows us to view the planner's environment, plan selections, decisions, conflicts and actions through the single eye of situation assessment and response. Because of our further commitment to the use of episodic memory as the vehicle for understanding, it also provides us with a powerful lever on the problem of learning from both planning and execution. In this paper, we draw an outline of our model of this relationship between planning and action, which we refer to as *agency*.

## Memory and Agency

Our model of planning and understanding rises out of three basic pieces of work: Schank's structural model of memory organization (Schank, 1982), our own work in case-based planning and dependency directed repair (Hammond, 1989), and the work of Martin and Riesbeck in Direct Memory Access Parsing (Martin 1989). Our model has been articulated in two programs, TRUCKER and RUNNER, (Hammond, Marks and Converse, 1988 and Hammond, 1989).

The model was first developed to deal with the problem of recognizing execution-time opportunity in the context of a resource-bound agent that is forced to suspend planning in order to attend to execution (Hammond, 1989). The goal of this model was to capture the ability of an agent to suspend goals, yet still recognize execution-time opportunities to satisfy them.

To accomplish this goal, we use a single set of memory structures both to store suspended goals and to understand the agent's circumstances in the world. In response to a blocked goal, an agent's first step is to do a planning-time analysis of the conditions that would favor the satisfaction of the goal and then *suspend* the goal in memory, indexed by a description of those conditions. For example, a goal to buy eggs that was blocked during planning would be placed in memory associated with the condition of the agent being at a grocery store.

During execution, the agent performs an ongoing "parse" of the world in order to recognize conditions for action execution. Following DMAP (Martin, 1989), this parse takes the form of passing markers through an already existing episodic memory. Because suspended goals are indexed in the same memory as is used for understanding, they are activated when the conditions that favor their execution are recognized. Once active, the goals are then reevaluated in terms of the new conditions. Either they fit into the current flow of execution or they are again suspended.

Because the agent's recognition of opportunities depends on the nature of its episodic memory structures, we called the initial model *opportunistic memory*. As we have turned to the broader issues of integrating planning and action we refer to our work as the study of *agency*.

## Initial Results

Our initial implementation of this model, TRUCKER, exhibited exactly the behavior we wanted. A combined scheduling planner and executive, TRUCKER was able to suspend blocked goals and then recognize later opportunities to satisfy them. It also learned plans for goal combinations that it determined would occur again. The recognition of opportunity and the resulting learning of specific optimizations rose naturally out of the agent's ongoing understanding of its environment.

Our model of planning is case-based and, as such, both TRUCKER and now RUNNER plan by recalling existing plans that are indexed by a set of currently active goals. Our initial model of this indexing (Hammond, 1986) was based on the notion that a planner could amortize its planning efforts by caching plans in memory under descriptions of the goals that they satisfied and the problems (in the sense of interactions between steps of a plan) that they avoided. In TRUCKER, we worked on the idea that this caching and later retrieval could itself be cast as a problem of characterizing the situations in the world under which a plan could be run and then treat retrieval as a process of recognition, similar to that used in the understanding of the world. The result of this was that plans were cached in the same memory organization used to suspend blocked goals and to understand the changing states of the world.

On examining the work on TRUCKER and RUNNER we realized that while we had been trying to solve the specific problem of recognizing execution-time opportunity we had actually built a more general mechanism to control planning and action. That is, in order to produce our planners and the desired execution-time opportunism, we had to built a general model of planning and action that was based on embedding the knowledge of plans and goals, as well as the control of action itself, in a memory-based understanding system.

## A model of Agency

Our process model is based on Martin's DMAP understander as well as its antecedent, Schank's *Dynamic Memory*. DMAP uses a memory organization defined by part/whole and abstraction relationships. Activations from environmentally supplied features are passed up through abstraction links and predictions are passed down through the parts of partially active concepts. When activations meet existing predictions, the node on which they meet is itself active. When all of the parts of a concept are activated, it is also activated. Subject to some constraints, when a concept has only some of its parts active, it sends predictions down its other parts. Unlike the initial versions of DMAP, RUNNER is not confined to concepts that are built out of linear sequences of features.

To accommodate action, we have added the notion of PERMISSIONS. PERMISSIONS are handed down the parts of plans to their actions. The only actions that can be executed are those that are PERMITTED by the activation of existing plans. Following McDermott (McDermott, 1978), we have also added the notion of POLICIES. POLICIES are statements of the ongoing goals of the agent. The only goals that are pursued are those generated out of the interaction between POLICIES and environmental features.

Most of the processing takes the form of recognizing circumstances in the external world as well as the policies, goals and plans of the agent. The recognition is then translated into action through the mediation of PERMISSIONS that are passed to physical as well as mental actions.

Roughly speaking, the activation of goals, plans and actions are as follows:

- Goals are generated through the interaction between existing policies and environmental features.

  For example, in RUNNER, the specific goal to HAVE COFFEE is generated when the system recognizes that it is morning. The goal itself rises out of the recognition of this state of affairs in combination with the fact that there is a policy in place to have coffee at certain times of the day.

- Most plans are generated out of existing ones. These are activated through the presence of new goals and environmental features. No new structure is created. Existing structures are simply activated.

  Any new MAKE-COFFEE plan is simply the activation of the sequence of actions associated with the existing MAKE-COFFEE plan in memory. It is recalled by RUNNER when the HAVE-COFFEE goal is active and the system recognizes that it is at home.

- Actions are **permitted** by plans and are associated with the descriptions of the world states appropriate to their performance. Once a set of features has an action associated with it, that set of features (in conjunct rather than as individual elements) is now predicted and can be recognized.

  The action of filling the coffee pot is **permitted** when the MAKE-COFFEE plan is active and is associated with the features of the pot being empty and the agent being near the pot. This means not only that the features are now predicted but also that their recognition will trigger the action.

- Actions are specialized by features in the environment and by internal states of the system. As with Firby's RAPs (Firby, 1989), particular states of the world determine particular methods for each general action.

  For example, the specifics of a GRASP would be determined by information taken from the world about the size, shape and location of the object being grasped.

- Action level conflicts are recognized and mediated using the same mechanism that recognizes information about the current state of the world.

  When two actions are active (such as those associated with filling the pot and filling the filter) this state of affairs is recognized as an instance of two actions being active, both requiring the full attention of the agent. This in turn activates a mediation action that selects one of the actions as the one to perform. During the initial phases of learning a plan, this can in turn be translated into a specialized recognition rule that will in fact always determine the ordering of the specific actions.

- Finally, suspended goals are also associated with the descriptions of the world states appropriate to their enablement.

For example, a new goal HAVE-ORANGE-JUICE, if blocked, can be placed in memory, associated with the conjunct of features that will allow its satisfaction, such as being at a store, having money and so forth. Once put into memory, this conjunct of features becomes one of the set that can now be recognized by the agent.

Along with these basic concepts, we are also exploring the issues involved with the recognition of situations that suggest particular plan modifications and interleaving as well as different failure types.

## A framework for the study of agency

In no sense do we see this model as a solution to the problems of planning and action. Instead, we see this as a framework in which to discuss the issues and construct content theories of the knowledge required of an agent in a changing world. We suggest, however, that there are certain aspects of this model that will greatly facilitate work on these issues. These include:

1. A unified representation of goals, plans, actions and conflict resolution strategies.
2. A natural framework for learning through specialization of general techniques.
3. A fully declarative representation that allows for meta-reasoning about the planner's own knowledge base.
4. A simple marker-passing scheme for recognition that is domain — and task — neutral.
5. The flexible execution of plans in the face of a changing environment.

Further, we find that the basic metaphor of action as permission and recognition, and planning as the construction of descriptions that an agent must now try to recognize as a precursor to action, fits our intuitions about agency in general. Under this metaphor, we can view research into agency as the exploration of the situations in the world that are valuable for an agent to recognize and respond to. In particular, we have examined and continue to explore content theories of:

- The conflicts between actions that rise out of resource and time restrictions as well as direct state conflicts and the strategies for resolving them.
- The types of physical failures that block execution and their repairs.
- The types of knowledge-state problems that block planning and their repairs.
- The circumstances that actually give rise to goals in the presence of existing policies.
- The possible ways in which existing plans can be merged into single sequences and the circumstances under which they can be applied.
- The types of reasoning errors that an agent can make and their repairs.
- The trade-offs that an agent has to make in dealing with its own limits.
- And the different ways in which a goal can be blocked and the resulting locations in memory where it should be placed.

Our goal is a content theory of agency. The architecture we suggest is simply the vessel for that content.

## RUNNER

Most of our activity in studying this architecture has been within the context of the RUNNER system. The RUNNER project is aimed at modeling the full spectrum of activity associated with an agent—goal generation, plan activation and modification, action execution, and resolution of plan and goal conflict—not just the more traditional aspect of plan generation alone.

### RUNNER's world

The agent in RUNNER currently resides in a simulated kitchen, and is concerned with the pursuit of such goals as simulated breakfast and coffee. Such commonplace goals and tasks interest us in part because they are repetitive and have many mutual interactions, both negative and positive. We are interested in how plans for recurring conjuncts of goals may be learned and refined, as part of view of domain expertise as knowledge of highly specific and well-tuned plans for the particular goal conjuncts that tend to co-occur in the domain [9]. We are also interested in the issue of how these plans can be used in the guidance of action.

### Knowledge Representation in RUNNER

The knowledge and memory of the agent is captured in the conjunction of three types of semantic nets, representing knowledge of goals, plans and states. Nodes in these networks are linked by abstraction and packaging links, as in DMAP. In addition, there is a special SUSPEND link, which connects suspended goals to state descriptions that may indicate opportunities for their satisfaction. In addition to being passed to abstractions of activated concepts, activation markers are always passed along SUSPEND links.

Currently the state and goal portions of the memory only employ activation markers, and are activated if they receive such a marker. The plan portion of memory employs two marker types, activation and permission, the latter of which plays a role analogous to prediction markers in DMAP. Activation markers are passed up abstraction links and as a result of the completion of concept sequences (below); permission markers are passed downward from activated plans to their subplans and actions. Plans and actions are not fully activated until both activation markers and permission markers have been received.

In general, the only other way in which these nets are interconnected is via *concept sequences.* A node may be activated if all of the nodes in one of its concept sequences is activated – a concept sequence for a given node can contain nodes from any of the parts of memory.

Concept sequences need some restrictions if they are not to represent the ability to stipulate that any node is activated by any other desired set of nodes. We want such a restriction, but at the level of a content theory of the important types of interactions of knowledge. Here is a partial taxonomy of the types of concept sequences we want to allow:

- Activation of a goal node can activate a node representing a default plan.
- Activation of a plan node and some set of state nodes can activate a further specialization of the plan.
- Activation of a goal node and some set of state nodes can activate a further specialization of the goal.
- Activation of any state node that has a SUSPEND link will activate the associated goal.

## An Example: Making Coffee

The above discussion of representation makes more sense in the context of an example, currently implemented in RUNNER, of how a particular action is suggested due to conjunction of plan activation and environmental input.

One of the objects in RUNNER's simulated kitchen is a coffee maker. Our agent starts off with a plan for making coffee that makes use of coffee-maker. This plan involves a number of subsidiary steps, some of which need not be ordered with respect to each other. Among the steps that are explicitly represented in the plan are: fetching unground beans from the refrigerator, putting the beans in the grinder, grinding the beans, moving a filter from a box of filters to the coffee maker, filling the coffee maker with water from the faucet, moving the ground beans from the grinder to the coffee maker, and turning the coffee maker on.

The subplans of the coffee plan are associated with that plan via packaging links. In this implemented example, the agent starts out with a node activated which represents knowledge that it is morning. This in turn is sufficient to activate the goal to have coffee (this is as close as the program comes to a theory of addiction). This goal in turn activates a generic plan to have coffee. This turns out to be nothing but an abstraction of several plans to acquire coffee, only one of which is the plan relevant to our kitchen.

"Visual" input, in terms of atomic descriptions of recognizable objects and their proximities, is passed to memory. For example, the agent "sees" the following visual types:

```
a stove, countertop, a white wall,
a glass, a box of filters
```

Among sets of possible visually recognized objects are concept sequences sufficient for recognition that the agent is in the kitchen. The recognition of the stove and the countertop completes one of these sequences. The "kitchen" node in turn passes activation markers to its abstractions, activating the node corresponding to the agent being at home.

```
MEMORY:
       sending activation marker to [wall]
  Activating concept: [wall]
       sending activation marker to [filter-box]
  Activating concept: [filter-box]
       sending activation marker to [countertop]
  Activating concept: [countertop]
  concept sequence ([ ** wall ** ]
                    [ ** countertop ** ])
     for node [in-kitchen] completed.
       sending activation marker to [in-kitchen]
  Activating concept: [in-kitchen]
       sending activation marker to [at-home]
  Activating concept: [at-home]
```

The activation of this node in conjunction with the activation of the generic coffee goal completes the concept sequence necessary for the goal for making coffee at home, which in turn activates the default plan for that goal. In this way a specialized plan is chosen in response to a conjunction of a recognized state and a more generic goal.

```
Concept sequence ([ ** GOAL: drink-coffee ** ]
                  [ ** at-home ** ])
 for node [GOAL: drink-coffee-at-home] completed.
   sending activation marker to
                 [GOAL: drink-coffee-at-home]
Activating concept: [GOAL: drink-coffee-at-home]
Asserting new goal:
          [ ** GOAL: drink-coffee-at-home ** ]
   sending activation marker to
                 [PLAN: make-coffee-at-home-plan]
Node [PLAN: make-coffee-at-home-plan]
      has both permission & activation:
 ((MARKER [ ** GOAL: drink-coffee-at-home ** ]))
  (TOP-LEVEL-PLAN)
Activating
concept: [PLAN: make-coffee-at-home-plan]
```

Activation of the coffee-plan sends permission markers to its steps, including the step of moving the filter. Now that the action has been both permitted and activated, it is suggested.

```
Asserting new plan --
      [ ** PLAN: make-coffee-at-home-plan ** ]
Sending permissions to steps of plan
Sending permission markers from
      [ ** PLAN: make-coffee-at-home-plan ** ]
to steps
 Sending permission marker to [PLAN: end-of-plan]
 Sending permission marker to [PLAN: move-5]
Sending permission marker to [PLAN: close-1]
Sending permission marker to [PLAN: turn-on-2]
Sending permission marker to [PLAN: move-4]
Sending permission marker to [PLAN: turn-on-1]
Sending permission marker to [PLAN: move-3]
Sending permission marker to [PLAN: move-2]
Sending permission marker to [PLAN: fill-1]
Sending permission marker to [PLAN: open-1]
Sending permission marker to [PLAN: move-1]
```

The activation of the other object concepts in turn have sent activation markers to the actions that contain them in their concept sequences. Among these is the plan step for taking a filter from the box and installing it in the coffee maker. This action is not suggested until it has received a permission marker from its parent plan.

```
Concept sequence
 ([ ** PLAN: make-coffee-at-home-plan ** ]
  [ ** filter-box ** ])
 for node [PLAN: move-2] completed.
sending activation marker to [PLAN: move-2]
Node [PLAN: move-2]
       has both permission & activation:
  ((MARKER
    ([ ** PLAN: make-coffee-at-home-plan ** ]
     [ ** filter-box ** ])))
  ((MARKER
     ([ ** PLAN: make-coffee-at-home-plan ** ])))
Activating concept: [PLAN: move-2]
Asserting new plan -- [ ** PLAN: move-2 ** ]
```

Though in its preliminary stages, the overall idea in RUNNER is that the processing of the visual types, goals, plans, and actions themselves is done within the confines of a single architecture reflecting the multiple hierarchies of an episodic memory.

## Conclusion

We've presented a sketch of an architecture for memory that we believe will be of use in exploring various issues of opportunism and flexible plan use. We do not view the architecture as a solution to the problems of interest, but instead as a framework that may be useful in exploring content theories of plan types, action suggestion and arbitration. As we said before, our goal is a content theory of agency. The architecture we suggest is simply the vessel for that content.

## References

[1] Phil Agre and David Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of the Sixth Annual Conference on Artificial Intelligence*, pages 268–72. AAAI [2], 1987.

[2] American Association for Artificial Intelligence. *Proceedings of the Sixth Annual Conference on Artificial Intelligence*, Seattle, Washington, July 1987. Morgan Kaufmann.

[3] American Association for Artificial Intelligence. *Proceedings of the Seventh Annual Conference on Artificial Intelligence*, Saint Paul, Minnesota, August 1988. Morgan Kaufmann.

[4] David Chapman. Nonlinear planning: A rigorous reconstruction. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 1022–24. IJCAI [11], 1985.

[5] R. J. Firby. Adaptive execution in complex dynamic worlds. Research Report 672, Yale University Computer Science Department, 1989.

[6] Kristian Hammond. *Case-Based Planning: Viewing Planning as a Memory Task*, volume 1 of *Perspectives in Artificial Intelligence*. Academic Press, 1989.

[7] Kristian Hammond. Opportunistic memory. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*. IJCAI [10], 1989.

[8] Kristian Hammond. Explaining and repairing plans that fail. *Artifical Intelligence Journal*, In Press.

[9] Kristian Hammond, Tim Converse, and Mitchell Marks. Learning from opportunities: Storing and reusing execution-time optimizations. In *Proceedings of the Seventh Annual Conference on Artificial Intelligence*, pages 536–40. AAAI [3], 1988.

[10] International Joint Conferences on Artificial Intelligence, Inc. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Michigan, August 1989. Morgan Kaufmann.

[11] Aravind Joshi, editor. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, California, August 1985. International Joint Conferences on Artificial Intelligence, Inc., Morgan Kaufmann.

[12] Charles E. Martin. *Direct Memory Access Parsing*. PhD thesis, Yale University Department of Computer Science, 1989.

[13] D. McDermott. Planning and acting. *Cognitive Science*, 2, 1978.

[14] Roger C. Schank. *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*. Cambridge University Press, 1982.

# Controlling Inference in Planning Systems: Who, What, When, Why, and How

**Steve Hanks**[*]
Department of Computer Science and Engineering
University of Washington, FR–35
Seattle, WA 98195
*hanks@cs.washington.edu*

## Abstract

Various proposals have been advanced over the past few years as to how decision analysis might be profitably applied to the problem of planning. Both the problem of choosing an appropriate plan and the related problem of choosing whether to plan or whether to act seem amenable to decision-analytic techniques. In this paper I explore several of these proposals and show how they differ in the simplifying assumptions they make either about the planning domain or about the sort of solution that decision analysis can be expected to deliver.

I conclude that none of the domain-simplifying assumptions proposed are appropriate for planning problems characterized by (1) a rich set of planning operators and (2) significant subgoal interactions, and go on to suggest that the proper place to apply decision analysis is not to the *selection* of plans but to the *comparison* of alternatives.

## 1 Introduction and disclaimers

This paper began as a panel presentation at the 1990 Spring Symposium on Planning in Uncertain Domains. The panel's title was "Controlling Inference." The panel's membership, however,[1] indicated a more specific agenda: each of us has explored an aspect of using decision analysis as a technique for making planning decisions. The question, then, becomes how a decision-analytic approach might help us to control inference in planning—in particular how it might allow us to limit "deliberative" or "lookahead" planning in some principled way.

My purpose is therefore to examine the use of decision-analytic methods as applied to controlling planning inference, and I will especially attend to approaches suggested by the other panel members. I am thus concerned with the following questions:

1. what inferences are we making as we plan,
2. what inferences are we controlling,
3. how are we doing so,
4. what compromises are we making in the interest of tractability,
5. are those the ones we *should* be making?

Several disclaimers need be advanced here to protect my credibility as well as friendships with various panel members. The first is that the nature of my presentation, and hence of this paper, was to stimulate discussion rather than to engage in deep analysis. The characterization of work will necessarily be brief and thus will oversimplify to some extent. Furthermore, I will be talking about work that was published some time ago (though mostly within the last couple of years), so the conclusions I draw may not reflect the way the authors are currently thinking about the problem. In fact several panel members later noted that their current work was addressing the sorts of concerns I raise below.

Finally, my view of the problem seems to be quite different from that of the other panel members: my background is in classical AI planning programs and paradigms, and I look at decision analysis as a technique we might use to help solve those problems. Others on the panel come to the problem through training in decision analysis—they view planning as a problem to which their technique may be applied. I think the difference in backgrounds and approaches explains much of the diversity in the positions I will be discussing below.

## 2 The planning problem

Since we are examining methods for controlling planning inference, a good place to start is by reflecting for a moment on the sorts of inferences we're supposed to be making (and thus controlling). In other words, we should try to define what we mean by planning in the first place. An obvious first cut might be

Given $\mathcal{I}$, my best course of future action is $\mathcal{P}$,

where $\mathcal{I}$ is some set of information or evidence and $\mathcal{P}$ is a course of action or plan. $\mathcal{I}$ might consist, for example, of sensory observations, of some model of the external world, of some current goals and plans to achieve those goals, and probably some characterization of our internal

[*]This research was supported in part by NSF grant CCR–8619663.

[1]The panel consisted of Tom Dean, Eric Horvitz, Andrew Mayer, Mike Wellman and I.

state of information about the state of our computation. We will talk more about this information state later on.

Plan $\mathcal{P}$ is what we used to call a series of "primitive actions," but now we realize that there is probably no such thing as truly primitive actions, or if there are, they describe the agent at such a low level, and there are so many, that we'd never be able to plan effectively using them. Instead we will call $\mathcal{P}$ a set of "instructions" to an execution system of some sort (I'm using the RAP system of [Firby 1989] as a model). We don't know exactly what these "instructions" will look like, but they will tend to be fairly vague and highly dependent on the execution-time context, and thus more difficult to project.

Of course this definition is entirely too vague in that it leaves open, in addition to the definition of $\mathcal{I}$ and $\mathcal{P}$, a definition of "best." As such, most any sort of decision problem can (and probably has been at some point) been characterized as planning. The question is how researchers in AI have characterized the problem.

Basically the two main assumptions have concerned the set of operators or instructions:

1. the set of operators is *fixed*,
2. the set of operators is *rich*.

The first just means that the planner has to work with a prespecified set of instructions. Its task is to string these instructions together, but not to synthesize new instructions. The first characterizes planning whereas the second characterizes design.

The second assumption is both more vague and more important to our analysis. It implies that the robot has a variety of skills at its disposal, and hence it will be able to solve most tasks in a number of different ways. This assumption is what sets a planning problem apart from a scheduling or routing problem. The latter involves knowing ahead of time exactly the operators to be applied and deciding on an order whereas the former involves *selecting* operators appropriately as well as determining their order.

Next we might wonder what makes planning difficult. What problems have researchers in automated planning focused on?

The second property of the planning problem suggests a one such issue: of appropriate operators or sequences thereof (sometimes called "strategies" or "methods"). Although the blocks-world domains were simple enough so that this problem was ignored, it *has* been recognized as one that must eventually be confronted, and we see progress on this front in more recent efforts, *e.g.* [Hammond 1986].

The problem that has received most attention in the planning literature is that of *subgoal interaction.* The problem arises in planning to satisfy conjunctive goals: if we are trying to achieve both P and Q it generally will not work to find a solution to P, independently find a solution to Q, and combine them in a naïve manner (by concatenating the two solutions for example). The whole notion of nonlinear planning, now more or less a standard in the community, grew out of this realization.

Thus the two issues most central to planning (as we have come to view the problem) are those of method selection and of subproblem combination. Those are the inferences we have to make in order to choose a "best" plan $\mathcal{P}$ for our state of information $\mathcal{I}$.

We must finally note how poorly understood is the process of planning. [Chapman 1987] points out that the problem is intractable in the worst case, even under the horrifying assumptions that characterize the microworld planners. Implemented planning programs such as SIPE [Wilkins 1988] and FORBIN [Dean *et al.* 1987] are too slow for a real-time agent, even given their simple models of the world. Attempts to speed up the planning process, like case-based or transformational algorithms ([Hammond 1986] and [Simmons 1988] respectively) rely on various *ad hoc* techniques, which will certainly turn out to be useful tools for building systems, but do not constitute a general solution to the problem (if indeed there is one).

## 3 Controlling planning inference

Now that we have some characterization of the inferences we'd like to make, we should move to the problem of *controlling* those inferences. In other words, when is our $\mathcal{I}$ sufficient to ensure that our $\mathcal{P}$ is good enough? Various techniques—gathering additional information about the world, generating more alternatives, reasoning in more detail about the effects of alternatives—might lead us to discover a $\mathcal{P}$' that turns out to be preferable to the current plan $\mathcal{P}$, or might cause us to prefer an alternative that earlier we passed over.

The general problem we face is whether to spend additional time deliberating (performing more information gathering or inference) on the hope that we can find a better $\mathcal{P}$'. We balance this hope against the cost associated with deliberation—that deliberation uses resources that could be put to better use elsewhere. Time is generally the resource in question, and to the extent that deliberation and action are mutually exclusive we must consider the opportunity cost of deliberation: if we wait too long to commit to plan $\mathcal{P}$ we may fail to realize its benefit, and if we deliberate too long instead of acting (or at least paying attention to our surroundings) we may be inviting catastrophe. We can thus pose the plan versus deliberate problem as a decision problem: spend a unit of time in deliberation if the expected marginal benefit of thinking exceeds the expected marginal cost of failing to act. And in fact each member of the panel has proposed applying this paradigm to the planning problem, in some form or to some extent.

But given our discussion of planning above it should be very clear that we cannot, in general, get a precise characterization of this tradeoff. We have no general-purpose planning algorithm at this point, so computing its "time derivative" would be guesswork at best. Computing the opportunity cost associated with failure to act involves sophisticated reasoning about the hypothetical unfolding of events and about the agent's capabilities. Finally we face a potential regress of decision problems: making the deliberate/act decision itself takes time, and a nontrivial amount of time at that, which must also be taken into account.

Clearly any proposal involving the mediation of deliberation and action must make some serious compromises. The question is, what compromises have researchers proposed (either explicitly or implicitly) and are these appropriate for our characterization of the AI planning problem? I will discuss this question by examining the proposals advanced by the panel members. Their approaches tend to divide into two groups. The first I will characterize with the slogan "redefine the problem so a solution becomes available," and into that group I will place Dean and Boddy [1989], Horvitz and his group [Horvitz *et al.* 1989] and Mayer and his group [Hansson *et al.* 1990]. The second group, "restrict the notion of a solution until it can be applied to the problem," includes Wellman [1988] and myself [Hanks 1990].

## 4 Redefining the problem

Here will attempt a quick review of the research efforts whose success may depend, to some extent, on somehow limiting the scope of the planning problem.

### 4.1 Anytime algorithms

Perhaps the best-known work in the area of controlling plan execution through decision-theoretic analysis is the research into "anytime algorithms," appearing in [Dean and Boddy 1988] and [Boddy and Dean 1989]. Anytime algorithms have three important properties:

1. they can be interrupted at any time and will produce *some* solution to the problem,
2. given more time they will tend to produce better solutions,
3. whoever is using the anytime algorithm has some explicit characterization of the tradeoff between the algorithm's performance (that is, quality of solution) and the amount of time the algorithm is given to compute the solution.

The third item is, of course, exactly the information we need in order to do the expected-utility analysis—deciding whether to plan or whether to act. So we can solve the inference-control problem if we can couch the problem as an anytime algorithm *and* if we have a good characterization of the opportunity cost associated with delaying action.

The question then arises as to what sorts of problems might be amenable to "anytime analysis." Boddy and Dean give several examples, mostly having the flavor of more-or-less-traditional optimization problems like vehicle routing, scheduling, and so forth. Now these sorts of problems are certainly *components* of a planning problem—they unquestionably arise in the process of planning. The question is whether they *in and of themselves* cover the space of planning problems. I argue that they do not, mainly because they fail to address the problem of operator or method selection, which I noted above was central to the problem of planning.

Boddy and Dean [1989] recognize this shortcoming, of course, and propose an architecture that involves breaking down the planning problem into many "anytime" problems, then introducing a control structure that optimally allocates time to each one and coordinates their results. The problem with this approach is that it lets the planning problem "in the back door," as it were, as the need to manage the interactions among solutions to the subproblems. As I mentioned above, years of planning research has taught us that subproblem interaction is a crucial part of the planning process. It's hard to imagine why this problem is not going to re-emerge as we try to coordinate the outputs of anytime algorithms.

And if that's the case—if in composing the outputs of anytime algorithms we lose that crisp characterization of the time-versus-performance tradeoff—it's not clear what is left of the theory, apart from the insight that planning algorithms should be flexible enough to *realize* such a tradeoff. We're left, in effect with the first "anytime property," but it remains to be demonstrated whether the second and third can be revived, and thus to what extent the "anytime architecture" represents a solution to the deliberation/action tradeoff.

### 4.2 Decision-making in high-stakes domains

Moving now to the work of Horvitz and his group at Stanford, presented for example in [Horvitz 1988] and [Horvitz *et al.* 1989], I should note that it's perhaps unfair to accuse this effort of "redefining the problem," since I don't think the Stanford group ever claimed to be doing planning in the same style as Dean or Boddy or I claim to be doing.

Horvitz's group is working on a class of problems they call "high stakes decision problems," exemplars of which are found in the domain of medical decision making. These problems are quite different from those faced by, say, an errand-running robot. First of all, the problems tend to be more complex, as the system being studied is usually the human body. Second, the problems tend to be more static: the decision options are known ahead of time, and it's less crucial to have an explicit model of time and change like those that have characterized recent "temporal" planners. Finally, as the name, implies, the stakes are higher. If our robot buys ice cream first and upon returning home finds it's melted (because it had always executed this plan in winter before and had neglected to take the ambient temperature into account), we're likely to say "live and learn." Contrast this with a medical treatment plan and neglecting interactions among drugs.

So Horvitz has much more of an incentive to get things right (or better) than we do. Computing the tradeoff between marginal improvement of solution quality versus inference time then becomes much more of an issue, simply because incremental improvements are *themselves* much more important. We want our errand-running robot to avoid disasters (running down people) and demonstrably stupid behavior (making ten separate trips to the same store), but if it takes a slightly suboptimal route we really don't care.

But Horvitz is working on a somewhat more structured problem as well. That problem (see, for example [Horvitz *et al.* 1989]) is that of doing the expected-utility analysis on an *influence diagram* that is provided as an input. An influence diagram is a graph in which some nodes represent propositions (states of the system), some nodes represent decisions (things the agent can do to change the state of the system), and whose arcs represent and quantify the influences among propositions and between decisions and propositions.

Since the graph contains a node for each possible decision, all the decision options (possible plans) are coded into the problem statement itself. This option is simply not feasible, in general, for those of us doing "traditional AI planning." We are at least as concerned with the problem of *structuring* the problem (by which I again mean selecting from a rich set of methods) as we are with ultimately solving the problem. Now it may be the case that after we *do* the structuring we will want to undertake an analysis like Horvitz does, but personally I doubt it, for reasons I will discuss below.

### 4.3 The Bayesian problem solver

The research proposal of [Hansson *et al.* 1990] is hard to evaluate: since the effort is in its early stages the authors present no empirical results, thus they may not have had to make compromises in the interest of efficiency.

The general idea is to view the planning problem as a state-space search where nodes represent states of the world and arcs represent the hypothetical execution of a plan, where the plan may be expressed at various levels of abstraction. The search frontier is expanded by committing to actions or otherwise refining the plan. And the decision problem, of course, is that of action selection: which action should next be tried, or, equivalently, how should the search frontier be expanded. To control search they associate with each arc a heuristic cost function—actually a probability distribution over the amount it will cost to move the system from the source state (node) to the destination or goal state (node). The planner can refine a plan or gather information, both of which will tend to lower the distribution's variance.

One question that arises is whether this notion of selecting planning strategies according to the cost distributions will be effective in mediating the selection of actions. It will be effective to the extent that (1) the distributions associated with alternative choices suggest a clear choice (which will tend to be the case only if their variances are low) and (2) the cost distributions can be calculated quickly. It is not clear at this point why either (1) the planning problem will not re-appear in the problem of computing cost distributions or (2) the cost distributions will be so crude that they will not effectively limit the search. The "Bayesian Problem Solver" architecture has been tried on traditional search domains like the eight-puzzle; the question is whether it will scale up to the rich set of actions we (would like to) associate with planning problems. Another question is whether the architecture's state-space orientation can be extended to temporally rich domains (*e.g.* actions that take time, deadlines) which tend to exacerbate the action-selection problem.

## 5 Lowering expectations

Now let's move to the "lowered expectations" camp, members of which tend to look for ways to apply decision analysis in a more modest fashion, for example by applying numeric methods to some subproblem in conjunction with other symbolic problem-solving methods.

### 5.1 SUDO-planner

A good example of this approach is Wellman's [1988] SUDO-planner. This program looks not to choose the optimal plan, but instead to look for and eliminate classes of *dominated* plans. This approach can be viewed as the probabilistic analogue of pruning infeasible plans. Focusing on dominance relationships allows not only for faster computation, but also for a restricted qualitative language (qualitative probabilistic networks) that may be more plausibly assessed.

### 5.2 Probabilistic projection

My work [Hanks 1990] applies decision theory *not* to the process of generating alternative plans, but rather to the more focused tasks of debugging a proposed plan, comparing alternative plans, and discovering what information would have to be gathered in order to convince the planner that a plan is reasonable or preferable to another. In other words, a planner is responsible for generating alternatives, but then probabilistic projection allows it to compare, refine, or modify the alternatives according to decision-theoretic criteria.

The general task I've undertaken is to maintain the planner's *world model*, which consists of a network of temporally scoped, interconnected, probabilistic *beliefs*, representing what the agent believes to be true at various times, past present and future. A planner asks questions about the world model, and answers are returned as beliefs, which are then used to select plans, to debug them, and to compare alternatives.

The basic architecture, which is due to Jim Firby and myself,[2] consists of three main modules: the execution system, the planner, and the projector or world-model manager. See Figure 1.

The execution system is described in [Firby 1989]. It executes instructions down to the sensor/effector level, and is "reactive" in the sense that it has an analogue to the first "anytime" property: the planner can provide it with instructions expressed at essentially any level of abstraction, and the execution system will do *something* with them. What it does, however, may be shortsighted in that the execution system does no lookahead and only a very limited form of action selection. More important, however, is that the execution system is controlled by the (symbolic) instruction library, which can be examined by the planner and projector. Thus the reactive system's performance can be modeled and biased by the "deliberative" modules of the system. This is a major difference between our architecture and other "reactive planners," whose behavior tends to be quite opaque.

The planner, which has not been implemented at this writing, is responsible for generating some small number of plan alternatives. It can use any combination of traditional planning techniques, for example retrieving cases from memory, applying optimizing or repairing transformations, or constructing plan fragments from more primitive instructions.

[2]See [Firby 1989] as well.

To do so the planner will have to ask questions about the expected effects of executing a plan. It does so by posting queries to the world-model manager (projector), which returns answers in the form of beliefs as described above. The planner bases planning decisions (*e.g.* to commit to one course of action over another) on these beliefs.

The projector's job is to compute the probability that a proposition will be true at some (future) point in time. Its computation is driven by the planner's queries, which are of the form "is the probability that proposition $\varphi$ will be true at time t greater than threshold value $\tau$?" These queries will typically involve questions about a plan's effectiveness in achieving the goal, or about how efficiently it does so. The point is that the projector does only the inference necessary to answer the query, which is to say only the inference the planner needs in order to make decisions like choosing a plan, selecting a transformation, or committing to one alternative over another.

Note that the query determines not only what aspects of a plan's execution are important, but also (through the threshold) *how* important the answer is. More restrictive thresholds will tend to take more inferential work to verify.

Projection can also point out what important facts the planner *doesn't* know, those that prevent it from knowing which alternative is preferable. These results can indicate to the planner the need for information-gathering actions.

Projection and planning are thus interleaved and iterative. The planner suggests several vague alternatives, but the projector can only give vague answers to queries, which will not lead the planner to favor one alternative over another. The planner can then transform a plan by refining it (then asking more questions) or it can improve its state of information by scheduling appropriate sensing actions which eventually show up in the world model and are reflected in subsequent projections.

The projector is also responsible for maintaining the integrity of the information it passes back to the planner: if subsequent information (sensory observations or new plans) invalidates a particular prediction the projector will notice and notify the planner, allowing it to re-plan if necessary.

So the way we control inference is extremely simple: we require the planner to ask very focused questions, and we do only the inference necessary to answer them. Note that we do no "inferential performance versus inferential time" tradeoff—neither the planner nor the projector have the "anytime" property. On the other hand, incremental queries to the projector tend to be extremely fast, so it's not clear that such an analysis would be fruitful anyway.

## 6 Conclusions

So my conclusions are the following: it's pointless to apply decision analysis to the general planning problem. We have no good characterization of the performance of action-selection strategies, nor do we have a good way of computing the opportunity cost of failing to act. Even if we understood these computations they could not possibly be effected quickly enough to do us any good.

So we have to decide which we're going to give up, planning or decision theory. Most have taken the option of giving up planning, at least as I view the problem. Personally I'm all for giving up full-blown decision analysis. I find that the methodology has nothing to offer regarding the problem of operator selection. What it *does* offer, however, is a rigorous and focused way of *comparing* alternative courses of action. And the way it does so, by dictating acceptable probability/utility tradeoffs, proves quite valuable in limiting the time spent in "planful deliberation."

## References

[Boddy and Dean 1989] Mark Boddy and Thomas Dean. Solving time-dependent planning problems. In *Proceedings IJCAI*. AAAI, August 1989.

[Chapman 1987] David Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32(3):333–378, 1987.

[Dean and Boddy 1988] Thomas Dean and Mark Boddy. An analysis of time-dependent planning. In *Proceedings AAAI*, pages 49–54, 1988.

[Dean *et al.* 1987] Thomas Dean, R. James Firby, and David Miller. The FORBIN paper. Technical Report 550, Yale University, Department of Computer Science, July 1987.

[Firby 1989] R. James Firby. Adaptive execution in complex dynamic worlds. Technical Report 672, Yale University, Department of Computer Science, January 1989.

[Hammond 1986] Kristian Hammond. Case-based planning: An integrated theory of planning, learning, and memory. Technical Report 488, Yale University, Department of Computer Science, October 1986.

[Hanks 1990] Steven Hanks. Projecting plans for uncertain worlds. Technical Report 756, Yale University, Department of Computer Science, January 1990.

[Hansson *et al.* 1990] Othar Hansson, Andrew Mayer, and Stuart Russell. Decision-theoretic planning in BPS, 1990. This volume.

[Horvitz *et al.* 1989] Eric J. Horvitz, Gregory F. Cooper, and David E. Heckerman. Reflection and action under scarce resources: Theoretical principles and empirical study. In *Proceedings IJCAI*, pages 1121–1127, 1989.

[Horvitz 1988] Eric J. Horvitz. Reasoning under varying and uncertain resource constraints. In *Proceedings AAAI*, pages 111–116, 1988.

[Simmons 1988] Reid G. Simmons. Combining associational and causal reasoning to solve interpretation and planning problems. Technical Report 1048, MIT Artificial Intelligence Laboratory, September 1988.

[Wellman 1988] Michael P. Wellman. Formulation of tradeoffs in planning under uncertainty. Technical Report MIT/LCS/TR-427, MIT Laboratory for Computer Science, August 1988.

[Wilkins 1988] David E. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan-Kaufmann, 1988.

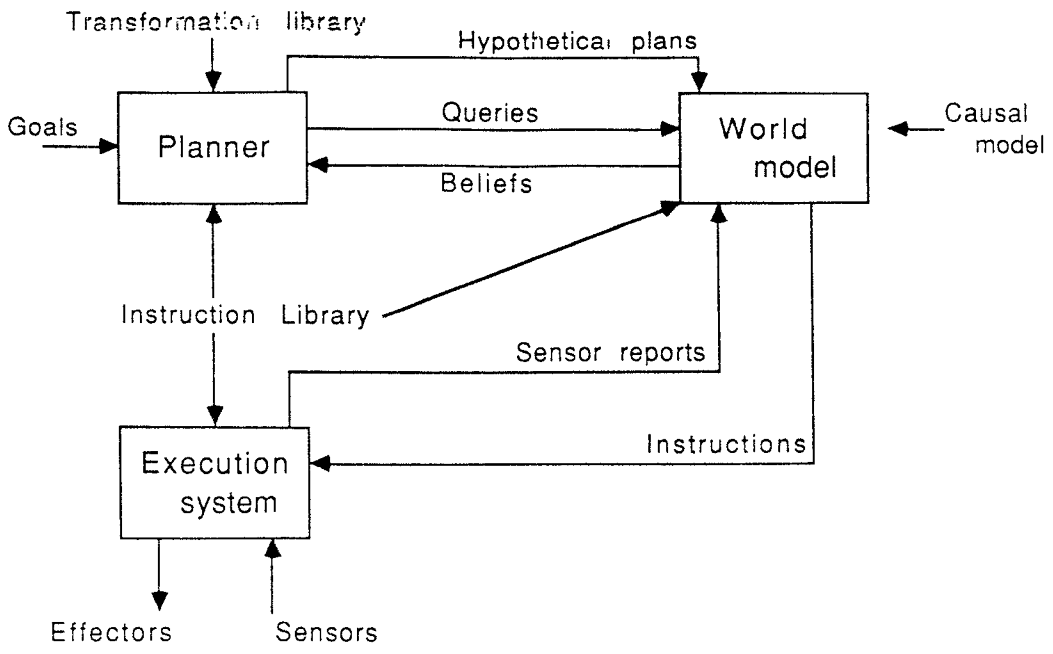


Figure 1: An architecture for planning, projection, and execution

# Decision-Theoretic Planning in BPS[*]

Othar Hansson Andrew Mayer Stuart Russell

Computer Science Division
University of California
Berkeley, CA 94720

## Abstract

The BPS (Bayesian Problem-Solver) project aims to reconstruct and extend AI methods using normative decision-making principles. Earlier research showed that traditional heuristic search methods could be subsumed by such a normative approach, implemented as probabilistic inference on a Bayesian network corresponding to the problem-space. This paper sketches an extension of the BPS approach to planning domains. By admitting abstract states (corresponding to sets of states in the problem-space) to the problem description, we show that advanced control techniques, such as goal-directed search, abstraction, and goal reduction, appear in BPS without requiring explicit encoding. Rather, they emerge naturally from groups of primitive search control decisions. Problems of "situatedness" such as expectation violations, lack of information, and time pressure cause severe difficulties for logical and search-based planning frameworks, but seem to require little additional structure in our approach.

## 1 Planning and Action Selection

McCarthy's original paper on formal reasoning for AI systems [6] addressed the question of deciding what an agent *should do next.* We will refer this as the *action selection problem.* In his "implementation example", the problem was solved by proving that a certain sequence of actions would result in a desirable state, and should therefore be executed. This method became the paradigm for planning research. Within this context, several technical problems have been studied. What might be called "inference problems" centered around the representation and use of domain knowledge to construct plans that actually achieved the specified goal. These problems will not be focused on here. Another class, "control problems", involved mechanisms for reducing the complexity of planning search. *Goal reduction,* in which solutions to goal conjuncts are found separately and then stitched together to form complete solutions, has given rise to a large number of techniques [2] and is the main feature distinguishing the field of planning from that of heuristic search. Another such feature is the widespread use of *goal-directed* generation of intermediate state descriptions. Abstraction [11] works by reducing problem-space size, at the risk of generating partial solutions that cannot be completed.

Unfortunately, the collection of methods which has evolved, while capturing valuable insights, is not the product of a coherent theory. We will argue that a Bayesian action selection system can preserve the advantages of these mechanisms without undue complication.

Planning, viewed as the identification of action sequences leading to goal states, is only one way to select actions. There has been some debate over different approaches to the action selection problem [1]. Rather than coming down on one side of the debate or another, we will simply make some observations that are accepted by all parties.

- Consideration of the consequences of actions (that is, lookahead) is necessary in some domains.
- Logical planning techniques are not applicable when only non-deterministic domain theories are available.
- Even in deterministic domains, finding guaranteed plans can be unreasonably expensive.
- It is often valuable to express preferences among otherwise correct plans.

We believe it is possible to state and solve the action selection problem in such a way that these concerns are addressed.

In the next section, we review the application of the BPS methodology to the action selection problem in

[*]This research was made possible by support from Heuristicrats, the National Aeronautics and Space Administration, and the National Science Foundation (NSF IRI 8903146)

standard search domains. We then outline the extensions needed to handle planning domains, and illustrate the ideas with a simple travel planning example.

## 2 Bayesian Problem-Solving

BPS is a decision-theoretic action selection system; that is, it chooses actions according to the principle of maximum expected utility [12]. The initial version of BPS was applied to the domain of heuristic search, with the intent of demonstrating that this class of problems could be solved by a system which explictly follows the prescriptive axioms of Bayesian decision theory. This avoids the *ad hoc* assumption made in standard game-playing and problem-solving algorithms that the heuristic evaluations of non-terminal leaf nodes are exact. Instead, BPS treats heuristic evaluations as probabilistic evidence regarding the true values of states, thereby casting the search problem as a standard problem of inference under uncertainty.

As BPS searches, it incrementally constructs a Bayesian network in which each state in the partially expanded state-space is represented by a variable node corresponding to its true cost, together with an evidence node corresponding to the heuristic evaluation of that state. Because standard evidence propagation methods [7] can be used to continually update this network as new states are expanded and evaluated during search, BPS can maintain a *belief* for each state it has expanded (a probability distribution over the state's possible outcomes conditioned on all heuristic evidence observed throughout the state-space). Within this representation, it is straightforward to compute the expected utility of an available action.

In addition to heuristic information, BPS can use local consistency constraints on neighboring variable nodes and neighboring evidence nodes in order to achieve a consistent global interpretation of the heuristic evidence. This illustrates a further advantage of the approach: additional knowledge can be incorporated in the system to improve performance without changing the basic inference mechanism.

Experiments on the eight-puzzle problem have shown this normative decision procedure to outperform the best known traditional search algorithm: despite searching only a few hundred states, BPS made correct decisions as often as the traditional algorithm, which examined several million states [5].

An additional benefit of the explicit representation of uncertainty is the possibility of using information-value theory [8] to control state expansion [3; 9]. The theory treats computations as actions, selecting among them according to expected utility – essentially, relevance to the quality of the final decision. In the face of time pressure, the expected utility of computation (further search) can be compared to the cost of delaying action, in order to limit deliberation to exactly the amount appropriate to the situation.

## 3 Applying BPS to Abstraction Spaces

In the eight-puzzle experiments, the operation of BPS was limited to a single deliberation phase followed by selection of the best immediately available action. Extending the BPS system to perform as a situated planning agent requires only that we allow the system to carry out actions and possibly update its evidence nodes with new sensory information; that we represent abstract states and possible future actions in the Bayesian network; and that we encode the dependencies among costs of "plans" at different levels of abstraction. Having done this, the inference and selective search components of BPS can be applied directly to the resulting network. The Bayesian network makes clear the only purpose in engaging in any search or planning activity: gathering information to refine the expected utility estimates of competing available actions, in order to better discriminate among them.

We use the word "plan" to denote not a particular sequence of actions, but the information about possible future actions which is encoded in the Bayesian network. The word "goal" simply describes an abstract state whose utility is well specified. The constraints of the Bayesian network propagate information from goals and abstract plans to the immediately available actions. For simplicity, we will use the word "distance" when, in fact, multi-attribute utility estimates are being computed. In addition, a simpler graphical description of the variables and dependencies in a Bayesian network will suffice for the discussion. The arcs in the diagrams below correspond to variable nodes in Bayesian networks, and the dependencies of the Bayesian network are implicit in the graph structure.

In what follows, we will first describe some of the gross changes the network might undergo as deliberation progresses. We will then discuss how the system selects among its primitive graph operations to produce these changes.

### 3.1 Base-Level States

To understand the representation of abstract plans, we first consider the representation of base-level states and operators. Figure 1 shows the initial plan network given to BPS – from state $S_0$, BPS is to choose to move to either state $S_1$ or state $S_2$. The solid arrows in the figure correspond to these concrete actions, which have costs $b$ and $c$ respectively. To maximize expected utility, the action selection must be informed by some estimate of the cost of moving from either $S_1$ or $S_2$ to the goal state. These as yet unspecified plans, denoted by broken arrows, have costs $d$, $e$, with uncertainty as illustrated. Initially, these are simply uninformed prior estimates.

In contrast to their use in traditional search algorithms, heuristic evaluations are not taken at face value, but are viewed in the light of the triangle-inequality constraints between the distances $a, b, c, d, e$

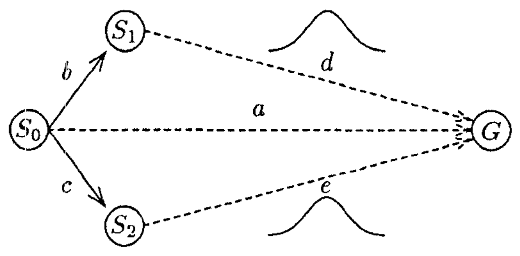


Figure 1: Initial Network

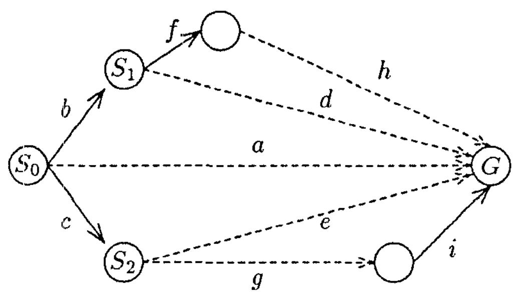


Figure 2: Acquisition of information by search

in the graph. These constraints, or probabilistic dependencies, allow us to propagate evidence through the graph. For example, further computation, such as a heuristic evaluation, that yielded a more precise estimate of the distance $d$ would indirectly reduce uncertainty in the distance $e$.

## 3.2 Forward and Backward Chaining

We can gather further information about $d$ and $e$ by searching the base-space and evaluating other states. For example, in Figure 2, we have forward-chained from $S_1$, adding an estimate $h$ to the goal state, and backward-chained from the goal state, adding an estimate $g$ to $S_2$. Either of these operations provides constraining information to reduce our uncertainty in $d$ and $e$.

## 3.3 Abstraction

Heuristic evaluations provide perhaps the simplest example of the value of abstraction in problem-solving [4]. An agent is repeatedly faced with the problem of estimating the cost of moving from one state (e.g., $S_1$) to another (e.g., $G$). If nothing is known about the distance between this particular pair of states, it may be useful to classify them provided something is known about distances between random instances of the two classes. The dependency between that estimate and the base-level distance estimate can be expressed directly in the Bayesian network.

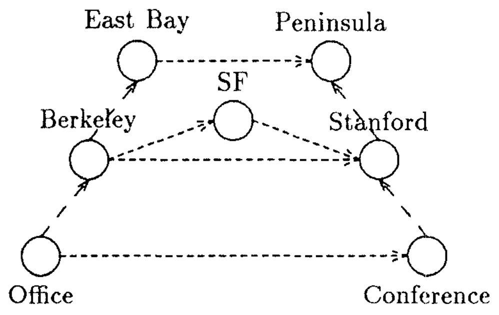


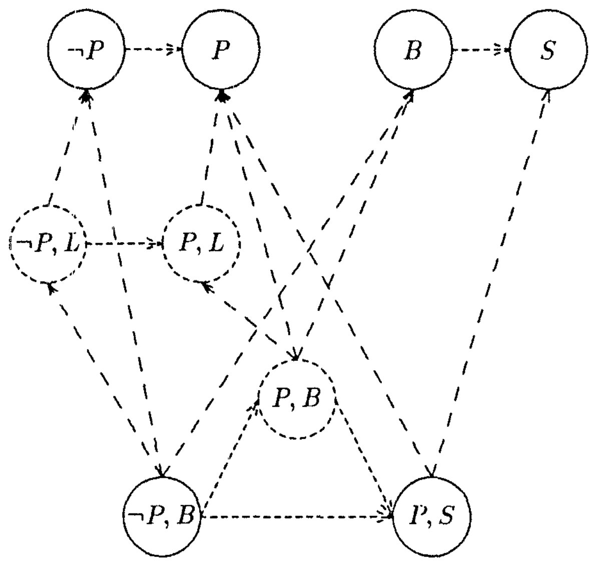
Figure 3: Acquisition of information by abstraction

Consider, for example, the travel-planning problem faced by the authors, who must choose an initial action from their office in Berkeley, in an attempt to reach the location of this conference. We may know nothing *per se* about the distance from our office to the conference location, but we may know something about distance between two arbitrary points in the abstract classes "Berkeley" and "Stanford", perhaps learned through experience (Figure 3). Alternately, we may know only the distances from each campus to San Francisco, requiring further elaboration of the abstract plan to reduce uncertainty in the base-space. Or we may know only the distances between the even more general classes "East Bay" and "Peninsula", requiring yet another level of abstraction.

BPS views the segments of these abstract plans as operators in an abstract space. The only distinction between these abstract operators and those at the base-level is that the former have less predictable resource requirements – perhaps even to the point of infeasibility. However, by elaborating the abstract plans this uncertainty can be reduced. For example, our utility function may dictate that we guarantee arrival at the conference by the time of the first coffee break, but our cost estimate between the two cities may still be very uncertain. To help choose between the available directions at an intersection in Berkeley, we may elaborate the abstract plan by specializing it, and then choose (implicitly, in the network calculations) to cross, e.g., the most predictable of the three bridges across the bay between the campuses.

## 3.4 Information-Value and Control

Within the BPS architecture, abstract plans are represented simply as states and operators, which are indistinguishable from base-level states and operators – the probabilistic inference and decision-theoretic control mechanisms see a completely homogeneous graph of variables and dependencies. Search in the plan network may therefore proceed exactly as within the earlier version of BPS. Uncertainty in the actual cost of an abstract plan can be represented explicitly, and choices among, e.g., expanding a base-level node, expanding an abstract node, abstracting a node or elaborating



| | |
|---|---|
| $P$ : | $possess(Agent, Permit)$ |
| $B$ : | $loc(Agent, Berkeley)$ |
| $S$ : | $loc(Agent, Stanford)$ |
| $L$ : | $loc(Agent, x) \wedge loc(Permit, x)$ |

Figure 4: Resolving Conjunctive Goals

an abstract plan, can be controlled naturally by the decision-theoretic meta-level. Classical approaches to planning require committing *a priori* to a restricted set of complex control strategies. The decision-theoretic approach suggests that familiar planning behaviors are emergent properties of individual search control decisions at a much finer grain.

We have space here to consider only a simple example of such emergent behavior. In it, the reduction of a conjunctive goal occurs, an intermediate abstract state is generated as a subgoal, and plans for two conjuncts are ordered to avoid precondition violations. None of these behaviors need to be programmed explicitly.

Consider the following simple extension to our planning example. We must get to Stanford for the symposium, but we must remember (i.e., plan) to bring the parking permit which is in the Berkeley office. We must, despite our rush to not miss the coffee break, do enough planning to realize this precondition.

Imagine that our current plan network contains only the solid-outlined nodes of the diagram in Figure 4. In other words, we know that we must get from a state in which we are in Berkeley and do not have the parking permit, to one in which we are in Stanford with the parking permit.

The estimated cost of the abstract operator between these two states would be influenced by the great uncertainty associated between moving from a more general state ($\neg P$) in which we do not have a parking permit to a state ($P$) in which we do (this could, in turn, be inherited from an even more general possession operator).

Assuming that the information-value mechanism judges that a little planning is worth the cost, we would probably try to reduce the uncertainty in establishing possession of the parking permit. One way of doing so would involve expanding the nodes ($\neg P$) and ($P$), reflecting the abstract description of a "pick-up" operator.

A further information-rich planning step would be to create the link between ($\neg P, L$) and ($\neg P, B$), i.e., realizing that the initial state satisfies the preconditions (the example is not complicated much by requiring "subgoaling" to establish the preconditions).

The final step, of specializing ($P, L$) to ($P, B$), and seeing that the cost of moving from ($P, B$) to ($P, S$) is much less than the arc from ($\neg P, B$) to ($P, S$), would greatly reduce the expected cost of any plan which had previously "included" the arc from ($\neg P, B$) to ($P, S$). Consequently, the expected utility of any available action that could lead to that plan segment would be greatly increased by these few steps of planning.

Note that once the "expensive" conjunct (possessing the permit) has been achieved cheaply by linking it to the current state, the ordering among the conjuncts is implicitly achieved. Note also that it is the possibly high expense of achieving two conjuncts that spurs planning to decide on their ordering. Two conjuncts which are cheap need not trigger any planning – if we simply required a pen rather than a parking permit at the conference, we should not bother to plan to acquire one before leaving.

## 4 Conclusions and Further Work

There seem to be four potential contributions of this approach:

1. Decision-theoretic action selection allows the planning system to deal with uncertainty and conflicting goals via the mechanisms of probability and utility.

2. Abstract states and goals are represented as nodes in a belief network, just like ordinary states; this allows a uniform planning mechanism with the possibility for arbitrary mixtures of forward and backward reasoning, island-driving, means-ends analysis and so on.

3. Standard information-value calculations can generate these behaviors automatically, and can provide appropriate tailoring of reasoning effort to the situation.

4. A situated Bayesian planner can easily adapt to new information that may violate its 'expectations' or create new opportunities.

We can view the Bayesian network that a BPS planner develops as a partial, abstract model of its future environment and intentions. The amount of detail and scope of the model varies according to the situation, and is governed by information-value calculations; there will usually be concrete detail regarding the immediate future. As time passes, BPS "moves through" its model, extending its horizon and updating its intentions as evidence is gathered. In the extreme of unlimited computation time and deterministic domain knowledge, BPS will act as a utility-maximizing planner, since it will initially construct a network corresponding to a guaranteed plan, and will then move along it with no need for recomputation. In the extreme of critical time pressure, information-value calculations will limit lookahead to the amount needed to produce an informed decision – that is, reactive behavior. BPS will therefore sometimes look like a planner, but it must be kept in mind that a new best action is selected at each stage, and the previously-calculated Bayesian network is always subject to updating.

Some hard problems, common to many systems, still have to be solved:

- Currently, no probabilistic formalism exists that will allow us to integrate the system's domain knowledge (e.g., the effects of actions) with the plan representation. Furthermore, we do not expect to escape from the frame problem, representation engineering of the abstraction hierarchy, or interval book-keeping.
- Under time pressure, the system should be able to fall back on compiled strategies [10]. It is not clear how to generate and integrate these within the standard decision system.
- Since information-value theory has never been applied to such complex decision systems, further conceptual development is necessary. Myopic formulations [7] in particular may be inadequate in planning domains.

In summary, decision-theoretic approaches to planning seem a promising way to satisfy the goals of all sides of the "planning debate". Despite the parsimony of the basic principles, when bounded resources come into play a rich structure emerges, generated by the application of those same principles to the selection of computations.

## References

[1] P. Agre and D. Chapman. Pengo: An Implementation of a Theory of Activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, 1987.

[2] D. Chapman. Planning for Conjunctive Goals. *Artificial Intelligence*, 32:333–377, 1987.

[3] O. Hansson and A. Mayer. Decision-Theoretic Control of Search in BPS. In *Proceedings of the AAAI Spring Symposium on Limited Rationality*, Palo Alto, 1989.

[4] O. Hansson and A. Mayer. Subgoal Generation from Problem Relaxation. In *Proceedings of the AAAI Spring Symposium on Planning and Search*, Palo Alto, 1989.

[5] O. Hansson and A. Mayer. Heuristic Search as Evidential Reasoning. In *Proceedings of the Fifth AAAI Workshop on Uncertainty in AI*, Windsor, Ontario, 1989.

[6] J. McCarthy. Programs with Common Sense. In *Readings in Knowledge Representation* (R. J. Brachman and H. J. Levesque, eds.), Morgan Kaufmann, San Mateo, CA, 1985.

[7] J. Pearl. *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, San Mateo, CA, 1988.

[8] H. Raiffa and R. Schlaifer. *Applied Statistical Decision Theory.* Harvard University, 1961.

[9] S. J. Russell and E. Wefald. Principles of Metareasoning. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, Toronto, 1989.

[10] S. J. Russell. Execution Architectures and Compilation. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, 1989.

[11] E. D. Sacerdoti. Planning in a Hierarchy of Abstraction Spaces. *Artificial Intelligence*, 5:115–135, 1974.

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

# Dynamic Control Planning in Intelligent Agents

Barbara Hayes-Roth

Knowledge Systems Laboratory
Stanford University
Palo Alto, California 94304

## 1. Control in Intelligent Agents

Intelligent agents perform multiple interacting tasks in uncertain, dynamic environments under resource limitations. They have capabilities for perceiving information from the environment, performing reasoning tasks (interpret perceived information, diagnose exceptional events, predict future events, and plan actions), and performing actions that impact on the environment. Performance of a given task involves execution of many component operations in some temporal pattern. Although the agent continuously senses information from the environment, the information is noisy, incomplete, and perishable. Real-time constraints imposed by dynamic phenomena in the environment limit the utility of even logically correct actions.

At any point in time, an intelligent agent theoretically is capable of performing many different operations involved in various perception, reasoning, and action tasks. Because it has limited resources, the agent can perform only a subset of its potential operations. Thus, it must perform a control task, determining which of its potential operations to perform at each point in time. Control decisions for a given task determine whether or not the agent achieves the goal, what resources it consumes, what side effects it produces, and the apparent orderliness of its behavior to a human observer. The agent's method of making control decisions determines the range of problems it can solve, the range of problem-solving strategies it can apply, its flexibility in adapting to run-time conditions, and its ability to explain its behavior to a human user.

Even for a single, static task, an agent's approach to control plays a critical role in its performance (Garvey87). With multiple tasks in a dynamic environment, the control task is more challenging. In its efforts to meet global objectives, the agent must coordinate interacting tasks, as well as control its selection and execution of operators within separate tasks. It must take into account asynchronously sensed information, balancing adaptation to new demands and opportunities with purposeful pursuit of previously established goals. Finally, the agent must take into account its own limited computational resources and real-time constraints on the utility of its actions imposed by the environment.

Traditional planning paradigms allow agents to generate and execute coherent longer-term courses of action, but do not allow them to adapt quickly to unanticipated events. They also pose unrealistic computational demands. More recently developed reactive paradigms provide flexibility, but no global coherence. They also pose unrealistic storage demands. The proposed approach aims to integrate both capabilities under a uniform and flexible control architecture.

## 2. Our Approach

Our approach to control in intelligent agents elaborates the concept of *dynamic control planning* proposed in (Hayes-Roth and Hayes-Roth, 1979; Hayes-Roth, 1985). Taking into account the dynamic environment, the agent incrementally constructs and modifies explicit control plans that guide its choice among potential operations at each point in time. In contrast to traditional planning paradigms, these plans are not precise sequences of executable operations that the agent faithfully performs. Rather they are abstract characterizations of rough sequences of useful classes of actions. The agent uses plans to guide its identification and selection of operations suggested by perceptual and cognitive events. Thus, the agent balances purposeful, goal-directed behavior with adaptation to demands and opportunities arising from external events. The agent controls multiple tasks with multiple task-specific control plans. It coordinates multiple tasks with global control plans.

We start with an agent architecture (Hayes-Roth, 1990) comprising several subsystems. A *cognitive subsystem* interprets perceived information, performs all reasoning tasks, and plans actions. Multiple *perception subsystems* sense data from the environment and preprocess it for use by the cognitive subsystem. Multiple *action subsystems* interpret action programs sent by the cognitive subsystem and execute them by actuating effectors. Subsystems operate concurrently and asynchronously, interacting by passing data among globally accessible I/O buffers.

Within the cognitive system, reasoning operations occur in the context of a *global memory*, which contains the agent's knowledge, perceptual inputs, action ouputs, and the results of all reasoning operations. On each reasoning cycle, a bounded-time *agenda manager* uses recent *perceptual events* (inputs from the perceptual subsystem), *cognitive events* (caused by prior reasoning operations), and the current *control plan* (discussed below) to identify the most important potential reasoning operations (a subset of all possible reasoning operations). It records them on an *agenda*. A *scheduler* chooses the highest-rated operation to be executed by an *executor*, producing associated changes to the global memory.

Now we focus on the representation, use, and generation of control plans. We take examples from the Guardian system for monitoring intensive care patients (Hayes-Roth et al., 1989).

A *control plan* is a temporally organized pattern of one or more *control decisions*. Each decision describes a *class of operations* the agent is *inclined* to perform until some *goal* is achieved, along with the *importance* and *urgency* of those operations. The class of operations may be more or less specific and different operations may vary in strength of class membership. Our phrase "is inclined" signifies a weak form of intention, assuming feasibility of at least one operation within the specified class and no preferable alternatives. (See discussion below.) Goals are expressed as states of the global memory. Importance is a measure of the expected value of performing the specified operations. Urgency is a measure of how soon the specified operations must be performed to have their intended effects. All control plans are recorded in the global memory.

Figure 1 shows control plans P1-P6 for a Guardian monitoring episode spanning the time interval t1-t6. Each decision defines a class of operations in terms of relevant features. Goals are not shown, but are discussed below. The line under each decision signifies the interval during which the decision is *active*, that is the interval during which Guardian is inclined to perform the specified class of operations. The heaviness of the line signifies the combined importance and urgency, which we call *criticality*, of performing the specified classes of operations: low, moderate, high. For example, P4 is a moderately critical decision to interpret respiratory parameters throughout the monitoring episode. Note that Figure 1 shows the history of Guardian's control decisions during t1-t6. Presumably it generated the decisions incrementally during t1-t6, as discussed below. Thus, at any point in t1-t6, Guardian may be assumed to have made only those decisions shown in Figure 1 to be active at that point or earlier.

An agent performs multiple *tasks*, where a task is the application of some *method* to an *instance* of some *problem*. A *task-specific control plan* embodies the agent's decision to perform a task and its strategic approach to the task. For example, P6 is a decision to respond reactively (performing the fastest appropriate operations immediately) to the patient's high

PIP (peak inspiratory pressure). The goal (not shown in Figure 1) is to lower the PIP and correct the underlying problem. P5 is a decision to respond more systematically to the patient's low temperature. The goal is to identify and correct undesirable consequences of the low temperature. Presumably, the agent could respond reactively or systematically to either problem, but chooses to react to urgent problems, such as high PIP, and to respond more systematically when time allows.

P5 illustrates multi-decision control plans with temporal organization. It has a strategic sequence of subordinate decisions: predict changes in temperature, infer the effects of temperature on other variables, plan therapeutic actions to correct undesirable effects, and perform the planned actions. The infer and plan decisions also have subordinates (unlabeled lines in Figure 1) that more precisely control component operations during corresponding time intervals.

An agent performs multiple tasks sequentially under a sequence of control plans or concurrently under concurrent control plans. Control plans manage task interactions and sequence operations within tasks. At one extreme, if tasks are completely independent and have no particular time constraints, the agent can order them arbitrarily and perform each one in turn. On the other hand, if tasks would benefit from access to one another's intermediate results or have severe time constraints, the agent can perform them concurrently, interleaving their constituent operations so as to enable the needed exchange of intermediate results and meet time constraints. To achieve this higher-level organization, the agent makes *global control decisions* that express preferences for general classes of operations. They might specify operations that address particular problem classes or instances, meet certain resource requirements, are involved in certain kinds of reasoning methods, etc.

In Figure 1, P1, P2, and P3 are global control plans. P1, spanning t1-t6 and moderately critical, favors operations related to certain problem classes, in the order: control

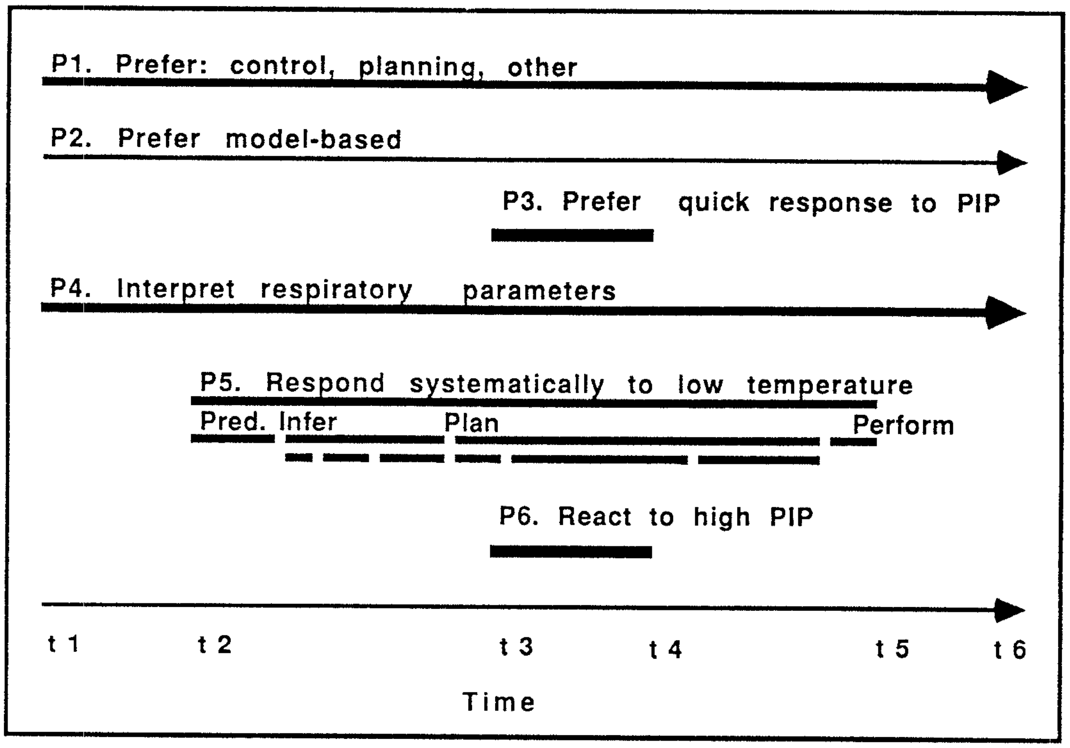

P1. Prefer: control, planning, other
P2. Prefer model-based
P3. Prefer quick response to PIP
P4. Interpret respiratory parameters
P5. Respond systematically to low temperature
Pred.
Infer
Plan
Perform
P6. React to high PIP
t 1
t 2
t 3
t 4
t 5
t 6
Time

Figure 1. Illustrative Guardian Control Plans.

planning (discussed below), therapy planning, other. P2, spanning t1-t6 and less critical, favors operations involved in model-based reasoning over other reasoning methods. P3, spanning t3-t4 and extremely critical, gives overriding preference to very fast operations related to the patient's high PIP.

Note that global control plans, like task-level control plans, refer to intended classes of operations. They do not transfer control among tasks or refer to fully-instantiated tasks. Thus, there is no infinite regression of "meta-levels."

An agent uses control plans to guide its behavior. On each cycle, the agent uses all active control plans (both task-specific and global plans), along with recent perceptual and cognitive events, to identify and rate the most important potential operations. *Ratings* reflect the degree to which operations match the specified classes of operations. *Weighting* each rating by the criticality of the control decision, an operation's *priority* is the sum of its weighted ratings. The agent executes the highest-priority operation on each cycle.

Suppose that at t3 Guardian has three potential operations. O1 would reactively diagnose the patient's high PIP. It would perfectly match P6 and P3, two extremely critical control decisions, and have a very high priority. O2 would give a model-based response to the high PIP. It would perfectly match P2, a somewhat critical decision, and partially match P6 and P3 (because it responds to the high PIP, but is neither reactive nor quick), two extremely critical decisions, and have a moderately high priority. O3 would interpret new data regarding the patient's kidney function. It would partially match P4 (because interpretats patient data, but not respiratory data), a moderately critical decision, and have a low priority. Guardian would execute the highest-priority operation, O1.

Now we can clarify our concept of an agent's "inclination" to perform planned classes of operations. If the agent has a single control plan, its inclination to perform the planned operations is equivalent to an intention. The agent will perform operations within the specified class--assuming that at least one such operation is possible and the agent is not prevented by some external force from performing it. However, if the agent has multiple active control plans specifying different classes of operations, it ordinarily will not have a potential operation that satisfies all of them. The agent will perform an operation that satisfies any one of those control plans only if no other potential operations have higher priorities based on other active plans.

Concurrent control plans also lead to task concurrency. If several plans are of comparable criticality and there exist potential operations for all of them, the agent will interleave their component operations over a sequence of reasoning cycles. If, on the other hand, one plan is much more critical than the others, the agent will consistently prefer operations that match that plan until its goal is achieved.

For example, we can infer the Guardian's behavior during t1-t6 from active control plans during successive sub-intervals. During t1-t2, P1, P2, and P4 are active. Guardian would interleave control operations (discussed below) and model-based reasoning operations to interpret respiratory parameters. It would not perform operations related to other tasks, involving other interpretation methods, or involving non-respiratory parameters. During t2-t3, P5 also is active. Along with the above operations, Guardian would interleave model-based operations in response to the patient's low temperature. In the context of this task, Guardian would generate and begin to perform its strategic sequence of subtasks. During t3-t4, the extremely critical P3 and P6 would dominate the other plans. Although the earlier plans remain active, Guardian effectively would interrupt them to perform quick reactive operations in response to the high PIP. During t4-t5, with the crisis resolved, P3 and P6 are no longer active. Guardian would resume its interrupted interpretation of respiratory parameters, model-based response to low temperature, and control reasoning. During t5-t6, with the low temperature problem resolved, only P1, P2, and P4 are active. Guardian would interpret respiratory parameters and perform control reasoning.

Generating control plans is one of the

tasks an agent performs. Thus, it performs *control planning operations* that generate or modify control decisions. As with other tasks, the agent may apply different methods to control planning, such as top-down refinement, schema instantiation, backward chaining from goals, or opportunistic planning (Johnson87). For example, Guardian presumably made decisions P1 and P2 because they are standard monitoring policies. It made decisions P5 and P3 and P6 opportunistically in response to perceptual events, the patient's low temperature and high PIP. It made P5's subordinate decisions by instantiating a general schema for systematically responding to exceptional data.

Like other reasoning operations, control planning operations are suggested by perceptual or cognitive events, rated against active control plans, and scheduled for execution. This means that the agent can modify its control plans on any reasoning cycle. Thus, for example, Guardian begins the episode in Figure 1 with three plans, P1, P2, and P4. At various times during the episode, it introduces and subsequently deactivates plans P3, P5 (and its subordinates), and P6, in response to perceived patient conditions. Guardian always behaves in accordance with whatever plans are active.

Also like other reasoning operations, control planning operations are rated against active control plans and scheduled for execution. By making certain control decisions, an agent can modulate its goal-directedness and, conversely, its responsiveness to the environment. For example, P1 is a moderately critical decision favoring control planning operations over all other problem-specific types of operations. Under P1, Guardian would be responsive to new events, making new control decisions to begin, modify, or terminate tasks promptly. In particular, it makes decisions P5 (and its subordinates) and then decisions P3 and P6 immediately upon observing the patient's low temperature and high PIP, respectively. On the other hand, Guardian's extremely critical decisions P3 and P6 dominate its behavior, making it unlikelythat Guardian will perform control planning or other reasoning operations until the high PIP problem is solved.

## 3. Final Remarks

We have proposed an approach for agents to dynamically plan and control performance of multiple interacting tasks in dynamic uncertain environments. Explicit control plans guide the agent's identification and selection of important operations. The agent integrates careful planning of longer-term strategic courses of action with reactive response to urgent situations. The agent also parameterizescontrol activities to determine how persistently it pursues established goals versus how flexibly it responds to environmental changes. Ongoing research aims to evaluate the approach both formally and empirically.

## 4. References

Garvey, A., Cornelius, C., and Hayes-Roth, B. Computational costs and benefits of control reasoning. Proceedings of the National Conference on Artificial Intelligence, 1987.

Hayes-Roth, B. A blackboard architecture for control. Artificial Intelligence, 26:251-321, 1985.

Hayes-Roth, B., Washington, R., Hewett, R., Hewett, M., and Seiver, A., Intelligent real-time monitoring and control. Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, 1989.

Hayes-Roth, B. Architectural foundations for real-time performance in intelligent agents. Journal of Real Time Systems, 1990.

Johnson, M.V., and Hayes-Roth, B. Integrating diverse reasoning methods in the BB1 blackboard control architecture. Proceedings of the National Conference on Artificial Intelligence, 1987.

## 6. Acknowledgements

This work was supported by DARPA and Boeing. Other members of the Guardian project are: R. Washington, R. Hewett, D. Ash, A. Vina, A. Collinot, I. Sim, and A. Seiver. Thanks to Ed Feigenbaum for sponsoring the work at KSL.

# ABSTRACTION AND REACTION

James Hendler

Department of Computer Science
University of Maryland
College Park, Md. 20742
hendler@cs.umd.edu

Any AI planning system, or in fact any planning agent, must base its plans on a model of the environment in which it functions. This model can correspond closely to the "real-world," reasoning about every action and object in the environment, or it can abstract away, ignoring entities, making simplifying assumptions, ignoring the time course of events, etc. In the past, a critical aspect of planning research has centered on finding *the* correct level of abstraction and developing planning primitives to enable the agent to cope with a world modeled at *that* level. In this brief paper I will attempt to demonstrate that this approach is fatally flawed — to interact with a realistic world via perception the agent must be able to reason across different levels of abstractions and to have different levels of primitive acts for these different levels. A real-time scheduling (as opposed to search) metaphor is proposed to handle the interactions between the levels. As this work is most crucial in rapidly changing dynamic environments, we describe this work in the context of the DR Real-time planning model (Hendler, 1989; Sanborn and Hendler, 1988) which is being extended to use multiple level of abstractions in planning and reaction.

Planning systems suffer from two connected problems. The first of these is inefficiency, generating complete and accurate plans in even simple domains is an exponentially hard problem (Chapman, 1986) and in more complex domains, particularly those involving other agents interacting with the planner, it may be undecidable (Sanborn & Hendler, 1988). The second limitation, which is again particularly serious in multiple agent situations, is that the model of generating followed by executing the plans is too simplistic. Change in the world occuring during the running of a plan may render portions of it either temporarily or permanently unachievable. Appropriately responding to such changes in the environment requires a reactive component not available in most planners.

As a simple example, consider a robot attempting to cross a street with a traffic flow. The robot cannot simply wait until a large enough gap occurs as (a) this may simply never happen, leaving the robot standing on the curb ad infinitum, or (b) once the gap appears and the robot starts, one of the oncoming cars might change speed, direction, etc. and thus run it over. The robot must be able to react quickly to change in the environment.

The difficulty in getting current planning systems to handle dynamic situations is caused by the reliance of most current planning techniques on some very strong underlying assumptions:

1) the planner has complete knowledge of the world relevant to its task,
2) The planner effects change only by executing primitive plan steps. This change must be discrete, and the planner must be completely aware of all effects of its actions.
3) the planner acts alone in the world; there are no outside forces.

Unfortunately, real-world planning situations rarely conform to these assumptions. Typically occuring domains may include continuous change over time, incomplete specifiability at any point in time, and change due in part to the actions of other agents. Thus, the traditional planning paradigm has been shown to be inadequate in practical situations and much current research focuses on solutions. (A good set of papers on such work can be found in the Proceedings of the DARPA Workshop on Knowledge based Planning, Austin, 1987.)

Our past research has focused on developing an architecture for managing observation and action in dynamic domains (known as "dynamic reaction" (DR), Sanborn & Hendler, 1988). This model is designed for dynamic worlds, where change is ongoing regardless of an agent's actions — strictly goal-directed methods are inadequate. Instead, the planning agent must constantly observe the world and make predictions about how events will turn out. Given these predictions, the agent coordinates its actions in order to act in harmony with ongoing events in the world. In this way, the environment dictates an agent's possible actions and longer-term goals become heuristics in determining which these actions is best pursued.

The DR model has been used to handle the interactions arising in a rapidly changing simulation domain in which objects move rapidly through the simulation under external control. The "reacting agent" must cross this environment without being impacted, but under the control of a higher level directional goal. A full description of this work can be found in (Sanborn and Hendler, 1989).

Recently we have been attempting to extend our architecture in two directions: we are trying to make the work more compatible with real robotics and real-time control systems (cf. Hendler, 1989) and we are trying to extend the system to interact gracefully with long term planning models. These two together goals taken together, however, appear to led to a contradiction. To handle real sensor data, as in robotics, we must be able to have an abstraction which closely matches the external world (this is, in fact, the same level as the model we have been using in the DR system). To interact with a more traditional, higher-level planner, however, the system needs to abstract away from the actual motions of the vehicles and etc — If the planning system is too sensitive to the short term changes occuring in the world it is unable to generate long term plans as the obvious combinatoric explosion rears its ugly head.

Clearly, to deal with this problem we must have at least two, and we currently believe more, levels of abstraction of the world. Keeping these levels consistent with one another, handling the interactions at each level in different time scales (for example the reactor may need to react in milliseconds, while the planner can take minutes), and propagating the perceptual information to the appropriate level are the critical problems we believe must be addressed to allow planers to integrate "high-" and "low-level" knowledge.

Consider the following situation: a path planning system is to prepare a route over some map — The system designs a set of points to reach and deadlines to reach them. Such a system needs not know the actual location of other objects in the world, it simply needs to know roughly how difficult different regions are to traverse. Once the plan starts executing, however, the situation changes drastically. The reactive controller needs to know what other objects are in the perceptual field, what their heading and directions are, and when they will interact with the current path. The high level planner generates plans like "GET-TO POINTA DEADLINE: +24min." The reactive controller controls moves like "PROCEED-LEFT NOW!"

In fact, the path planner itself may have no work to do during the actual running of the plan (this is a simplifying assumption used in several domain-dependent planning systems). However, if the world starts to get complicated, this assumption won't hold. Once, due to some reaction, the planner is to miss a deadline, it must replan and decide what route to take. During this replanning, however, it may not remain still — the objects which are causing it to miss the deadline may still be around!

The solution to handling such problems, appears to lie in designing a system which can be reasoning "simultaneously" about different levels of the problem. In the DR model the a "parsed" version of low-level perceptual data is supplied directly to low level agents, called "monitors" which process it (actually hierarchically) to check a particular condition (the direction of a particular car, the speed of some object, whether any new object has appeared, etc.) When a monitor discovers that some condition holds, it updates a global "state-of-the-world" model (used for providing accurate information if needed by the high level planner) with a symbolic abstraction of what it has seen (for example "Speed CAR1 40-60units"). This information is also reported to higher-level monitors, which are used to control the low level actions of the system (this coupling of monitoring and acting is described in the aforementioned Sanborn & Hendler, 1988).

The newer part of our system comes in the design of higher level reactive agents (higher-level monitors) which compute for violations of required conditions. Thus, as the lower level reactors change the direction of movement, this is reported to a higher level entity which is checking that a deadline can or cannot be reached. It too updates the state-of-the-world model, but with higher level information — the information is kept at a level of abstraction useful to the path planner (for example, what previous deadlines have been met, the current location of the object, and the projected time to reaching the destination). The planner is then able to take over, when time permits and do the appropriate replanning. Just as was the original plan, this new plan is "compiled" down to new monitoring tasks and the system continues.

The final step in getting this system to work is to use a scheduling metaphor (or, in fact, an actual scheduler) to allow the planning and reacting agents to work together as time permits. Low-level monitoring must occur frequently, but for short periods of time. How much processing is required depends on the number of objects which the reactor must take into account ("the more bullets a-coming, the more dodging is needed"). The higher level monitors and the planner itself require more processing time, but over longer intervals. Thus, as the time taken by pure reaction is reduced, the higher levels get more time. In a "safe" environment, this allows the planner to take almost complete control. Thus, in a highly reactive situation (crossing the street) the system is primarily reactive. In a relatively static world, the system becomes more like a traditional strategic planner. (See figure 1).

The interspersed scheduling of reactive and planning tasks is currently being implemented using the MARUTI hard real-time scheduling system developed by the SDAG group at the University of Md. The coupling of actual sensors to a controller based on the DR model is being pursued in conjunction with the AI group of the MITRE Corporation. Current plans include an integration of these components into a single system, and an examination of using this model for more complex planning tasks where, possibly, more levels of abstraction will be called for.

## REFERENCES

Chapman, D. Nonlinear Planning: A rigorous reconstruction, IJCAI-9.

Proceedings of DARPA workshop on Knowledge-based planning, Dec. 1987.

Hendler, J.A. "Real Time Planning" AAAI Spring Symposium on Planning and Search, 1989.

Sanborn, J. and Hendler, J. Monitoring and Reacting: Planning in dynamic domains *International Journal of AI and Engineering*, Volume 3(2), April, 1988. p. 95

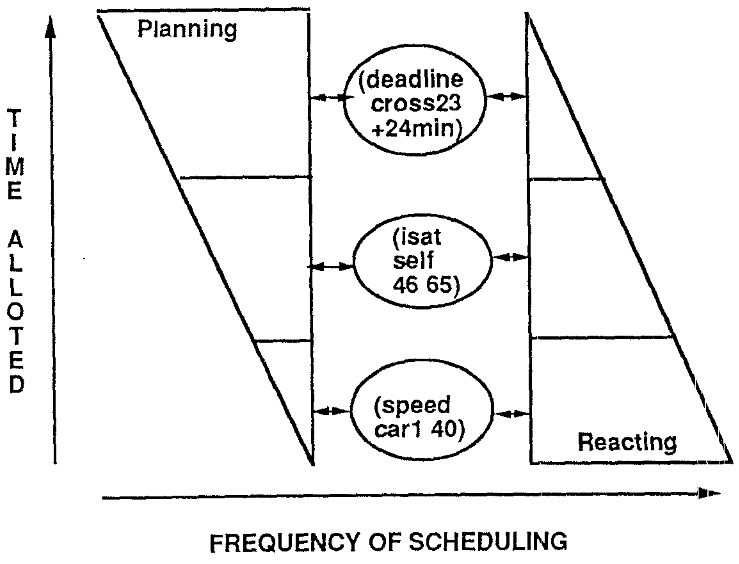


Figure 1: Planning and reacting

# Integrating Adaptation with Planning to Improve Behavior in Unpredictable Environments

**Adele E. Howe**
Experimental Knowledge Systems Laboratory
Department of Computer Science
University of Massachusetts
Amherst, MA 01003

As research in planning has progressed from static well-defined domains to dynamic unpredictable domains, planning and acting have become more closely coupled, and adaptability has become increasingly important. Because a planner cannot predict with certainty the outcome of actions in these domains, planners must adapt on-going plans in response to failures. Constructing plan knowledge bases for these domains is costly and difficult to perfect; so, a planner should adapt its own model of how to act, how far ahead to plan, and how often to assess its progress in response to changing conditions.

## Why is "good" planning not always enough?

Good planning is enough when plans can be prevented from failing. Plans do not fail when they do not commit to action in the future or when their operating environments are predictable and static. They may not fail in domains in which it is possible to enumerate contingencies. Even the best laid plans are likely to fail in domains with multiple agents, resource limitations (particularly time pressure), unpredictability, and/or complex dynamics. These domains usually cannot be characterized by tractable domain models and are fraught with subtle interactions between the agents and their environment.

Crisis management domains, such as forest fire fighting and oil spill containment, exhibit most of these characteristics. The goal of planners in these domains is to contain an environmental process that occupies an increasingly large geographic area and behaves somewhat unpredictably. The actions available to agents operating in these domains produce localized, limited effects, thus requiring the coordination of many agents over a large area and a large span of time. The Phoenix project provides just such a domain in the form of a simulation of forest fires in Yellowstone National Park (Cohen et al. 1989). The simulation and basic agent architecture supported by Phoenix constitute the basis for these explorations into adaptable planning.

## What is adaptable planning?

Because of the complexity and unpredictability of the environment, plans may not progress as intended and cannot be constructed to include contingencies for all plan-damaging events.The agent needs to adapt to changing conditions and to adapt its model of how to act through experience with its environment. Thus, we can distinguish two senses of adaptation: responsive and impressionable. A *responsive* planner responds to environmental changes by modifying plans in progress. An *impressionable* planner remembers the results of plan modifications and so adapts behaviors over extended periods of time.

The two kinds of adaptation increase flexibility and reduce brittleness in the underlying planner. The Phoenix planner is a lazy skeletal expansion planner, designed to delay commitment to precise actions as long as possible. It generates plans by searching a plan library of skeletal plans for one appropriate to the situation. These plans are expanded into a network of actions, which are represented along with their resource requirements and execution priority on an internal agenda mechanism, called the timeline. A scheduler selects actions from the timeline for execution and allocates resources (usually processing time) to them. When executed, these actions may initiate sensor or effector actions, perform problem solving actions, or search for subplans to accomplish the action's objective. Integrating responsive and impressionable adaptation with this style of planning should produce an agent able to act in a changing environment and able to exploit this experience by modifying its model of how to act.

## Responsive Adaptation

Selecting the best action in a complex domain often requires up-to-date information. The two most common approaches to this requirement are situated action and error recovery. In the situated action approach, planning and acting are tightly coupled so that decisions about how to act are made at the time of action (Agre & Chapman 1986). Rapid response to changing situations is a natural result of this tight loop. However, this approach does not naturally support reasoning about resource allocation and coordination of disparate activities or agents (Georgeff & Lansky 1987). In error recovery, situations that indicate failure of plans in progress trigger mechanisms that change those plans. Unlike the situated action approach, in error recovery, adaptation is viewed as an adjunct to the standard planning process usually requiring additional mechanisms like monitoring and replanning (Wilkins 1987).

Many complex domains are characterized by scarce resources, both physical and temporal. Each approach, situated action and error recovery, possesses capabilities desirable for these domains. Situated action offers immediate response that is smoothly integrated with normal agent actions. Error recovery offers a plan structure that facilitates coordination and control in support of resource reasoning. Responsive adaptation should offer both capabilities.

Responsive adaptation should change the intended plan by the minimum required for the agent to continue acting successfully. Plans provide the structure for controlling action coordination, undesirable plan interactions and resource use. Responsive adaptation should preserve the expectations of these constraints while still addressing the changes in the environment. It should make the changes as quickly as possible because computation time is itself a resource and because the environment may have changed before the response can be determined. Finally, because responsive adaptation is activated when exceptional conditions occur, it should provide broad coverage of possible situations; it is, in effect, the action of last resort.

An agent detects failure conditions in several ways. Actions may be unable to execute to completion, thus signalling a failure. In systems with multiple control layers (such as Brooks' subsumption architecture (Brooks 1986)), settings or commands from different layers may conflict. Potential failures can be detected by monitoring the progress of actions and plans toward their objectives. Each of these occurrences provides a context for the error along with information about its nature and perhaps its cause.

Responsive adaptation in Phoenix proceeds by using the context of the failure to search a skeletal plan library of general recovery plans. These plans make mostly simple repairs to the structure of the evolving plan. These repairs are based on structural manipulations of the plan representation and can be used in different situations. This strategy for recovery plans is similar to that of SIPE (Wilkins 88).The recovery plans vary from simply re-scheduling the failed action to aborting the plan in progress and selecting another. Other plans re-instantiate plan variables or update information from the environment. These plans are represented in the same action description language as the domain specific plans and so are interpreted by the normal planning mechanisms.

Furthermore, these plans serve a dual purpose: they are intended to salvage the failed plan and to guide improvements to the plan knowledge base.

## Impressionable Adaptation

Failures occur when the environment changes unexpectedly and detrimentally. At times, this occurs because the environment is wildly unpredictable; most often, it is because the plan was not quite right. Relying on the contents of a static plan library biases the agent toward repeating the same plans regardless of whether these actions are the best for the situation.

Impressionable adaptation changes the agent's internal model of how to act in response to experience with failure. Plan failure provides an ideal opportunity for model refinement for two reasons: plan failures tell the planner where its knowledge is inadequate or brittle, and the agent has less to lose in trying something new. Several systems have exploited the opportunity of plan failure to refine the knowledge base. For example, Kristian Hammond's CHEF system modifies plans to anticipate and avoid failures by explaining the cause of the failure (Hammond 1986). SOAR learns new rules by chunking the results of searches triggered by impasses (Laird 1987). Each relies on a domain model. For CHEF, that model is a model of relationships between actions and effects; for SOAR, it is a model of a search space of operators and their applicabilities.

When the domain model is incomplete or approximate, impressionable adaptation must rely on other characteristics of the environment. In on-going environments in which the potential cost of acting is not catastrophic, the environment can be viewed as a laboratory for discovering how best to act. Changes to the plan library can be treated as hypotheses about how to act, which are confirmed or refuted by their later execution. Successful planning reinforces known behaviors; unsuccessful planning suggests alternative behaviors. Thus, inadequacies in the domain model can be counterbalanced by experience.

Because impressionable adaptation must share computational and physical resources with responsive adaptation and other activities in support of planning, it must opportunistically gather information or use what was gathered in the service of other activities. Adaptation mechanisms must operate from limited knowledge of the domain and make use of available temporal and other resources to minimize competition with other activities of the system.

Impressionable adaptation in Phoenix is initiated in response to information about the success of a plan, usually the lack of success. When initiated, the adaptation mechanisms search for a skeletal plan that gather evidence and modify the failed plan. As in responsive adaptation, these plans rely as much as possible on information already available. The modifications to suspect plans are mostly simple changes to monitor particular conditions or actions for potential failure, and avoid or recover more easily from failure. These plans include adding choice points to prevent premature commitment to a course of action, changing the conditions of plan application, restricting action selection at choice points, and adding monitoring. The modified plans are added to the plan library causing the original plan to be either replaced or restricted in its applicability. By augmenting the plan library, the agent has the option of alternative actions when later it encounters the situations that previously led to plan failure.

## Integrating Adaptation into Phoenix

Like planning, adaptation is just one of the many problem solving activities that an agent performs. Consequently, it should be subject to the constraints of any problem solving activity. For systems that operate under time pressure, the primary constraint on problem solving is computation time. As described earlier, all problem solving activities in Phoenix are represented uniformly on the timeline. The cognitive scheduler accesses this structure in determining processing order and allocating processing time. Because responsive and impressionable adaptation are represented as problem solving actions, they can be included in existing resource reasoning and activity monitoring.

Adaptation actions are added to the timeline when failures occur. Failures are recognized through three mechanisms in Phoenix: execution errors, reflexes, and envelopes. Execution errors occur when an action cannot execute to completion because the state of the world was not right, required information was not yet available, or, for some problem solving actions, no solution exists. Reflexes are a low level control mechanism that compensate for time delays in planner response to keep the agent out of catastrophe. When they are triggered in response to dangerous environmental conditions, they program effectors to remove or at least reduce the danger. Envelopes predict impending failure (Hart, Cohen & Anderson 1990, Powell & Cohen 1990). They perform sophisticated monitoring of the plan's progress in the world, integrating the efforts of many agents, to determine whether the plan can complete within its environmental and resource limitations. If a plan will be unable to complete successfully under the present conditions, the performance envelope is violated and an impending failure is signalled. These mechanisms signal failures by adding adaptation actions to the timeline. These actions include information readily available about the circumstances of the failure.

Adaptation actions on the timeline are treated as a type of planning action. They rely on the same planning method--lazy skeletal expansion--and when executed, search a taxonomy of skeletal adaptation plans in the plan library to find the appropriate response to the situation. As a type of planning action, adaptation may access and use the same methods as other planning methods and can be smoothly integrated into the planning process. As a timeline action, adaptation actions have access to the same memory structures and are subject to the same resource management techniques as are other timeline actions.

## Improving Behavior through Adaptation

Adaptation compensates for inadequacies in planning. When the environment changes so that on-going plans will fail, responsive adaptation searches the plan library for an appropriate response and changes the on-going plan to allow the agent to continue from the failure. When the model of how to act in the environment fails, impressionable adaptation changes the plan library to avoid or anticipate the problem in the future. Thus, impressionable adaptation reflects the experience gained from responsive adaptation into the plan library to improve planning behavior. Figure 1 shows how the two types of adaptation influence the plan library and the timeline. Responsive adaptation modifies the timeline based on plans found in the plan library. Impressionable adaptation modifies the plan library based on information gleaned from the timeline.

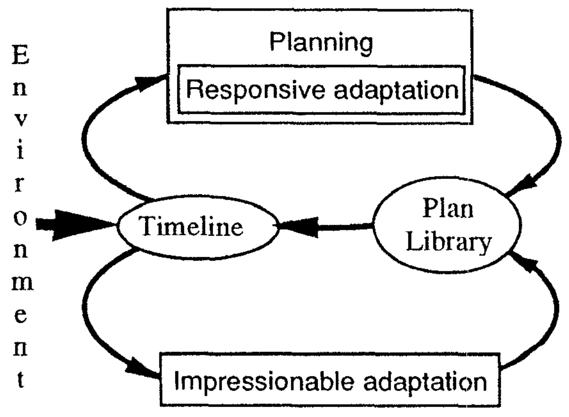


**Figure 1**: Responsive and impressionable adaptation interact through the timeline and the plan library.

Integrating responsive and impressionable adaptation into the Phoenix system forms the core of my dissertation, as proposed in (Howe 1989). My hypothesis is that when the agent adapts its own behavior to its environment, its behavior will be better suited to that environment, and the agent's internal model will more accurately reflect the constraints of that environment. Consequently, the goal of the project is to understand the relationship between adaptation and the characteristics of the environment. The internal structures resulting from adaptation should evidence the impact of environment; for example, plans should reflect the variability and rate of change of the environment in the number and nature of the decision points in them. The benefits of additional plan knowledge should more than compensate for the computational overhead; in other words, adaptation must have demonstrable utility.

## Acknowledgments

This research was supported ONR University Research Initiative grant N000014-86-K-1764 and by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Air Force Office of Scientific Research under Contract No. F49620-89-C-00113. The United States Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon.

## References

Philip E. Agre and David Chapman. 1986. Pengi: An Implementation of a Theory of Activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence*. Seattle, WA.

Rodney A. Brooks. 1986. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, RA-2(1).

Paul R. Cohen, Michael Greenberg, David M. Hart, and Adele E. Howe. 1989. Trial by Fire: Understanding the Design Requirements for Agents in Complex Environments. *AI Magazine*, 10(3).

Michael P. Georgeff and Amy L. Lansky. 1986. Reactive Reasoning and Planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence*. Seattle, WA.

Kristian J. Hammond. 1986. Learning to Anticipate and Avoid Planning Problems through the Explanation of Failure. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, PA.

David M. Hart, Paul R. Cohen and Scott D. Anderson. 1990. Envelopes as a Vehicle for Improving Plan Efficiency. submitted to the Eighth National Conference on Artificial Intelligence.

Adele E. Howe. 1989. Adapting Planning to Complex Environments. PhD Dissertation Proposal, Dept. of Computer and Information Science, University of Massachusetts.

Gerald M. Powell and Paul R. Cohen.1990. Operational Planning and Monitoring with Envelopes, in *Proceedings of the IEEE Fifth AI Systems in Government Conference*, Wash., DC.

John E. Laird, Allen Newell, and Paul S. Rosenbloom. 1987. SOAR: An Architecture for General Intelligence. *Artificial Intelligence Journal*, 33:1-64.

David E. Wilkins. 1988. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan-Kaufmann Publishers, Palo Alto, CA.

# Partial Planning with Incomplete Information

Jane Yung-jen Hsu
Logic Group, Computer Science Department
Stanford University
Stanford, CA 94305

## Abstract

This paper presents a new framework for planning with incomplete information for agents acting in complex environments. By modeling the behavior of an agent as a function from its perceptual histories into actions, our approach enables the system to maintain a partial plan, which can be updated incrementally when new information or time becomes available. Our planner uses an anytime algorithm that always choose the *best* actions based on the knowledge and computational resources utilized so far. The framework of partial planning provides a nice integration of reactivity and reasoning with declarative knowledge.

## Introduction

Traditional planning systems assume the availability of complete knowledge at planning time, so that once a plan is constructed by the planner, it is guaranteed to be carried out successfully by the plan executor. Unfortunatly, in complex environments, an agent must often deal with incomplete and changing information. In the face of such uncertainty, a traditional planner will either fail to generate any plan at all, or fail to achieve its desired goal(s) by executing the generated plan.

There are several sources of incompleteness. First of all, a planner may not know the exact initial situation due to its sensory limitations and certain uncontrollable randomness in its environment. For example, suppose that a robot is given a goal to find a piece of gold somewhere in a maze, and to find its way out with the gold. Rather than starting from a prespecified cell in the maze, the robot is carried by a helicopter into the maze at random. Let's further assume that the robot is able to perceive whether the gold is in the current cell, but it cannot identify in which cell it is positioned exactly except when it happens to be at the exit. Consequently, the initial state is not completely known to the robot. The second sort of incompleteness is a result of a planner's imperfect knowledge about the preconditions and effects of its operators. For instance, the robot may expect itself to be holding the piece of gold after performing a pickup action, while its arm may fail to pick up the gold because the weight exceeds its capacity, or some other agent may have taken the gold away in the mean time.

It is desirable for an agent to be able to function reasonably well in the world despite the lack of a complete (and correct) plan at the beginning. One possible solution is to *interleave planning with plan execution*. In an embedded planning system, which continuously interacts with its environment, additional information can often be acquired during the course of its actions. Such information can then be used to guide the system's future actions, especially in unpredicted situations. On the other hand, a planning system may choose to trade increased planning time for improved performance in plan execution. It should be stressed that the amount of available information puts a realistic bound on how much a plan can be optimized. In other words, when the information is not sufficient to warrant a specific optimal plan for the given situation, no amount of planning will help find the optimal solution. Furthermore, a planner may not always have enough computational power to figure out the right actions to perform under resource constraints. There has been active research on strategies for interleaving planning and acting [Bratman *et al.*, 1988, Georgeff and Ingrand, 1989], although it remains largely an open problem.

At the other end of the spectrum of approaches to planning in dynamic environments is the various work under the rubic of *reactive planning* [Agre and Chapman, 1987, Kaelbling, 1987, Schoppers,1987]. Instead of assuming the intended effects of an action to always hold, the world can be in any of all the possible states after action execution. Constant feedbacks from sensory inputs are necessary, and planning becomes the task of determining an action to do given the current situation. One important research issue is the integration of goal-directed reasoning with reactivity [Drummond, 1989b, Nilsson, 1989].

To address the problem of incomplete information, we proposes *partial planning* as a new way to control the behavior of an intelligent agent. The basic idea underlying our approach is to plan as much as is permitted by the planner's current resource without overcommiting one's actions. Our planner maintains a partial plan at all times, and refines the plan when new resource, i.e. *information* or *time*, becomes available. Our approach allows the system to act at any time according to its partial plan, although the quality of the actions performed depends strongly on the amount of resource that has been utilized by the planner.

In the following sections, we first describe a plan representation formalism suitable for planning in uncertain and changing environments. We then analyze the important role of information (or knowledge) in planning, and show how new information is used to prune the space of possible plans. A simple strategy for interleaving planning and execution used is given. The process of incremental specialization of partial plans will be illustrated by an example in Section 4. Section 5 sketches an anytime algorithm [Dean and Boddy, 1988] for partial planning.

## Plan Representation

In classical planning, a planner typically takes as input the initial state(s) of the world as well as the goal state(s), and outputs a plan that will achieve the goal(s) when executed from the initial state. Two types of plan representation that have been most widely used in earlier planners are: state-space and action-ordering [Drummond, 1989a]. As will be discussed below, such plan representations are not adequate for dealing with incomplete information. We will introduce a more general plan representation formalism with functional semantics that is suitable for the job.

**State-space plan representation** A state is a snapshot of the world, and an action is a transformation of the world from one state into another. A state-space plan is a path from (one of) the initial state(s) into (one of) the goal state(s) in the space of all possible world states. The order of actions in a standard state-space plan is completely specified, and the states (including intermediate ones) are explicitly represented. A partial plan (or a partially constructed plan) can be defined to be an incompletely specified path, i.e. the prefix or suffix of a complete plan.

The problem with representing plans this way is that the causes/effects of the operators are assumed to be completely known, and there is no provisions for handling unexpected situations. It is also difficult to describe behavioral properties such as constraints among actions in a plan without prematurely committing to a fixed sequence.

**Action-ordering plan representation** An action-ordering representation has no explicit notion of states. A plan is a set of operators with constraints on the order of variables. This representation allows the use of *least-commitment* strategy in plan construction, i.e. actions in the plan are left unordered as long as possible [Sacerdoti, 1975]. A partial plan, or a partially-ordered plan, is one in which the order over the (possibly incomplete) set of actions is not fully specified. Partially-ordered plans allow more flexibility, but it still assumes complete knowledge about the effects of actions, and thus cannot handle unpredicted situations. In particular, the reasoning necessary for deciding the ordering of actions in the plan depends on the planner's knowing the preconditions and postconditions of all the actions. If the information is incomplete, then the resulting ordering may be incorrect.

**Functional plan representation** Let $S$ be the set of states and $A$ be the set of actions, then each action $a$ in $A$ can be described by a function from states into states, i.e. $a : S \rightarrow S$. A complete plan $\alpha$ consisting of a *sequence* of actions $a_1, \ldots a_n$ from $A$ can thus be represented by a function $\alpha : S \rightarrow S$ defined as follows (where the operator $\circ$ stands for function composition):

$$\alpha = a_1 \circ a_2 \circ \cdots \circ a_n.$$

A partial state-space plan is a function $\beta : S \rightarrow S$ such that $\alpha = \beta \circ \alpha'$ or $\alpha = \alpha' \circ \beta$ for some sequence of actions $\alpha'$. A partially-ordered plan is a function $\gamma : S \rightarrow S$ such that $\alpha = \mathcal{F}(\gamma, \alpha')$ for some functional $\mathcal{F}$.

In order to relax the assumption of complete knowledge, reactive or situated planners consider a plan to be a set of mappings from situations into actions. A reactive plan can be described by a function,

$$\alpha : S \rightarrow A.$$

The sequence of actions is decomposed into individual actions so that the reliance on the effects of actions to hold is eliminated. However, such approach based its action selection on the *current* state only, therefore it is not powerful enough to deal with cases that call for a finite number of repeated actions without looping. One can address this problem by having some metalevel mechanism that reasons about the actions (e.g. counters, random noise etc.) and get the system out of trouble whenever necessary.

Our approach generalizes the view taken in reactive planning in two ways: We uses the notion of *percepts* (denoted by $P$), i.e. a possibly partial description of the world, to account for the sensory limitations of an agent. We also includes the history of percepts by modeling the behavior (plan) of an agent as a function from its perceptual histories into actions, i.e.

$$\alpha : P^* \rightarrow A.$$

Consequently, we define a *partial plan* to be a partial specification of this function, i.e. some mappings are not (uniquely) defined. Using perceptual histories allows us to represent state sets (e.g. for uncertainty regarding initial state) as well as internal states of the agent. This formalism is more expressive than the traditional plan representations in that we can describe the correlations between states and actions, sequences or ordering of actions, as well as constraints among actions in different states [Genesereth and Hsu, 1989].In addition, it offers the ability to handle incomplete information since one can start with a partial specification of a plan and refine the plan incrementally by simply adding information to it. We will explain this process in more details in the following section.

## Information-specific planning

Our planner starts out by assuming the most general partial plan, in which every action is applicable at every situation. Intuitively, each partial plan corresponds to a set of possible complete plans. When the specification of the plan entails a unique action for any given perceptual history, the plan becomes complete. At any point of time, the planner tries to find the maximally specific plan based on the current information and computational resources. This idea is analogous to least-commitment strategy in non-linear planning. The choice of action for any given perceptual history is not made prematurely so that new information can be utilized to make better decisions. As a result, the efficiency of both planning and execution can be improved.

Planning is thus a process of incremental specialization of the partial plan at hand. For example, if the planner receives information about the goal, it can assert that the proper action for any situation satisfying the goal is to stop (succeed). It can also assert that an action should be performed at a given situation if the result of executing that action satisfies the goal condition. Similarly, the planner can declare an action undesirable if it prevents the goal from being achieved. In any case, the partial plan becomes increasingly specific as information accumulates. The incremental specialization is performed using a technique based on partial evaluation.

Given incomplete plan specifications, the behavior of a system is underconstrained such that more than one action may be possible in some situations. The amount of information available puts a lower bound on the number of possible plans consistent with the partial specification. When the information is incomplete, more planning won't help. Therefore, one only needs to (or *can*) plan ahead as much as what one knows. On the other hand, it has been shown that there is no benefit to delay planning if complete information is available and time is not critical [Genesereth, 1989].

However, under resource constraints, it is not always possible to compute all such mappings in a partial plan. Even if it's computationally feasible, it may not be economical to do so, since some of the situations will occur rarely or none at all. The latter problem is due to the fact that the planner does not take any knowledge about state transitions into account. If we assume the world is predictable most of the time, then the planner can use a model of the world, i.e. a state transition funtion that maps the perceptual history and an action into a new percept, to guide the computation.

Let's define the size of the search space for our planner to be the number of action sequences that can be chosen based on the current computation. The lower bound on the search space is the the number of possible complete plans corresponding to the current partial plan. In general, given complete information, the size of the search space is inverse proportional to the planning time spent as shown by curve $I_c$ in Figure 1.

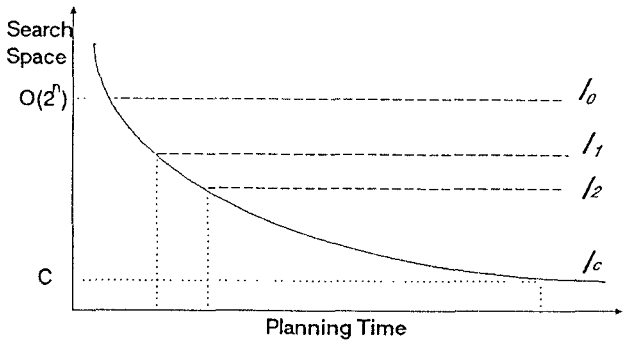


Figure 1: Effects of information and planning time.

The above diagram also shows the lower bounds of the search space when there is no information ($I_0$) or partially specified ($I_1$ and $I_2$). The more information is available, the lower the bound on the search space becomes. For example, in Figure 1,

$$I_c > I_2 > I_1 I_0.$$

**Interleaving planning and execution** It should be noted that the boundary between planning and plan execution is quite artificial. Interleaving planning and execution is beneficial, if not necessary, when information is incomplete. For example, results from action execution can sometimes help improve future steps in the plan, i.e. the planner acquires more information, thus effectively lowers the bound of the search space.

The strategy used by our planner is to spend as much time planning until it reaches the lower bound prescribed by the current information, or until it has to act. (See Section 5 for more details.) Our approach makes explicit the roles of information and time, which are the two most valuable resources in planning. So far, we have not consider the optimality of the resulting plans. Given a goal, all the plans that can lead to the goal state(s) are considered to be equally good.

## Bay-Area Transit Problem

Let's look at an example of planning with incomplete information. Suppose we are given a map of the highways in the Bay Area. To simplify the presentation, we assume that all roads are either vertical or horizontal, and two roads always intersect at the right angle. The set of possible percepts are the names of the intersection of any two roads, such as P0, P1, ..., P10 shown in Figure 2. There are five actions available to the agent: N (move-to-north), S (move-to-south), E (move-to-east), W (move-to-west), and Stop. We further assume that any two adjacent nodes can be connected by a bus or not, and the agent needs to take a bus from one node to another.

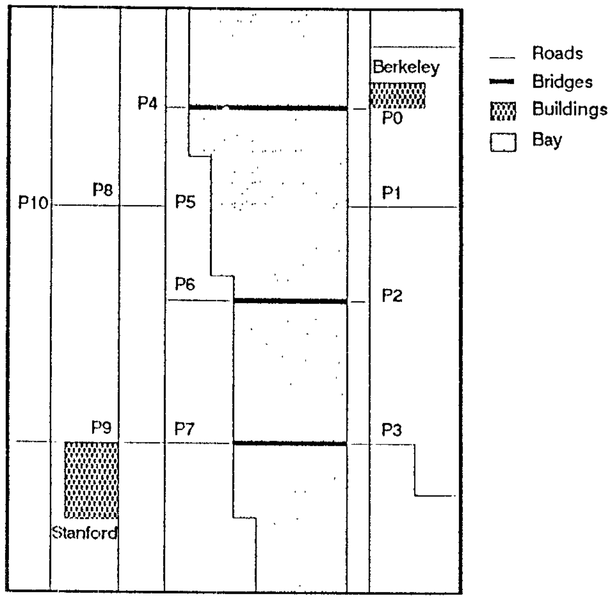


Figure 2: A map of the Bay Area.

The incompleteness of this problem stems from the facts that the agent does not have a map with the bus routes to begin with, there may be traffic jams blocking the road, and roads (or bridges) may be closed temporarily etc.

In general, if there are $n$ percepts and $m$ actions, the initial search space for the agent will be:

$$m^{1+n+n^2+\cdots+n^k+\cdots}.$$

Our task for the agent is to get from Berkeley to Stanford. Even without any specific information other than the map, the agent may end up in the right state after a tremendous amount of search since there are finite number of nodes and connections.

If the agent tries to come up with *a* plan in the absence of complete information, it will often find itself failing to achieve the goal, and having to revise it's plan by backtraking and replanning. This process is very inefficient. Instead, the agent may try to acquire more information, e.g. asking for advice, and plan as much as what the information entails. For example, some other agent can give world specifications or procedural hints like to following:

- World specifications: e.g.
  There are three bridges for crossing the Bay.
  Berkeley is located at P0.
  Stanford is located at P9.
- Elimination of specific actions: e.g.
  Do not take the bus from P4 to P5.
- Prescription of specific actions: e.g.
  Move south from p1.
- Constraints on actions: e.g.
  Cross the Bay exactly once.
- Preferences among possible alternatives: e.g.
  Prefer moving south from P5 than moving west.

After giving the above information to the agent, the search space has been reduced significantly. Most of the statements above rely on the agent's current percept only except constraints such as "Cross the Bay exactly once". It is necessary to refer to the perceptual history in order to specify this sort of information. Although statements expressing preferences among actions does not reduce the search space, it affects the order in which the search is carries out by the planner.

## Anytime Planning

The constraints on the amount of knowledge and computational resources currently available prompted us to come up with a practical way of implementing the ideas described in the previous sections. We designed an anytime algorithm that utilizes a data structure for caching precomputed results in order to react in real-time situations.

Our system executes a simple *perceive-select-act* loop. It observes its environment to decide the current percept, selects an action based on the perceptual history, and then acts on the action. Let's assume the observation and action execution are performed by the agent's sensory and effectory component. The bulk of what the system does at execution time is to decide which action to perform. We can imagine the planner constantly tries to prove correlations among its perceptual histories and actions, and the results are stored in a huge lookup table[1]. Each perceptual history has a corresponding entry in the table, and its value contains the list of actions that are applicable, i.e. those that have not been

[1]The actual implementation is done by a tree structure for correlations that have been explicitly referenced, but we won't go into details in this paper.

pruned. The actions are ordered in a way that is consistent with the information regarding preferences in the partial plan.

The planning algorithm can be roughly describe as follows:

```
LOOP forever
   Observe
   Update perceptual history
   Look up entry in the table
   IF more than one actions found
   THEN Plan (reaction-time-bound)
        Choose an action A1, s.t.
        no other actions are preferred to A1.
   Act
END LOOP
```

When a unique action is not prescribed, the next action is chosen at random from the set of applicable actions. As such, the system is able to act on demand at any time, and its performance should improve over time. This approach will work in any domain in which the effects of actions does not prevent goals from being achieved if any solution actually exists. Under this condition, our planning algorithm is guaranteed to find a solution provided one exists, although it may not find the optimal solution when solutions are considered to be different in terms of their performance.

## Acknowledgements

Thanks go to Mike Genesereth for motivating the work on partial programs. Discussions with John Woodfill, Nils Nilsson, and Charlie Koo have influenced the development of the ideas presented in this paper.

## References

[Agre and Chapman, 1987] Philip E. Agre and David Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence. Seattle, WA*, pages 268–272, July 1987.

[Bratman *et al.*, 1988] Michael E. Bratman, David J. Israel, and Martha E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4(4):349–355, 1988.

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

[Drummond, 1989a] Mark Drummond. Ai planning: A tutorial and review. Technical Report AIAI-TR-30, University of Edinburgh, January 1989.

[Drummond, 1989b] Mark Drummond. Situated control rules. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, May 1989.

[Genesereth, 1989] Michael R. Genesereth. A comparative analysis of some simple architectures for intelligent agents. Technical Report Logic-89-1, Stanford University, March 1989.

[Genesereth and Hsu, 1989] Michael R. Genesereth and Jane Yung-jen Hsu. *Partial programs.* Technical Report Logic-89-20, Stanford University, 1989.

[Georgeff and Ingrand, 1989] Michael P. Georgeff and Francois Felix Ingrand. Decision-making in an embedded reasoning system. In *Proceedings of the Eleventh IJCAI. Detroit, Michigan*, pages 972–978, August 1989.

[Hsu and Genesereth, 1989] Jane Yung-jen Hsu and Michael R. Genesereth. Knowledge compilation for informable agents. In: *Proceedings of the AAAI Spring Symposium Series on AI and Limited Rationality*, March 1989.

[Kaelbling, 1987] Leslie Pack Kaelbling. An architecture for intelligent reactive systems. In M. P. Georgeff and A. L. Lansky, editors, *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, pages 395–410. Morgan Kaufmann Publishers, Los Altos, CA, 1987.

[Nilsson, 1989] Nils J. Nilsson. Action networks. In *Proceedings of the Rochester Planning Workshop: From Formal Systems to Practical Systems.* Tenenberg, J. et al. (eds.), University of Rochester, 1989.

[Sacerdoti, 1975] Earl D. Sacerdoti. The Non-linear Nature of Plans. In *Proceedings of IJCAI-75*, pages 206–214, 1975.

[Schoppers, 1987] Marcel J. Schoppers. Universal plans for reactive robots in unpredictable domains. In *Proceedings of the Tenth IJCAI. Milan, Italy*, pages 852–859, 1987.

# A Framework for Replanning in Hierarchical Nonlinear Planning*

**Subbarao Kambhampati**

Center for Design Research
Stanford University
Stanford CA 94305
email: *rao@sunrise.stanford.edu*

## 1. Introduction

An important characteristic of planning in unpredictable and changing and environments is that the planner may often be called upon to modify its plans in response to unexpected changes in the environment. Such modification, called *replanning*, is necessitated during plan execution when the planner's expectations of the world are being violated either due to the actions of some hostile agent or due to the occurrence of some random unexpected events.[1] To manage replanning efficiently, the planner requires the ability to detect when a modification is needed to its plan and to flexibly and efficiently carry out the modification. To facilitate this, the planner needs access to an appropriate representation of the internal causal dependency structure of the plan. The efficiency considerations demand that the modification be controlled in such a way as to minimally change the existing plan, preserving as many of its applicable parts as possible.

Much of the previous work on replanning for hierarchical planners did not focus on conservative replanning capability. In this paper, we will present a domain independent framework for replanning in *hierarchical nonlinear planning*, the dominant method of abstraction and least-commitment in classical planning [ChM84]. Our framework uses a formal hierarchical representation of plan rationale to guide and control replanning. In contrast to previous approaches to replanning, our approach explicitly focuses on conservatism of plan modification by attempting to reuse all the applicable parts of the existing plan during replanning. We shall argue that these characteristics make it particularly suitable for integration into a general architecture for reactive planning.

The replanning approach we present here is a direct product of our research on PRIAR, a framework for flexible reuse and modification of plans [Kam89] [KaH89a] [KaH89b], and it has been implemented in the PRIAR system. In PRIAR, the internal dependencies of a plan are represented explicitly in the form of its *validation structure* (see below). The validation structure is then used to guide and control the modification and reuse of that plan in a new problem situation. Our main claim in this paper is that PRIAR's validation structure based plan modification approach provides an efficient and conservative framework for replanning in hierarchical planning. In particular, we develop the notion of validation-states for monitoring the execution of a plan, and to check if the current plan itself can be restarted in the event of execution-time failures due to unexpected events. If the current plan cannot be restarted, it is flexibly reused to achieve the original goals from the current world state.

We begin with a brief overview of PRIAR's validation structure based plan modification framework. Then, we explain how PRIAR monitors the execution of its plans, and how it carries out replanning efficiently. Next we discuss the role of this type of replanning capability in a reactive planning architecture, and finally discuss the relationship to previous research.

## 2. Overview of PRIAR Plan Modification Framework

### 2.1. Validation Structure

In PRIAR framework, the building blocks of the stored plan dependency structure are validations. A validation is a 4-tuple $\langle E, n_s, C, n_d \rangle$ where the effect $E$ of the task $n_s$ in the hierarchical task network (HTN) is used to satisfy (support) the condition $C$ of task $n_d$. For any plan synthesized by a hierarchical planner, there are a finite set of validations, corresponding to the *protection intervals* [ChM84] that are maintained during planning; we denote this set by V. The individual validations are classified based on the type of the conditions they support.

The uniqueness of PRIAR framework is in the way the plan validations are stored on the HTN to guide its subsequent modification. Each task $n$ in the HTN is annotated with the set of validations that are supplied by, consumed by, or necessarily preserved by the tasks belonging to the

*The support of the Defense Advanced Research Projects Agency and the U.S. Army Engineer Topographic Laboratories under contract DACA76-88-C-0008, and that of Office of Naval Research under contract N00014-88-K-0620 are gratefully acknowledged.

[1] Note that replanning in this sense does not cover the execution time failures that arise due to the incorrectness and incompleteness of the planner's domain knowledge [Kam89].

sub-reduction (hierarchical wedge) rooted at $n$. We call these the external effect conditions (*e-conditions*), external preconditions (*e-preconditions*) and persistence conditions (*p-conditions*) respectively of task $n$. These annotations are computed efficiently for each node in the HTN in a bottom-up, breadth-first fashion at the planning time for every plan that is generated by the planner. In [Kam89], we provide a $O(N^2)$ algorithm (where $N$ is the length of the plan) for doing this.

The annotated validation structure effectively provides a hierarchical explanation of correctness of the plan with respect to the planner's knowledge of the domain[2]. In PRIAR, it is used (*i*) to locate the parts of the plan that would have to be modified, (*ii*) to suggest appropriate modification actions, (*iii*) to control the modification process such that it changes the existing plan minimally to make it work in the new situation, and also (*iv*) to assist in plan mapping and retrieval.

## 2.2. Plan Modification via Annotation-Verification

Given a plan to be reused to fit the constraints of a new problem situation, PRIAR first maps the plan into the new problem situation. This process, known as *interpretation*, marks the differences between the plan and the problem situation. These differences in turn are seen to produce *inconsistencies* in the plan validation structure (such as missing, failing, or redundant validations). In PRIAR framework, a plan is modified in response to inconsistencies in its validation structure. PRIAR uses a process called *annotation-verification* to suggest appropriate modification to the plan for removing those inconsistencies from the validation structure of the plan. These domain independent modifications depend on the type of the inconsistency. They include removal of redundant parts of the plan, exploitation of any serendipitous effects of the changed situation to shorten the plan, and addition of high level refit-tasks to re-establish any failing validations. At the end of the annotation-verification, which is a polynomial time process [Kam89], PRIAR will have a *partially reduced* plan with a *consistent* validation structure. Next, PRIAR's hierarchical nonlinear planner accepts this partially reduced plan and produces a completely reduced HTN. The planner uses a conservative heuristic search control strategy called *task kernel-based ordering* (see [KaH89a]) to control this process of refitting, so as to localize the modification to the plan and preserve as many of its applicable portions as possible.

## 3. Replanning in PRIAR

A general replanning framework requires strategies for (*i*) monitoring the world to detect unexpected events that may cause problems to the plan executability and (*ii*) for modifying the plan conservatively to make it work in the changed situation. PRIAR's approach is to characterize the ramifications of the unexpected events on the executability of the plan in terms of the inconsistencies they cause in the validation structure of the plan, and to use its plan modification strategies to efficiently repair those inconsistencies in the presence of the planner. To detect the inconsistencies in the validation structure, PRIAR uses the notion of *validation-states*. To correct the inconsistencies, it uses the annotation verification and refitting processes.

### 3.1. Execution Monitoring

In PRIAR framework, a plan is considered executable as long as there are no inconsistencies in its validation structure. To monitor problematic events during execution, PRIAR keeps track of the set of validations that should hold before and after the execution of each primitive task in the plan. These structures, called *validation-states*, formally characterize the conditions whose preservation should be monitored, and thereby help the planner recognize the need for modifying its plan during execution. Using the task annotations (introduced above), PRIAR defines the notion of a *validation-state* preceding and following each primitive executable action in the plan as follows:

*Preceding Validation-state* $A^p(n) =$
$e\text{-}preconditions(n) \cup p\text{-}conditions(n)$

*Succeeding Validation-state* $A^s(n) =$
$e\text{-}conditions(n) \cup p\text{-}conditions(n)$

The validation-states thus specify the set of validations that should hold at each point during the plan execution so that the rest of the plan will have a consistent validation structure (thereby guaranteeing its successful execution)[3]. Only those unexpected events that affect the validations in the current validation-state cause problems to the plan executability necessitating replanning; all the others can be safely ignored. This gives PRIAR the following simple model for execution monitoring.

Each primitive task $n$ in the HTN is considered an execution point for the plan. Normally, the primitive tasks can be executed in any way consistent with the partial ordering relations among them. Before executing a primitive task $n$ of the plan, a check is made to see if any of the validations in its preceding validation-state $A^p(n)$ are failing in the current world state $W$. Formally, we say $n$ is executable if

---

[2] For a complete and formal presentation of validation structure, see [Kam89].

[3] An important point to bear in mind here is that even if the execution monitoring finds no discrepancies between the validation-states and the world states, the plan may still fail to achieve its intended outcomes because of incorrectness and incompleteness of the planner's domain model. The replanning problem addressed here only guarantees detection and correction of failures that lie in the deductive closure of the planner's domain knowledge.

$\forall v: \langle E, n_s, C, n_d \rangle \in A^p(n),\ W \nvdash \neg E$

where $W \vdash E$ is true if $E$ can be deductively inferred from $W$ and the domain causal theory. Thus, by "executable", here we mean that not only the current primitive action, but the rest of the plan can be successfully executed. If any validation belonging to $A^p(n)$ does not satisfy the above condition, it implies that there are inconsistencies in the plan validation structure; the plan cannot be successfully executed from this point.

In practice, this check could either be done by making explicit monitors for keeping track of the truth values of the validations or by allowing an external module to input an arbitrary predicate as the effect of an unexpected event. In our current implementation, we have the user specify an arbitrary description $P$ as the partial effect of some unexpected event, and the system uses that information to compute the resultant world state $W$ and checks to see if any of the validations in the current validation-state fail in $W$[4].

## 3.2. Replanning

### 3.2.1. Restartability

We have shown above that if any of the validations of $A^p(n)$ do not hold in $W$, the plan cannot be successfully executed from $n$. However it may be possible to restart the plan from some other execution point. The advantage of checking for such a possibility is that it gives the planner a chance to reduce its response time by eliminating the necessity for replanning, there by making it very reactive. However, it should be noted that given sufficiently general types of unexpected events, a plan is typically not restartable, unless happens to be a "*spanning tree plan*" [Nil89] (see below).

Validation-states can help us decide efficiently whether or not a plan can be restarted from the current world state to achieve all its intended goals. When the execution monitoring detects discrepancies between the current validation-state and the current state of the world, PRIAR scans the primitive tasks in the HTN, starting backwards from the goal node, to see if there exists a task $n_j$ such that all the validations in its preceding validation-state $A^p(n_j)$ are satisfied in the current world state $W$. If such an $n_j$ is found, i.e.,

$$\forall v: \langle E, n_s, C, n_d \rangle \in A^p(n_j),\ \mathrm{W} \vdash E$$

then the execution can be restarted from $n_j$ onwards. If no such validation-state exists, the plan cannot be restarted and we have to resort to a more general replanning strategy involving the modification of the plan.

Since only a finite set V of validations constitute the plan validation structure, we can ensure that each validation is checked at most once by keeping track of the validations that have already been checked. This gives a scanning algorithm that runs in time linear in the size of V.

### 3.2.2. Plan Modification during Replanning

When the current plan is not restartable from any of the execution points, then it has to be modified to fit it to the current situation. Efficiency considerations demand that as much of this plan be reused as possible to achieve the goals from the current situation. Depending upon the ramifications of the unexpected event on the plan, both the parts of the plan that are yet to be executed, and those that are already executed might be reusable in the new situation. (Suppose an unexpected event undoes some goals that are achieved by the parts of the plan already executed. Then it is possible that some of those goals could be reachieved efficiently by re-executing some of the already executed actions, instead of trying to plan for those goals again.) Thus, if a replanning strategy only attempts to repair the unexecuted parts of the plan, it might lead to unnecessary repetition of some planning activity. What we need here is the ability to flexibly reuse all the applicable portions of the current plan.

PRIAR's approach to this general replanning problem is to consider it as a problem of flexible plan reuse. In particular, if $R^o$ is the HTN being executed, $G^o$ the set of original goals of the plan, and $W$ the current state of the world (which necessitated the replanning), then, PRIAR converts the replanning problem into the following plan reuse problem:

> *Find a plan to achieve* $G^o$ from the current world state $W$, reusing the plan $R^o$ and retaining as many of its applicable parts as possible.

That is, instead of trying to repair the validation failures that are present in the validation-state preceding the current execution point, PRIAR tries to reuse the entire original plan to achieve the original goals starting from the current situation. The motivation is that a straight forward algorithm to repair validation failures preceding the current execution point would not be able to reuse already executed parts of the original plan.

During the reuse of $R^o$, the interpretation procedure marks the differences between $A^s(n_I)$ of $R^o$ and $W$.[5] These differences will be causing inconsistencies in the validation structure of $R^o$. The interpreted plan is then sent to the annotation-verification procedure, which carries

[4] Computation of the current world state after an unexpected event involves removing any descriptions that are contradicted by $P$ from the expected world state $W'$, and adding $P$ and any deductive effects that follow from $P$ to the result. PRIAR represents the causal theory of the domain in the form of "state rules" and "causal rules" (as described in [Wil88]). The causal theory is then used to infer the deductive effects of the unexpected event.

[5] Since this is a replanning situation, the interpretation mapping will be an identity mapping.

out various modifications to remove the inconsistencies in the validation structure. The modification actions depend on the type of the validation failure or inconsistency. They make use of the node annotations to remove redundant parts of the old plan or add high level refit-tasks to achieve missing or failing validations. In addition to repairing inconsistencies, the annotation-verification procedure will also take advantage of any serendipitous effects at the execution time to shorten the plan.(For complete details of these modification actions see [Kam89].)

The partially reduced HTN that is the result of the annotation-verified plan is then sent to the PRIAR refitting process, where any refit-tasks suggested by the annotation-verification process are reduced to produce a complete plan. This refitting process is controlled by the task kernel-based ordering which essentially involves reducing each refit-task by a task reduction schema that causes the least amount of disturbance to the validation structure of the remaining applicable parts of the plan; this strategy is detailed in [KaH89a]. At the end of this processing, PRIAR would have modified the original plan minimally to make it achieve the original goals from the current (changed) world situation.

By expending a polynomial amount of work during annotation-verification to pinpoint inapplicable parts of the plan, and by controlling refitting, PRIAR attempts to minimize the repetition of planning effort (thereby accruing possibly exponential savings in replanning time). However, while PRIAR tries to reuse as much of the original plan as possible to make the replanning efficient, unless we can circumscribe the ramifications of the unexpected events, the worst case performance of the replanning is still the same as that of planning from scratch. In particular, the unexpected events may be such that very little of the plan can be reused, making the replanning problem degenerate into from scratch planning. This is however to be expected; it is the average case efficiency of replanning that we believe will be improved by this strategy.

## 4. Replanning and Reactivity

Recent research in reactive planning has lead to increased interest in planning and execution architectures that allow an agent to rapidly respond to changes in its world situation. In view of this, an argument could be made that the plan modification strategies such as the ones proposed here would require so high a response time to change the course of action in the event of unexpected events as to be of little use in reactive planning architectures. In the following, we will counter this argument by pointing out the utility of a flexible and conservative plan modification strategy in reactive planning architectures.

Typical strategies for increasing the reactivity of plans involve construction of plans with a high degree of conditionality, which will be ablte to respond to most unanticipated events at execution time by restarting execution from an appropriate conditional branch. As Nilsson points out in [Nil89], to be able to deal with every unexpected event in this way, the planner needs to be able to construct a "*spanning tree plan*" (that represents the course of actions which can take the agent from any world state to the goal state) rather than the usual "*solution-path plan*" (which takes the agent from a specific initial state to the goal state). However, for any realistic planning domain, automatic construction of a spanning tree plan will be prohibitively expensive[6] [Gin89]. This implies that for any realistic plan, there may be several execution time events which cannot be handled by restarting the plan; the agent would have to modify the plan to make it work in the new situation[7].

We suggest a more plausible approach for construction of such reactive plans that retains the ability to replan, and combines the result of replanning with the current plan to incrementally construct a tree-plan [Nil89]. In such a strategy, an unexpected event which requires replanning the first time can be tackled by restarting the tree-plan the next time around. This gives the planner a way of incrementally acquiring a sufficiently reactive plan to respond to the typical types of unexpected events in the domain. It is in this sense that we believe that the replanning framework described here can become a component of a general architecture for reactive planning. In particular, we are currently investigating to see if the replanning framework of PRIAR can be used to construct and extend tree-plans incrementally during execution time. We believe that the flexibility and conservatism of modification offered by PRIAR framework would be of particular utility in ensuring both the efficiency of modification and the compactness of the acquired tree-plan.

## 5. Related Work

One of the assumptions made by the STRIPS plan execution monitoring system PLANEX [FHN72] was that execution time failures can always be handled by restarting (or skip starting) the plan execution from an appropriate past (future) execution point. PLANEX used datastructures called triangle tables to decide the point from where the plan can be restarted. The notion of validation-states in our approach can be seen as a generalization of the triangle table kernels [FHN72] to handle partially ordered plans. However, as we pointed out in the previous sections, unless the plan being executed has sufficient conditionality, the capability to restart will not be sufficient to

[6] Of course, a tree plan that anticipates the typical kinds of unexpected events and provides conditional branches only for them would also serve the purpose. However, it is not clear how this can be done *a priori*.

[7] Ginnsberg [Gin89] argues that knowing what to do when the cached plans are not applicable in the current situation, and incrementally increasing their coverage during runtime is a line of research which would lead to substantial improvements in the behavior of situated automata.

deal with unexpected events at execution time.

Very little past research has addressed the issue of flexible and conservative replanning of partially ordered plans. Work that does address this problem includes [Hay75] and [Dan77]. However, the replanning in these systems does not allow for much more than deleting redundant parts of the plan, and redoing the planning. Wilkins [Wil85] points out that these systems are non-conservative in the sense that they often delete potentially reusable parts of the plan, thus increasing the replanning cost. Wilkin's SIPE [Wil85] planning system has considerably more advanced replanning capabilities. However, SIPE's replanning strategy can also be construed as non-conservative in comparison to PRIAR's. In particular, SIPE could not reuse any already executed parts of the plan, even if the corresponding goals have been undone. It also does not attempt to ensure conservatism of the refitting once high level goals to be reachieved have been suggested. Another difference between the approaches of SIPE and PRIAR concerns the latter's reliance on validation structure as a formal hierarchical representation of plan rationale. Among other things, this gives PRIAR a precise way of specifying replanning actions to cover the various types of applicability failures [Kam89].

## 6. Summary

We have presented a framework for replanning where the plan validation structure, a formal hierarchical representation of plan rationale, forms the basis for execution monitoring, restarting and replanning strategies. We have shown that casting the problem of replanning as that of reusing the original plan to achieve the intended goals starting from the current world state gives rise to a flexible and conservative replanning strategy. We have also discussed the utility of our replanning strategy in a general architecture for reactive planning.

## References

[ChM84] E. Charniak and D. McDermott, "Chapter 9: Managing Plans of Actions", in *Introduction to Artificial Intelligence*, Addison-Wesley Publishing Company, 1984, 485-554.

[Dan77] L. Daniel, "Planning: Modifying non-linear plans", DAI Working paper 24, University of Edinburgh, December 1977. (Also appears as "Planning and Operations Research," in *Artificial Intelligence: Tools, Techniques and Applications*, Harper and Row, New York, 1983).

[FHN72] R. Fikes, P. Hart and N. Nilsson, "Learning and Executing Generalized Robot Plans", *Artificial Intelligence 3* (1972), 251-288.

[Gin89] M. L. Ginnsberg, "Universal Planning Research: A Good or Bad Idea?", *AI Magazine 10*, 4 (Winter 1989).

[Hay75] P. J. Hayes, "A Representation for Robot Plans", *Proceedings of 4th IJCAI*, 1975.

[KaH89a] S. Kambhampati and J. A. Hendler, "Control of Refitting during Plan Reuse", *11th International Joint Conference on Artificial Intelligence*, Detroit, Michigan, USA, August 1989, 943-948.

[KaH89b] S. Kambhampati and J. A. Hendler, "Flexible Reuse of Plans via Annotation and Verification", *Proceedings of 5th IEEE Conf. on Applications of Artificial Intelligence*, 1989, 37-44.

[Kam89] S. Kambhampati, "Flexible Reuse and Modification in Hierarchical Planning: A Validation Structure Based Approach", CS-Technical Report-2334, CAR-Technical Report-4698, Center for Automation Research, Department of Computer Science, University of Maryland, College Park, MD 20742, October 1989. (Ph.D. Dissertation).

[Nil89] N. J. Nilsson, "Teleo-Reactive Agents", Stanford University, Department of Computer Science, 1989. (Draft).

[Wil85] D. E. Wilkins, "Recovering from execution errors in SIPE", *Computational Intelligence 1* (1985).

[Wil88] D. E. Wilkins, "Causal reasoning in planning", *Computational Intelligence 4* (1988).

# Real-Time Search for Dynamic Planning

**Richard E. Korf**
Computer Science Department
University of California, Los Angeles
Los Angeles, Ca. 90024

## Abstract

We present a generic class of algorithms for planning in uncertain, unpredictable, or changing environments. The algorithms plan their actions by searching in a problem space to a search horizon determined by the computational or informational resources available. They then apply a heuristic evaluation function at the frontier nodes to estimate the relative merit of the predicted outcomes, and back up these values to the immediate children of the current state. Finally, one move is made to the best child. Successive actions are based on new searches that make use of new observations of the state of the world. We show that in the case of sliding tile puzzles, a simple combination of heuristic and subgoal searches yields a real-time search algorithm that scales up to arbitrarily large problems. In particular, the running time of the algorithm grows proportionally to the lengths of optimal solutions as the problem size increases.

## Introduction

Some of the most successful programs developed in the AI community are two-player game programs. For example, the best chess programs are comparable to the top 30 players in the country[1]. To some degree, these programs plan moves in uncertain, unpredictable, and changing environments. What general lessons so such programs hold for planning in general?

There are two primary sources of uncertainty in a domain such as chess. The first is uncertainty about the merit of a position, due to the combinatorially explosive size of the problem space. If board configurations could be searched all the way to terminal positions, the value of particular configurations could be determined exactly. However, since computation is severely limited, we have to rely on the value of an inexact heuristic function applied to positions at the search horizon. The second, and more important, source of uncertainty in two-player games is the opponent's moves. While the opponent's move can often be predicted with relative accuracy, the actual moves made often differ substantially.

Given this uncertainty, what is the planning algorithm employed by two-player game programs? Basically, these programs perform a full-width, fixed-depth lookahead search to as a great a depth as the computational resources and time constraints on moves allow. They apply the heuristic function to the frontier nodes, alternately back up the minimum and maximum values, and finally make one move to the child with the best value. Additional techniques, such as alpha-beta, allow them to make the same decisions with substantially less computation. This lookahead search is the planning component of a chess program.

After the machine makes its move, and the opponent makes a moves, the machine performs an entirely new search for the next move. Thus the plan formulated in the previous search is used only to make one move. After that move, the plan is discarded and a completely new plan is constructed for the next move. At first glance this seems rather wasteful. The rationale for this strategy is dictated by the combinatorics of the problem space.

Assume that at any given point, there are $B$ different moves that can be made, and that the opponent will then have $B$ alternative responses available. Since all possible moves and all possible responses are considered, the portion of the search space examined in one search that is still relevant after the machine chooses its move and the opponent responds is only $1/B^2$. In chess, the branching factor $B$ is typically about 35, resulting in wasted computation on the order of one part in 1225. Thus, there is little to be lost by replanning each move from scratch.

One could artificially add more uncertainty and unpredictability to the chess domain in any of a number of ways. For example, pieces on the board could randomly move at random times, or the move selected by the machine might only be executed with some probability, with other moves made in the remaining cases. While these changes would make the game more difficult to play, the same algorithms would be employed.

In particular, the strategy of replanning each move after the previous moves have been executed would be even more important. If the board configuration is changing, then planning should be based on the most recent observations of the board, and replanning prior to each move provides the most up-to-date information. Similarly, if the actual moves made are unpredictable, then subsequent moves can be more effectively planned based on the observation of previous moves, rather than their predicted values.

## Single-Agent and Multi-Agent Search

In previous work, we have extended the techniques of fixed-depth lookahead search to single-agent problems and multi-agent cooperative and competitive problem solving. Each of these algorithms interleaves planning and execution, and replans for each move from scratch in constant time.

For the multi-agent case, the minimax algorithm can be generalized to an algorithm called maxn[2]. Instead of a single heuristic value associated with each position, there is an N-tuple of values, with each component corresponding to the utility of that position for one of the $N$ agents in the game. The agents alternate moves, try to maximize their utilities, and are indifferent to the utilities of the other agents. The backed-up value of a position where player $i$ is to move is the entire N-tuple of the child for which component $i$ is a maximum.

In [3] we examine the extension of alpha-beta pruning to multi-player games. We find that shallow pruning is valid, but not deep pruning. Furthermore, while shallow pruning is quite effective in the best case, in the average case the asymptotic complexity of the search is not reduced over the brute-force case. Thus alpha-beta pruning is not effective on the average with more than two players.

For the single-agent case, we have developed several real-time algorithms[4]. Minimin lookahead search is a specialization of minimax search to single-agent problems. In this case, the A* evaluation function[5], $f(n) = g(n) + h(n)$, is applied to the frontier nodes, and the minimum values of children are backed-up to the parents. A single move is made to the child of the current state with the best backed-up values. Since in practice most heuristic functions yield an $f(n)$ function that is monotonically nondecreasing, branch-and-bound can be applied to dramatically speed up this search without effecting the decisions made. This algorithm is called alpha pruning, by analogy to alpha-beta pruning,

Since in a two-player game moves cannot be undone, the minimax algorithm is simply repeated for each successive move. In the single-agent case, however, backtracking over previously committed moves may be permissible, and in fact necessary in some situations. The challenge is to permit backtracking while avoiding infinite loops. We developed an algorithm, called Real-Time-A*, that backtracks when it is rational to do so, is guaranteed to find a solution when one exists, and makes locally optimal decisions on a tree. This algorithm effectively solves much larger problems than are solvable by A*, primarily by sacrificing optimal solutions. There is also a learning version of the algorithm that eventually learns exact heuristic values over repeated problem-solving trials.

Both the multi-agent and single-agent algorithms described above share with the two-player algorithms the property that significant replanning is done for each successive move. In the single-agent case, some information is saved from one move to the next principally to enable intelligent backtracking. In general, however, for the same reasons given in the discussion of two-player games, these algorithms would be very effective in uncertain, unpredictable, and changing environments. An overview of the main results in both of these areas can be found in [6].

## Scaling of Real-Time Search

Current research is focussed on extending these techniques to scale up to arbitrarily large problems. What does it mean for a search algorithm to scale within a class of problems? Since the optimal solution lengths may increase as problem size increases, and any algorithm that solves a problem must use at least as much time as the length of the optimal solution, the best we can hope for is performance that grows linearly with the length of optimal solutions. We will say that a search algorithm scales if its time complexity is linear in the length of the optimal solution. Note that this also guarantees that the solution lengths found will be bounded by a constant times the length of the optimal solutions. In the remainder of this section, we will consider the well-known sliding tile puzzles, such as the Eight Puzzle, Fifteen Puzzle, etc., as a case study in the scaling of real-time search techniques.

Consider the manhattan distance heuristic function for sliding tile puzzles. It is computed by determining the distance along the grid of each individual tile from its goal position, and summing these values over all the tiles. By giving up the requirement for optimal solutions, algorithms such as RTA* can effectively solve puzzles as large as the ten-by-ten ninety-nine puzzle, using the manhattan distance evaluation function.

In extending these algorithms to even larger problems, however, what typically happens is that almost the entire puzzle will be solved, leaving two tiles swapped out of place. The algorithm then spends an inordinate amount of time trying to reduce the heuristic to zero by manipulating tiles far removed from the two swapped tiles. It isn't smart enough to realize that it won't reach the goal unless it fixes those two tiles, regardless of how many tiles must be disturbed in the meantime. As the problem size increases, the number of moves required to rectify these impasses grows. Thus, pure heuristic search using this evaluation function does not scale up according to our definition. As another example, consider assembling a car engine from scratch using a heuristic function that gives credit for parts in their correct places. Such an algorithm could easily leave a piston out of the block, then proceed to assemble the rest of the engine without realizing until very much later that it will have to be largely disassembled to insert the piston.

The only way around this problem is to somehow order the subgoals in the problem. For the sliding tile puzzles, the obvious subgoals are to correctly position the individual tiles one at a time. This suggests the following algorithm, which we call pure subgoal search[7]. Given a sequence of tiles, $t_1, t_2, \ldots, t_n$, and a current state, $s$, the current focus is defined as the first tile $t_i$ in the ordering that is not in its goal position in the current state. If the current focus is tile $i$, search the space until a state is found in which the first $i$ tiles are in their goal positions, and then execute the corresponding sequence of moves that lead to this state. Recalculate the new focus, which must be strictly greater than $i$, and continue until the goal is reached.

A natural way to perform the individual searches is to use depth-first iterative-deepening[8]. This algorithm performs a series of depth-first searches to successively greater depths until the current focus tile and all preceding tiles in the solution order are solved. Note that since each successive search strictly increases the number of solved tiles, no memory is required between searches, other than the value of the current focus. Furthermore, since depth-first search is used, the memory required during a search is only linear in the depth of the search.

This idea is used by Ruby and Kibler in this same domain[9]. One of the problems immediately raised is how to determine the order of the subgoals, or in other words, the order the tiles are to be solved in. Ruby and Kibler give an algorithm that automatically generates reasonable solution orders, based on the *openness* heuristic. A closely related idea was presented in [10]. In addition, their system learns to improve its performance by caching the results of previous searches in the form of new subgoal sequences.

Unfortunately, as in the case of pure heuristic search, pure subgoal search by itself does not scale in this domain. The reason is that as the puzzle size increases, the maximum distance that a tile may have to travel to reach its goal position increases, and hence the search horizon required to solve the most difficult subgoal increases.

While neither heuristic search nor subgoal search by themselves scale up in this domain, a combination of the two techniques does scale. The idea is as follows. At the top level, we order the tiles and perform a subgoal search to solve them one at a time. Within a particular subgoal search, however, we employ a heuristic evaluation function based primarily on the current focus tile in order to guide its progress to its goal position. The obvious evaluation function is the manhattan distance of the current focus tile from its current position to its goal position. In order to prevent previous tiles in the solution order from being permanently dislocated, we actually calculate the manhattan distance of all tiles in the solution order up to and including the current focus, and sum these values. The resulting algorithm then performs depth-first iterative-deepening as before, terminating when this partial manhattan distance value is decreased from its current value. Another way of viewing this algorithm is that we have increased the granularity of the subgoals by making moving the current tile one step closer to its goal position by the manhattan distance measure a subgoal in itself.

Unfortunately, this combined algorithm doesn't quite scale up. The reason is that once the current focus tile is correctly positioned, the focus moves to the next tile in the solution order, and the algorithm must "find" this tile. In other words, the blank position must be maneuvered to a position adjacent to this tile before the tile can be moved. Since the distance between the blank and the next tile to be solved can grow with increasing problem size, this step may require increasing the worst case search horizon for larger problems, thus violating the scaling requirement.

Once we think of the heuristic function as estimating the cost of solving an individual subgoal, the solution to this problem becomes obvious. Namely, add to the manhattan distance of a tile from its goal position the manhattan distance of the blank from the given tile, minus one. The reason for subtracting one is that the precondition for moving a tile is that the blank be adjacent to it, or in other words, the manhattan distance of the blank to the tile is exactly one. This new heuristic gives a lower bound on the number of moves required to solve a particular tile. If more than one tile less than or equal to the focus tile in the solution order is out of place, then after summing the manhattan distances of all these tiles from their goal locations, we add the minimum manhattan distance of the blank to any of the tiles in question. Note that this modification of the heuristic captures the common sense notion that moves that carry the blank away from a given tile are generally not relevant to the positioning of that tile.

This new heuristic function is then combined with depth-first iterative-deepening. Since the modified heuristic function is still a lower bound on the number of moves required to solve current the focus tile, an important pruning opportunity presents itself. In a given depth-first iteration, the difference between the current depth of a state and the search horizon for that iteration is the maximum amount that the heuristic function of the given state can possibly decrease during that iteration. If this is not sufficient to decrease the heuristic value of the initial state for that iteration, the given state and all of its progeny can be pruned from further consideration during that iteration.

Even with a good solution order, impasses still occur where the current focus tile cannot be moved closer to its goal position without violating previously solved tiles. The search component of the algorithm provides a simple and natural mechanism for dealing with these subgoal interactions. For example, the way that a row of tiles is solved by this algorithm, assuming that they occur sequentially in the solution order, is as follows. All but the last tile are solved one at a time without moving previously solved tiles, nor with any regard for the positions of the following tiles in the order. Assuming that the final tile does not fortuitously appear in its goal position at this point, it will be moved to a position adjacent to its goal position, without disturbing the rest of the row. Then, a somewhat deeper search is required to correctly position the final tile, and involves temporarily moving the next to last tile in the row. One solution to this problem requires ten moves. Thus, a simple search algorithm replaces complex special case reasoning about subgoal interactions.

Assuming that the goal state leaves the blank in a corner, the solution order we chose for our experiments solves the $N \times N$ puzzle by first solving the row and column furthest from the final position of the blank. This reduces the problem to an $N-1 \times N-1$ puzzle. Using this solution order, the worst case search horizon required to solve any subgoal for any size problem is thirteen moves. Both the solution lengths, and the total number of nodes generated, grow proportional to $N^3$, where $N$ is the length of one side of the puzzle. We can easily show that the optimal solution lengths must also be $O(N^3)$. The reason is that there are $N^2 - 1$ tiles to be moved, and the average manhattan distance of an individual tile to its goal position is of order $N$. Thus, since the asymptotic time complexity of our algorithm is the same as the asymptotic growth of the length of an optimal solution, the algorithm scales up according to our definition.

Furthermore, since the algorithm saves no information from one move to the next, but recomputes it from scratch after every move, it is ideally suited to planning in an uncertain and unpredictable environment. While the sliding tile domain doesn't contain much uncertainty, it could be artificially added by introducing a certain amount of random rearrangement of the tiles, and actually executing selected moves with only a certain probability. The point is that in such a situation, the algorithm described here would be even more appropriate.

## Discussion and Conclusions

We have presented a generic class of algorithms for planning in uncertain and unpredictable environments. Their suitability for such domains stems from the fact that they recompute their plans after every action, based on new observations of the environment.

In addition, we developed and implemented a particular algorithm for sliding tile puzzles that scales up to arbitrary size puzzles, in the sense that the running time of the algorithm, and the solution lengths generated grow proportional to the lengths of optimal solutions. The algorithm is a simple combination of heuristic and subgoal search.

One may criticize this work as simply the design of a special purpose algorithm for solving sliding tile puzzles. Indeed, other scalable algorithms for this problem have been reported in the literature[11]. While there is some merit to this position, it is not entirely accurate. We view our algorithm as a very simple combination of two well-known weak methods: heuristic search and subgoal search, and not a dedicated algorithm designed for a specific problem.

The first piece of domain specific knowledge that we have used is the fact that the puzzle is composed of individual components, each one of which must be solved to solve the entire problem. This property is true of many combinatorial problems, and is immediately obvious when one is presented with such a problem. The second item of knowledge is the particular order in which the individual components are solved. While some solution orders for this problem would not result in a scalable algorithm, such as solving the tiles furthest removed from the final position of the blank last, most solution orders that solve the furthest tiles first will scale. This idea is also fairly obvious, comparable to the notion that in painting the floor of a room one should start with the corner furthest from the door, or that assembly tasks should proceed from the inside outwards. In addition, domain-independent algorithms have been developed that automatically produce such solution orders[10,9]. The final bit of domain-specific knowledge is the manhattan distance heuristic on the individual tiles. Again, we claim that this is fairly obvious and there exists a good theory[12] and implemented programs[13] that automatically derive such heuristics for problems.

As a demonstration of the generality of these ideas, one can easily see how to apply them to a different problem such as Rubik's Cube. There the individual subgoals would be to correctly position and orient the individual cubies. Furthermore, it is easy to generate plausible solution orders for this problem. Finally, we can also see how to calculate a three-dimensional manhattan distance heuristic for the individual cubies, and even one that includes orientation as well. Unfortunately, it is not clear that such an algorithm, not any other algorithm, would scale up for larger versions of Rubik's Cube.

In the case of the sliding tile puzzles, however, a simple combination of heuristic and subgoal search yields an algorithm that effectively scales up to arbitrarily large problems.

## Acknowledgements

I'd like to thank John McCarthy for helpful discussions concerning this work. This research was supported by an NSF Presidential Young Investigator Award, NSF Grant IRI-8801939, and an equipment grant from Hewlett-Packard.

## References

[1] Berliner, H., "Deep-Thought wins Fredkin Intermediate Prize", *AI Magazine*, Vol. 10, No. 2, Summer, 1989.

[2] Luckhardt, C.A., and K.B. Irani, An algorithmic solution of N-person games, *Proceedings of the National Conference on Artificial Intelligence (AAAI-86)*, Philadelphia, Pa., August, 1986, pp. 158-162.

[3] Korf, R.E., Multi-player alpha-beta pruning, to appear in *Artificial Intelligence*, 1990.

[4] Korf, R.E., Real-time heuristic search, to appear in *Artificial Intelligence*, 1990.

[5] Hart, P.E., N.J. Nilsson, and B. Raphael, A formal basis for the heuristic determination of minimum cost paths, *IEEE Transactions on Systems Science and Cybernetics*, SSC-4, No. 2, 1968, pp. 100-107.

[6] Korf, R.E., Depth-limited search for real-time problem solving, to appear in *Real-Time Systems*, 1990.

[7] Korf, R.E., Planning as search: A quantitative approach, *Artificial Intelligence*, Vol. 33, No. 1, 1987, pp. 65-88.

[8] Korf, R.E., Depth-first iterative-deepening: An optimal admissible tree search, *Artificial Intelligence*, Vol. 27, No. 1, 1985, pp. 97-109.

[9] Ruby, D., and D. Kibler, Learning subgoal sequences for planning, *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, Detroit, Mich, Aug. 1989, pp. 609-614.

[10] Korf, R.E., *Learning to Solve Problems by Searching for Macro-Operators*, Pitman, Boston, 1985.

[11] Ratner, D., and M. Warmuth, Finding a shortest solution for the NxN extension of the 15-Puzzle is intractable, in *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI86)*, Philadelphia, Pa., 1986.

[12] Pearl, J. *Heuristics*, Addison-Wesley, Reading, Mass, 1984.

[13] Mostow, J., and A. Prieditis, "Discovering admissible heuristics by abstracting and optimizing: A transformational approach", *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, Detroit, Mich, Aug. 1989, pp. 701-707.

# The Role of Meta-Reasoning in Dynamic Environments

**Daniel R. Kuokka**

Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

## Introduction

It has been argued that systems operating in complex, dynamic environments cannot rely on traditional planning. Instead, such systems must respond directly to the environment, an approach called situated action. However, many problems are more naturally solved via planning, such as when errors are very costly, or when the appropriate action is not apparent from the environment. This suggests that a system must be able to integrate a variety of problem solving tactics in order to operate effectively.

The problem is further complicated by the practical limitations on sensors. An agent cannot expect to have complete, or even sufficient perceptory input at all times. Hence, it may be necessary to take actions in order to augment the current sensory input. For example, an agent may have to move to gain a different view of an object in order to determine its shape and identity. This implies a deliberative aspect to the control of problem solving.

This paper describes a system, called MAX, that satisfies the above requirements by encoding and controlling its problem solving strategies explicitly, much like traditional systems control their basic actions. In addition, the system is designed to remain reactive to spontaneous stimuli. Thus, MAX can select and perform arbitrarily complex and protracted reasoning while remaining responsive, both to the external world and its internal state.

The next section gives a brief overview of the MAX architecture, including the knowledge representation that is instrumental to the requisite meta-reasoning. Subsequent sections cover the means by which MAX selects its problem solving technique, augments its sensors, and responds to unexpected situations. Finally, the relationship to other systems is explored, and conclusions are presented. The capabilities described in this paper have been implemented and tested in several domains, including a simulated household robot.

## The MAX Architecture

The MAX architecture is designed to support the explicit control of problem-solving. Since systems generally only reason about basic actions, the control of problem solving is a form of meta-level reasoning. There are several hurdles that must be overcome to permit such reasoning. First, the knowledge representation must be powerful enough to express the complexities of problem solving, while remaining simple enough to be the object of reasoning. Second, the control structure must allow the explicit problem solving knowledge to be directly executed without becoming blinded to the environment. Finally, since almost none of the reasoning capabilities of the system are hardwired, a significant body of reasoning expertise must be encoded as explicit knowledge. This paper can only give a brief overview of MAX, those interested in a more detailed description should refer to (Kuokka, 1990).

### Knowledge Representation

The main hurdle that must be overcome to support meta-level reasoning is the definition of an appropriate knowledge representation. The representation used by MAX is purely conjunctive first order logic augmented with a new data structure, called the *lframe*. An lframe can be thought of as an entire logical theory, or a state. Since a state is the basic object of much meta-level reasoning, lframes are particularly well suited to their intended task.

There are two main features of the representation that are central to its utility. First is the association of variables with terms instead of sentences, allowing a single term to represent an infinite number of possibly parameterized objects. This is crucial to the succinct representation of complex knowledge. The second feature is the definition of a set of basic relations over lframes. Five relations have been defined so far: **match**, **not-match**, **union**, **intersection**, and **difference**, which perform the associated set operation on the sets of literals constituting lframes. These relations allow entire states to be compared and constructed, a basic requirement of any reasoning system. In essence, they provide a lexicon for reasoning, allowing learning and problem solving to be implemented at a high level of detail.

Two example lframe structures are shown in figure 1: a meta-level rule, and the meta-level state against which the rule's condition would be matched. An lframe is depicted as a set of literals enclosed in square brackets. The `$vars` syntax declares the associated variables to be local to the associated lframe, thereby allowing it to represent many different instantiated terms. The function of the rule is to determine if a base-level operator is applicable to a problem by matching the operator's effects against the difference between the current state and goal. Notice how this is easily encoded in terms of the basic relations over lframes once the goal and state information is made explicit. The action portion of the rule is to fire an operator called `subgoal` (the specification of the operator is not shown, but world be at the same level as the rule).

```
Meta-Level Rule:
  [($vars ?state ?goal ?diff ?name ?add)
   (cond [(state ?state)
          (goal ?goal)
          (difference ?goal ?state ?diff)
          (domain [(operator ?name
                       [(add ?add)])])
          (match ?diff ?add)])
   (action subgoal [(operator ?name)])])

Meta-Level State:
  [(state [(on blocka blockb)
           (on blockb blockc) ...])
   (goal [($vars ?block)
          (on blockc ?block)
          (on ?block blocka)])
   (domain [(operator put-down
              [($vars ?b1 ?b2)
               (pre [(holding ?b1) ...])
               (del [(holding ?b1) ...])
               (add [(on ?b1 ?b2])])])])]
```

Figure 1: Partial encoding of a planner

There are many issues central to the lframe representation that cannot be covered here. The point of this section is only to introduce the general flavor of the lframe representation, and to illustrate how problem solving can be encoded succinctly and explicitly. In particular, entire states are the object of reasoning, which can be compared and constructed to produce arbitrary reasoning behavior.

## Control Structure

The above example demonstrates how reasoning can be encoded, but some kernel is still required for processing to occur. However, since all knowledge that performs reasoning is supposed to be explicit, the kernel can be extremely simple. In fact, to build significant expertise into the fixed kernel is to place learning and reasoning where it cannot be controlled.

The kernel can best be described as a procedure-based production system interpreter. Instead of there being one central production memory, there are many small production memories arranged as procedures. Each production memory is called a *behavior*, since it implements some form of reasoning behavior. There is only one behavior active at any time, but a behavior can call another behavior, installing it as the new active behavior. Thus, there is a stack of pending behaviors, much like a stack of pending procedure calls.

A behavior consists of a set of rules and a set of operators. The rules' conditions are matched against working memory, and the rules' actions specify which operator to fire. The operators, in turn, specify the adds and deletes to be applied to working memory. In fact, figure 1 is really a portion of a behavior that does means-ends planning, and is directly executable by the kernel.

A parent behavior invokes a child behavior via a *compound operator*. A compound operator is similar to a simple operator, except when fired, an associated behavior is invoked as the current behavior. When the child behavior is finished, it returns bindings that are then asserted by the deletes and adds of the compound operator within the parent behavior. In this way, a single operator can perform an arbitrary amount of reasoning. For example, the planning behavior of figure 1 would be invoked by an operator called `plan` within a higher-level behavior. Such an operator is shown in figure 2. When the planning behavior is finished, the generated plan is returned and bound to the variable `?plan` so it can be asserted within the higher-level behavior.

```
[($vars
  (pre [(state ?state)
        (goal ?goal)
        (domain ?domain)])
  (del [])
  (add [(plan ?state ?goal ?plan)])
  (compound plan-behav [(return ?plan)])]
```

Figure 2: Compound operator that invokes plan behavior

In addition to the current behavior, there is also a collection of monitors that run in the background. A monitor is structurally identical to a behavior, but can be defined dynamically. This allows the monitoring of unforeseen conditions independent of the current behavior. The monitor facility is extremely important to the reactivity of the system, and is discussed in more detail below.

So far, only an empty architecture has been described. The specific reasoning knowledge that constitutes each behavior represents the majority of any MAX-based system. In particular, the expertise that allows the system to operate in complex and dynamic environments is implemented as behavioral knowledge on top of the architecture. The following sections describe several of the more pertinent capabilities.

## Integrating Planning and Reaction

Since planning and situated action are applicable in different situations, there must be a means to select between them intelligently. Moreover, the very invocation of situated action requires deliberation, since a reactive procedure is generally applicable to a single class of goals. MAX provides the means to perform such reasoning by representing the various problem solving options as explicit choices within a meta-reasoning behavior.

The choice of problem solving method is made within the *solve* behavior, which is responsible for achieving a goal. The solve behavior is invoked directly by the system's top-level behavior, which is responsible for choosing the current focus of attention for the agent. The inputs to the solve behavior are the goal, the current state, and the domain theory specifying the allowable actions in the world. The domain theory may also contain a variety of other knowledge, such control rules for planning, reactive procedures, and domain axioms. The main operators available within the solve behavior are to plan, to execute a plan, and to run a reactive procedure. There is also an operator to obtain knowledge, which is discussed in the following section.

Currently, a relatively small set of heuristics are used within the solve behavior. If situated action is explicitly preferred for the domain, or the domain is known to be reversible, and there is a reactive procedure available, then execute the procedure; otherwise, use plan-then-execute. A goal class is associated with each reactive procedure which is matched against the current goal. This allows the appropriate reactive procedure to be executed.

It is unlikely that a single problem solving method is appropriate for all aspects of a problem. Thus, MAX allows a problem to be decomposed hierarchically, thereby allowing finer grained control of problem solving. The approach is to encode domain actions as compound operators that recursively invoke the solve behavior. To better illustrate this, consider a typical domain which has operators like `move` and `put-down`. Putting down an object in a specific location is still a fairly complicated operation, possibly requiring a significant amount of problem solving. This is achieved in MAX by making `put-down` a compound operator which invokes the solve behavior recursively.

The solve behavior invoked by put-down is instantiated with a different state, goal, and domain theory. The domain theory represents the details involved in putting down and picking up objects, such as positioning the robot in the correct position, avoiding obstructing objects, and manipulating the arm. The state and goal, correspondingly, are specified to a greater level of detail. Unlike the higher-level domain, the operations performed in the put-down domain are better controlled via situated action due to the increased attention to detail and the need for hand-eye coordination. Thus, solving a problem in this domain generally executes a reactive procedure instead of attempting to plan.

A portion of the behavior to pick up an object is shown in figure 3. This rule states that if the goal object (to be picked up) is perceived to be obstructed by another object, and the other object is known to be heavy, then move around the obstructing object. Furthermore, the location to be moved to cannot be in the set of failed locations; otherwise, the behavior could loop forever. Of particular importance in this example is the use of knowledge that is not directly perceivable (e.g., the weight of the obstructing block), and the memory capability (e.g., the set of failed locations). This allows the system to perform actions outside the scope of simple situated action, yet still respond directly to perceptions of the state.

Taking a step back to examine this whole scenario, a complex interleaving of planning and situated action emerges. At the highest level, a plan was formed over very abstract operators. This allows the agent to avoid costly and possibly destructive mistakes. The abstract operators are executed by performing situated action in the more specific, low-level domains. This avoids the complexities of planning over such details. It is quite natural to extend this technique to arbitrarily many levels, each of which uses the most appropriate method.

```
[($vars ?g ?o ?rl ?gl ?ol ?fls ?nl)
 (cond [(see [(at robot ?rl)
              (at ?g ?gl)
              (at ?o ?ol)
              (in-line ?ol ?rl ?gl)])
        (state [(weight ?o heavy)
                (next-to ?nl ?rl ?gl)])
        (failed-locs ?fls)
        (not-match ?fls [(elt ?nl)])])
 (action move [location ?nl])]
```

**Figure 3:** Meta-level situated action

As in most such systems, reactive procedures in MAX are approximate; they may perform an incorrect action. Unlike most other systems, though, MAX has an inherent recovery mechanism. Since the invocation of the reactive procedure was a deliberate choice, the system can make an alternate choice if the first fails. The alternate may even be to perform traditional planning. Of course, an incorrect reactive procedure may get the agent into irreversible trouble, but such cases are assumed to be rare (recall that the invocation of situated action is based on the assumption of a forgiving environment). Thus, the system can more freely make use of approximate reactive procedures.

## Sensor Augmentation

It is not practical to assume that the sensory capabilities of an autonomous agent can automatically present sufficient information about the world. The cognitive and physical capabilities of the agent must be used to augment perceptions. In particular, information no longer within the field of perception of the agent should be remembered, and actions may be required to direct the field of perception. The meta-level reasoning capability of MAX allows the system to perform both of these activities.

Remembering past states, and combining them with perceptions is relatively straightforward. Perceptions are presented as explicit states (e.g., `see` in figure 3); thus, they can be easily manipulated. For example, the perception lframe can be unioned with the modeled state lframe, thereby updating the current model of the world. The explicit nature of perceptions is of great utility, and is an integral part of the examples of figures 3 and 4.

The process of directing action to aid perception is somewhat more complex. The general technique is to detect the absence of required knowledge during problem solving, perform recursive reasoning and problem solving to obtain the required knowledge, and then proceed with the goal as usual. Missing knowledge is detected by a rule that checks for a condition in a goal that should be known in the current state, but is not.

For example, assume the robot has a goal of fetching a hammer, but the hammer has been moved to an unknown location. The system fires the solve behavior, which in turn, fires the plan behavior. During planning, a subgoal is generated containing a literal stating that the tool is at some unbound location. However, the state contains no assertion as to the location of the tool, since its location is unknown. This situation is detected by a rule in the planner, and the current planning path is aborted. If no alternate paths lead to a solution, the planner returns an error condition indicating the need for additional knowledge.

Upon receipt of the error condition, the solve behavior must attempt to obtain the required knowledge in order to satisfy the goal. This is done by firing an exploration behavior (exploration is actually invoked via an intermediate behavior that maps from a domain independent goal to a domain-specific method). The inputs to the exploration behavior are the needed knowledge and the current knowledge. Exploration uses a number of heuristics to efficiently search the environment. For example, adjacent rooms are explored first, as are those that the robot has not been in recently. Also, unlocked doors are favored over locked doors. It is within this behavior that the robot actually moves from room to room until it detects the hammer. Furthermore, a great deal of recursive problem solving may be required to implement the exploration activities.

To continue the example, once the hammer is located, the newly augmented model of the house is updated and returned to the problem solving behavior. Now equipped with the necessary knowledge, the planner can successfully produce a plan (starting from the new current state), which is then executed as in the previous section.

This example demonstrates how meta-level reasoning allows the system to detect a lack of information and direct its attention to getting that information. Such reasoning is outside the scope of traditional planning as well as basic situated action. However, both planning and situated action play an important role in the entire behavior.

## Monitoring the Environment

So far, the discussion has focused on operating in complex and slowly changing environments where the main concerns are the proper integration of planning and action. However, real environments can also change rapidly and dangerously, and a system must be able to respond to such changes in a timely fashion. MAX copes with this requirement by supporting monitors that cast a wary eye on the world, regardless of the current active behavior. Since the basic control cycle of MAX is forward chaining, such an extension fits naturally within the basic architecture.

A monitor can be defined within any behavior, and the rules within the monitor will be checked on each cycle as long as the behavior is pending. If a rule within an active monitor is satisfied, it fires, causing the associated operator to fire. Generally, only two operators are used in monitors: return from the current behavior, and return to the top-level goal selection behavior. The former allows the behavior that defined the monitor to deal with the unexpected condition, and the latter allows the top-level goal selection behavior to decide on an action.

Even though monitors can be defined dynamically, there is a default monitor that covers a great many cases. The rule simply checks for conditions in the current real world state that both match an emergency situation and are unexpected. For example, even if fires are in the set of emergencies, the robot will not react to a fire in the fireplace. The encoding of the default monitor rule is shown in figure 4.

```
[($vars ?emergency ?percepts ?state)
 (cond [(emergencies [(elt ?emergency)])
        (see ?percepts)
        (match ?percepts ?emergency)
        (state ?state)
        (not-match ?state ?emergency)])
 (action interrupt [])])
```

**Figure 4:** A default monitor rule

To better illustrate monitors, consider once again the example of fetching the hammer. Assume the robot is in the midst of computing the high-level plan to get the hammer when an ember explodes out of the fireplace onto the carpet. The default monitor rule detects the fire on the carpet as an unexpected emergency condition, and interrupts the planning behavior in favor of the top-level goal selection behavior. The goal selection behavior prioritizes the fire as being more important than any other current goals, and invokes the extinguish behavior. Once finished, the robot returns to its previous task (in fact, resumption is a very complex issue that is too involved for this discussion).

The monitor mechanism of MAX has several strong points. First, it keeps the knowledge of emergency situations separate from unrelated behaviors. For example, the planning behavior should not be compelled to watch the real world; it is reasoning about hypothetical states. Second, monitors can be defined dynamically, allowing specific conditions that become apparent only at run-time to be monitored. Third, monitors are always active; thus, the system will react to emergencies within one cycle. Finally, the capability to recursively call the top-level goal selection behavior, which is intended to handle focus of attention, allows a more intelligent response and avoids duplication of knowledge.

## Related Work

There are apparently two views of planning in dynamic domains. On the one hand, there are the situated actors, such as (Agre & Chapman, 1987, Brooks, 1985, Kaelbling, 1988, Schoppers, 1987). These authors argue convincingly that planning is inappropriate for many everyday tasks. The other side of the issue is represented by the serious planners, such as (Carbonell et al., 1990, Wilkins, 1984), and those who focus on making planning more appropriate for dynamic domains (Dean & Boddy, 1988, Doyle et al., 1986). These authors clearly believe that some form of planning is needed.

Even though the validity of both positions is widely acknowledged (even by the opposite camps), and the need for a unifying superstructure has been stated, few systems pursue an integrated approach. Work that spans both reasoning and reacting includes ROBO-SOAR (Laird et al., 1989) and PRS (Georgeff, 1987), and the task control architecture for the CMU planetary rover (Simmons & Mitchell, 1989). These systems, along with MAX, span the spectra of generality, uniformity, and practicality (in their current incarnations). Further work is needed to draw meaningful comparisons. In particular, increased effort must be devoted to the interaction with real domains. There would appear to be limited potential in examining dynamic planning within simulated blocks worlds.

## Conclusions

The basic motivation of this paper is that a variety of capabilities are required of a system that operates in complex or dynamic environments, such as traditional planning, situated action, memory, sensor augmentation, and reactive task interruption and resumption. Given that such a variety of behaviors are required, there must be a mechanism to effectively integrate them.

The central claim is that a form of situated action performed at the meta-level yields the benefits of both traditional planning and situated action, as well as provides the means to integrate a variety of other behaviors. The system can directly respond to the environment, yet it can perform traditional planning as well as complex meta-reasoning, reflection, and knowledge obtainment. Furthermore, the reactivity of the system is maintained at all times by the use of monitors.

This approach provides a great deal of power, but at what cost in terms of efficiency? MAX rules only perform matching; no chaining can occur within the basic cycle. More elaborate processing emerges over many cycles. Furthermore, since the architecture provides direct access to the matcher (via the `match` relation) the explicit base-level matching present in meta-level rules incurs no interpretive overhead. Thus, MAX is efficient to the extent that any system based on matching is efficient.

## Acknowledgments

This research was supported in part by ONR grants N00014-79-C-0661 and N0014-82-C-50767, DARPA contract number F33615-84-K-1520, NASA contract number NCC 2-463, and a grant from the Hughes Aircraft Corporation.

## References

Philip E. Agre and David Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of the National Conference on Artificial Intelligence*. Seattle, WA, 1987.

Rodney Brooks. *A Robust Layered Control System for a Mobile Robot* (Tech. Rep. 864). Artificial Intelligence Laborabory, Massachusetts Institute of Technology, 1985.

J.G. Carbonell, C.A. Knoblock, and S.N. Minton. PRODIGY: An integrated architecture for planning and learning. In Kurt Van Lehn (Ed.), *Architectures for Intelligence*. Lawrence Erlbaum, 1990.

Thomas Dean and Mark Boddy. An analysis of time-dependent planning. In *Proceedings of the National Conference on Artificial Intelligence*. Saint Paul, MN, 1988.

Richard J. Doyle, David J. Atkinson, and Rajkumar S. Doshi. *Generating Perception Requests and Expectations to Verify the Exectuion of Plans* (Tech. Rep. JPL D-3394). Artificial Intelligence Group, Jet Propulsion Laboratory, 1986.

M. Georgeff and A. Lansky. Reactive Reasoning and Planning. In *Proceedings of the National Conference on Artificial Intelligence*. Seattle, WA, 1987.

Leslie Pack Kaelbling. Goals as parallel program specifications. In *Proceedings of the National Conference on Artificial Intelligence*. Saint Paul, MN, 1988.

Daniel R. Kuokka. *The Deliberative Integration of Planning, Execution, and Learning*. Doctoral dissertation, School of Computer Science, Carnegie Mellon University, 1990.

J.E. Laird, E.S. Yager, C.M. Tuck, and M. Hucka. Learning in tele-autonomous systems using Soar. In *NASA Conference on Space Telerobotics*. Pasadena, CA, 1989.

M.J. Schoppers. Universal plans for reactive robots in unpredictable environments. In *Proceedings of the International Joint Conference on Artificial Intelligence*. Milano, Italy, 1987.

Reid Simmons and Tom Mitchell. A task control architecture for autonomous robots. In *Proceedings of the Conference on Space Operations and Autonomous Robotics*. Houston, TX, 1989.

David E. Wilkins. Domain-independent planning: Represenatation and plan generation. *Artificial Intelligence*, 1984, Vol. 22.

# Integrating Planning and Execution in Soar[*]

John E. Laird
Artificial Intelligence Laboratory
The University of Michigan
1101 Beal Ave.
Ann Arbor, Michigan 48109-2110

The classical approach to planning in artificial intelligence (AI) has emphasized a distinction between planning and execution. When a problem is presented, a plan is developed, and then the plan is executed. Difficulties with this approach arise in domains where the outcome of planned actions are uncertain or the environment is unpredictable. One advantage of the classical approach is that it separates planning and execution into two modules so that each module can be optimized for its specific task. The end result is that planning and execution are considered in isolation and planning knowledge is separated from execution knowledge. The weak link in this approach is the limited communication that is available between the two modules in domains where a high degree of run-time interaction between planning and execution is necessary because of unexpected changes occur in the environment.

In this paper, we present an alternative to classical planning based on the Soar architecture.[1] Soar successfully avoids the difficulties inherent to the modular approach to planning and execution by using a single architecture and knowledge base for both. Soar's approach is distinguished by the following characteristics: planning is invoked automatically when execution knowledge is inadequate; plans are not represented explicitly but stored as individual control rules; execution decisions are based on all previous experiences, not just the current plan; and learning improves both planning and execution. For the sake of brevity, work related to this approach will not be explicitly referenced. See the other papers in this volume for alternative approaches.

The remainder of the paper presents Soar and emphasizes its capabilities for integrating planning and execution. Throughout this presentation we demonstrate these capabilities using two systems. The first system is called *Robo-Soar* (Laird et al., 1989) , and it controls a Puma robot arm using a camera vision system, as shown in Figure 1. The vision system provides the position and orientation of blocks in the robot's work area, as well as the status of a trouble light. Robo-Soar's task is to align blocks in its work area, unless the light goes on, in which case it must immediately push a button. The environment for Robo-Soar is unpredictable because the light can go on at any time, and an outside agent may intervene at any time moving blocks in the work area, either helping or hindering Robo-Soar's efforts to align the blocks. There is also uncertainty in Robo-Soar's perception of environment because the robot arm occludes the vision system while a block is being grasped. Until the arm is removed from the work area, there is no feedback as to whether a block has been successfully picked up.

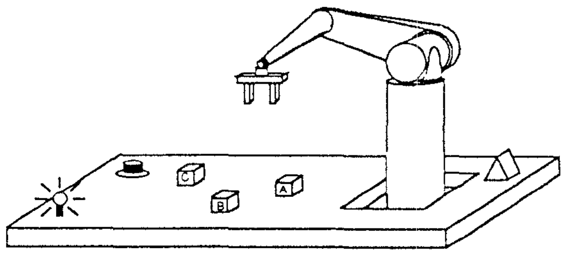


Figure 1: Robo-Soar

The second system, called *Hero-Soar*, controls a Hero 2000 robot, as shown in Figure 2. The Hero 2000 is a mobile robot with an arm for picking up objects and sonar sensors for detecting objects in the environment. Hero-Soar's task is to pick up cups and deposit them in a waste basket. Our initial demonstrations of Soar will use Robo-Soar. Only at the end of the paper will we

[*]This research was sponsored by grant NCC2-517 from NASA Ames and ONR grant N00014-88-K-0554.

[1]The systems described in this paper were implemented in Soar 5. For a complete description of Soar 5, see Laird et al. (1990) .

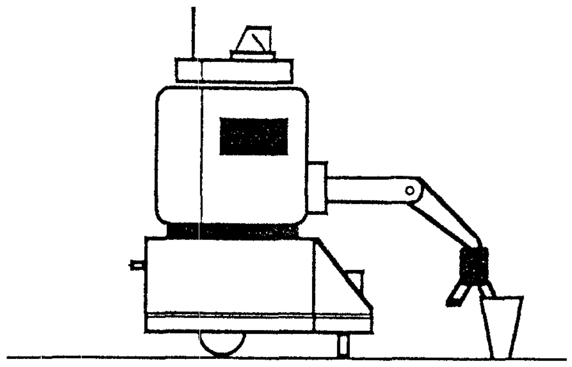


Figure 2: Hero-Soar

return to Hero-Soar, and at that time we will describe it more fully.

In presenting Soar, we first review its basic processing structure: the way it encodes knowledge and uses its knowledge to select and apply operators. Robo-Soar serves as an illustrative example. These details are necessary to set up Soar's approach to planning, which depends on the representation of long-term knowledge as productions and the run-time integration of short-term knowledge to selecting and applying operators. By necessity, some details of Soar's operation are skipped. See Laird et al. (1990) for a complete description of the Soar architecture.

## Execution

In Soar, all deliberate activity takes place within the context of goals or subgoals. A goal (or subgoal) is attempted by selecting and applying operators to transform an initial state into intermediate states until a desired state of the goal is reached. For Robo-Soar, one goal that arises is to align the blocks in the work area. The first decision to be made in a goal is the selection of a problem space. The problem space determines the set of available operators that can be used for a goal. In Robo-Soar, the problem space consists of operators such as `open-gripper` and `move-gripper`.

The second decision to be made is the selection of the initial state of the problem space. For goals involving interaction with an external environment, this is essentially given by the sensors. In Robo-Soar, the states consist of current and past data from Robo-Soar's sensors, as well as internally computed elaborations of this data, such as hypotheses about the positions of occluded blocks. Once the initial state is selected, decisions are made to select operators, one after another, until the goal is achieved.

Every decision made by Soar, be it to select a problem space, initial state, or operator for a goal, is based on *preferences* retrieved from its long-term memory. A preference is an absolute or relative statement of the worth of a specific object for a specific decision. The simplest preference, called *acceptable,* means that an object should be considered for a decision. Other preferences help distinguish between the acceptable objects. For example, a preference in Robo-Soar might be that it is better to select `close-gripper` than `move-gripper` for a given goal.

A preference is only considered for a decision if it has been retrieved from the long-term production memory. Productions are continually matched against the working memory. The working memory includes all active goals and their associated problem space, state, and operators. For example, a production in Robo-Soar that proposes the `close-gripper` operator might be:

```
If the gripper is open and surrounds a block
   then create an acceptable preference
        for close-gripper.
```

Soar's production memory is unusual in that it fires all matched production instantiations in parallel, and it *retracts* the actions of production instantiations that no longer match as in a JTMS. Thus, preferences and working memory elements exist only when they are relevant to the current situation. Decisions for selecting the problem space, the initial state and operators are made when Soar's production memory reaches quiescence, that is, when there are no new changes to working memory.

Once an operator is selected, productions sensitive to that operator can fire to implement the operator's actions. Operator implementation productions do not retract their actions when they no longer match. By nature they make changes to the state that must persist until explicitly changed by other operators. For an internal operator, the productions modify the current state. For an operator involving interaction with an external environment, the production augments the current state with appropriate motor commands. The Soar architecture picks up these augmentations and sends them directly to the robot controller. For both internal and external operators, there must be an additional production that tests to see that the operator was successfully applied. For external operators, feedback from sensors is tested to insure that an operator was actually carried out. This production creates a special preference that signals that the operator has terminated and that a new operator can be selected.

At this point, the basic execution level of Soar has been defined and it provides the basis for examining the integration of planning and execution. This basic execution level will be expanded later, to include both more reflexive and more deliberate execution. To summarize, execution consists of the selection and application of operators. Selections are based on the integration at run-time of preferences created by productions. It is these selections that are the basic control acts for which planning can provide additional knowledge.

## Planning

For situations in which Soar has sufficient control knowledge, the preferences created for each operator decision will lead to the selection of a single operator. However, for many decisions, the control knowledge encoded as productions will be incomplete or inconsistent, creating preferences that do not suggest a unique best choice. We call the situation when a decision is underdetermined an *impasse*. The Soar architecture detects impasses and automatically creates subgoals to determine the best choice. Within a subgoal, Soar once again casts the problem within a problem space, but this time the goal is to determine which operator to select, not to solve the original problem.

To determine the best operator, any number of methods can be used, such as drawing analogies to previous problems, asking an outside agent, or planning. At a minimum, planning requires additional knowledge, specifically, the ability to simulate the actions of external operators on the internal model of the world. As expected, this knowledge is encoded as productions that directly modify the internal state when an operator is selected to apply to an internal state. Additional planning strategies can be implemented through the addition of planning problem spaces with associated operators and control knowledge.

In Robo-Soar, an internal model of the environment is created and then searched to evaluate the alternative operators using any available control knowledge. The result of the search is the determination of the relative worth of the various alternative operators. These determinations are translated into preferences. When sufficient preferences have been created to allow a single choice to be made, the subgoal is automatically terminated and the appropriate selection is made.

This is the simplest example of planning in Soar, where a planning episode is used to resolve a single indecision. The planning can use both domain-independent strategies as well as domain-dependent control knowledge to restrict the search. The final plan is only a single step plan, consisting of the preferences that were created for the operators participating in the impasse. Even this simple example illustrates a unique property of planning in Soar; planning is invoked automatically as a response to insufficient knowledge; planning uses the same architecture and shares short-term and long-term knowledge with the execution system.

If this is the only point of indecision on the path to the goal, then there is no need to create a longer term plan. However, the typical case is that many of the decisions required to solve a problem are underdetermined and a more complete plan is required. If other decisions are underdetermined, then they will also lead to impasses and associated subgoals. The result is a recursive application of the planning strategy to each decision in the search where the current knowledge is insufficient. When a solution is found, one step plans of preferences are created in the subgoals for each of these decisions. Unfortunately, these preferences cannot directly serve as a plan because they are associated with specific subgoals, and they are removed from working memory when their associated subgoals are terminated.

At this point, Soar's learning mechanism, called *chunking*, comes into play to preserve the control knowledge that was produced in the subgoals. Chunking is based on the observation that: (1) an impasse arises because of lack of directly available knowledge, and (2) problem solving in the associated subgoal produces new information that is available to resolve the impasse. Chunking caches the processing of the subgoal by creating a production whose actions recreate the results of the subgoal. The conditions of the production are based on those working-memory elements that were tested by productions in the subgoal and found necessary to produce the results.

When chunking is used in conjunction with the planning scheme described above, new productions are learned that create preferences for operators. Since the problem solving in the subgoal originally created preferences based on searching for a solution to the goal, the productions include all of the relevant tests of the current situation that are necessary to achieve the goal. Chunks are learned not only for the original operator decision, but also for each decision that had an impasse in a subgoal. As a result, productions are learned that create sufficient preferences for making each decision along the path to the goal. Once the original impasse is resolved, the productions learned during planning will apply, creating sufficient preferences to select each operator on the path to the goal.

The ramifications of this approach are as follows:

1. **Planning without plans.**
In classical planning, the plan is the data structure that provides communication between the planner and the execution module. In Soar, an explicit plan is not created, but instead a sequence of control productions are learned to direct execution. These control productions not only suggest the appropriate operator to select, they can also suggest that an operator be avoided.

2. **On-demand planning.**
Soar invokes planning whenever knowledge is insufficient for making a decision and it terminates planning as soon as sufficient knowledge is found. Because of this, planning is always in service of execution.

3. **Learning improves execution and planning.**
Once a control production is learned, it can be used for future problems that match its conditions. These productions improve both execution and planning by eliminating indecision in both external and internal problem solving.

4. **Run-time combination of multiple plans.**
When a new situation is encountered, all relevant pro-

ductions will fire. It makes no difference in which previous problem this productions were learned. For a novel problem, it is possible to have productions from many different plans contribute to the selection of operators on the solution path.

It is this last observation that is probably most important for planning in uncertain and unpredictable environment. By not committing to a single plan, but instead allowing all cached planning knowledge to be combined at run-time, Soar can respond to unexpected changes in the environment, as long as it has previously encountered the situation. If it does not have sufficient knowledge for the current situation, it will plan, learn the appropriate knowledge, and in the future be able to respond directly without planning.

## Interruption

The emphasis in our prior description of planning, was on acquiring execution knowledge that could be responsive to changes in the environment. This ignores the issue of how the system responds to changes in its environment *while* it is planning. Consider two scenarios from Robo-Soar. In the first scenario, one of blocks is removed from the table while Robo-Soar is planning how to align the blocks. In the second, a light goes on while Robo-Soar is planning how to align the blocks. This light signals that Robo-Soar must push a button *as soon as possible.* The key to both of these scenarios, is that Soar's productions are continually matched against working memory including incoming sensor data and all goals and subgoals. When a change is detected, planning can be revised or abandoned if necessary.

In the first example, the removal of the block does not eliminate the necessity to plan, it just changes the current state, the desired state (fewer blocks need to be aligned) and the set of available operators (fewer blocks can be moved). The change in the set of available operators modifies the impasse but does not eliminate the need to plan. Within the subgoal, operators and data that was specific to moved block will be automatically retracted from working memory. The exact effect will depend on the state of the planning and its dependence on the eliminated block. In the case where an outside agent suddenly aligned all but one of the blocks, and Robo-Soar had sufficient knowledge for that case, the impasse would be eliminated and the appropriate operator selected.

In the second example, we assume that there exist a production in long-term knowledge whose purpose is to direct the Robo-Soar to push a button when a light is turned on. This production will test for the light and create a preference that the `push-button` operator *must* be selected. When the next decision is made for the operator, there is no longer a tie, and the `push-button` operator is selected. One disadvantage of this scheme is that it eliminates any subgoals from working memory, so that any partial planning that has not been captured in chunks will be lost.

Interruption of planning can be predicated on a variety stimuli. For example, productions can keep track of the time spent planning and abort the planning by creating preferences to select some action, if some action is better than none.

## Hierarchical Planning and Execution

In our previous examples of Robo-Soar, the set of operators corresponded quite closely to the motor commands of the robot controller. However, Soar has non restriction that problem space operators must directly correspond to individual actions of the motor system. For many problems, planning is greatly simplified if it is performed with abstract operators far removed from the primitive actions of the hardware. For execution, the hierarchical decomposition provided by multiple levels of operators can provide important context for dealing with execution errors and unexpected changes in the environment.

Soar provides hierarchical decomposition by creating subgoals whenever an operator there is insufficient knowledge encoded as productions to implement it directly. In the subgoal, the implementation of the operator is carried out by selecting and applying operators, until the operator is terminated.

To demonstrate Soar's capabilities in hierarchical planning and execution we will use our second system, Hero-Soar. Hero-Soar searches for the cups using sonar sensors. The basic motor commands include positioning the various parts of the arm, opening and closing the gripper, orienting sonar sensors, and moving and turning the robot. A more useful set includes operators such as `search-for-object`, `center-object`, `pickup-cup`, and `drop-cup`. The execution of each of these operators involves a combination of more primitive operators that can only be determined at run-time. For example, `search-for-an-object` involves an exploration of the room until the sonar sensors detect an object.

In Hero-Soar, the problem space for the top-most goal consists of just these operators. Control knowledge, provided either manually or acquired through planning, can select the operators when appropriate. However, once one of these operators is selected, an impasse arises because there are no relevant implementation productions. For example, once the `search-for-object` operator is selected, a subgoal is generated and a problem space is selected that contains operators for moving the robot and combining sonar readings.

Operators such as `search-for-object` would be considered a goal in most other systems. In contrast, goals in Soar arise only when knowledge is insufficient to make progress. One advantage of Soar's more uniform approach is that all the decision making and planning methods also apply to these "goals" (abstract operators). For example, if there is an abstract internal simulation of an operator such as pickup-cup, it can be used in planning for the top-goal in the same way planning would be performed at more primitive levels, and control knowledge can select between competing abstract operators.

A second advantage of treating operator as goals is that it even seemingly primitive acts, such as move-arm can become goals, providing hierarchical execution. This is especially important when there is uncertainty as to whether a primitive action will complete successfully. Hero-Soar has exactly these characteristics because it can lose motor commands and sensor data between the Hero and the controlling workstation. Hero-Soar handles this uncertainty by selecting an operator, such as move-arm, and then waits for feedback that the arm is in the correct position before terminating the operator. While the command is executing on the Hero hardware, a subgoal is created. In this subgoal, the primary operator is wait which adds one to a counter and terminates itself. It is then reselected, continually counting how long it is waiting. If appropriate feedback is received from the Hero, the move-arm operator terminates, a new operator is selected, and the subgoal is removed. However, if the motor command or feedback was lost, or there is some other problem, such as an obstruction preventing completion of the operator, the waiting continues. Productions sensitive to the selected operator and the current count detect when the operator has taken too long. These productions propose operators that directly query the feedback sensors, retry the operator, or attempt some other recovery strategy. Because of the relative computational speed differences between the Hero and Soar on an Explorer II+, Hero-Soar spends approximately 30% of its time waiting for its external actions to complete.

## Reactive Execution

Hierarchical execution provides important context for complex activities. Unfortunately it also exacts a cost in terms of run-time efficiency. In order to perform a primitive act, impasses must be detected, goals created, problem spaces selected, and so on, until the motor command is generated. Selecting and applying operators directly is more efficient, but has its own overheads. The actions of an operator will only be executed after the operator has been selected. Operators are selected only upon quiescence, thus forcing a delay. The advantage of these two approaches is that they allow knowledge to be integrated at run-time, so that a decision is not based on an isolated production. This allows Soar to recover from incorrect knowledge through learning (Laird, 1988) .

Soar also supports direct reflex actions where a production creates motor commands without testing the current operator. These productions act as reflexes for low level responses, such as stopping the wheel motors when an object directly in front of the robot. Along with the increase responsiveness comes a loss of control; no other knowledge will contribute to the decision to stop the robot.

The ultimate limits on reactivity rests with Soar's ability to match productions and process preferences. Unfortunately, there are currently no fixed time bounds for Soar's responsiveness. Given Soar's learning, an even greater concern is that extended planning and learning will actually reduce responsiveness as more and more productions must be matched (Tambe and Newell, 1988) . Recent results suggest that these problems can be avoided by restricting the expressiveness of the production conditions (Tambe and Rosenbloom, 1989) .

Although there are no time bounds, Soar is well matched for both Hero-Soar and Robo-Soar. In neither case does Soar's processing provide the main bottleneck. However, as we move into domains with more limited time constraints, further research on bounding Soar's execution time will be necessary.

## Acknowledgments

The author would like to thank Michael Hucka, Eric Yager, Chris Tuck, and Clare Congdon for help in developing Robo-Soar and Hero-Soar.

## References

[1] J. E. Laird. Recovery from incorrect knowledge in Soar. In *Proceedings of the AAAI-88*, August 1988.

[2] J. E. Laird, K. Swedlow, E. Altmann, and C. B. Congdon. *Soar 5 User's Manual.* University of Michigan, 1990. In preparation.

[3] J.E. Laird, E.S. Yager, C.M. Tuck, and M. Hucka. Learning in tele-autonomous systems using Soar. In *Proceedings of the 1989 NASA Conference on Space Telerobotics*, 1989. In press.

[4] M. Tambe and A. Newell. Some chunks are expensive. In *Proceedings of the Fifth International Conference on Machine Learning*, 1988.

[5] M. Tambe and P. S. Rosenbloom. Eliminating expensive chunks by restricting expressiveness. In *Proceedings of IJCAI-89*, 1989.

# Autonomous Prediction and Reaction with Dynamic Deadlines

**Richard Levinson**
Recom Technologies Inc.
NASA Ames Research Center
Mail Stop: 244-17
Moffett Field, CA 94035

## Abstract

We describe a distributed planning architecture that allows an autonomous agent to dynamically balance planning time vs. execution time according to changing execution time deadlines. Our approach is being applied toward a NASA project for autonomous soil analysis. This paper introduces the concepts and terminology of our planning system. A high level description of key underlying mechanisms is then illustrated using a simplified formal example from our soil analysis domain.

## 1 Basic Terminology in Our Approach

We are modeling our domain using a multiple agent framework. In our domain, an AGENT is an entity that independently modifies data in the external world. The EXTERNAL WORLD is a collection of data, called EXTERNAL DATA, accessible to all agents. Each agent possesses a different set of perceptors which interpret the external data in ways useful to that agent. Our agent continuously reasons with and updates a qualitative model of the external world, called the WORLD MODEL. The world model is a collection of facts that represent our agent's perception of the world. These facts are either raw sense data or inferences from that data, and may express quantitative or qualitative information. Each agent requires external data for input, and modifies external data as output. We call this input-output cycle the RESPONSE CYCLE. We define a HYPOTHETICAL WORLD to be a copy of the world model that can be modified by the agent, without modifying the actual world model or external data. We define PREDICTION to mean creating any model of the future. We define PROJECTION to mean a type of prediction that creates and analyzes hypothetical worlds. Our agent has a 3 component response cycle:

1. INTERPRETATION = External data input and transformation.
2. PLANNING = Using prediction to synthesize behaviors which modify external data.
3. EXECUTION = Execution of behaviors which modify external data.

The real-time requirements of our domain demand that the total response cycle time be tractable and bounded. This crucial parameter is defined as : RESPONSE TIME = INTERPRETATION TIME + PLANNING TIME + EXECUTION TIME

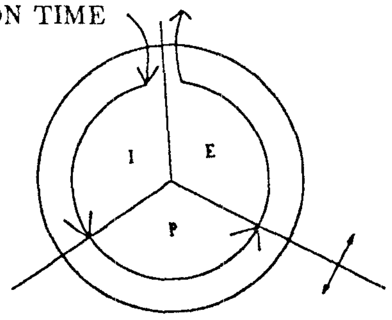


Figure 1: Our agent's response cycle (R=I+P+E)

Due to the unpredictable nature of our target environment, the proper proportions of each component in the response cycle cannot be fixed at system design time. The partition between each component in the response cycle can be seen as a sliding wall. By sliding a partition in either direction, the "percentage of the pie" is changed. It is probably necessary to vary all 3 partitions dynamically in an autonomous system. However, we are focusing on the Planning/Execution ratio in order to study dynamic adjustment on the predictive/reactive spectrum. Our system will provide the flexibility of automatically adjusting the P/E RATIO according to unpredictable changes in the external world. We call this a DYNAMIC RESPONSE TIME RATIO.

Our system will handle DYNAMIC P/E RATIOs in the following way: We can associate each subgoal with a deadline describing when that subgoal must be executed. This deadline can either be an absolute time in the real world, or an offset from the starting time of the task. We can state what percentage of task execution time should be used to achieve each subgoal, essentially describing how time is to be distributed within the task body. This information can then be used to determine if the tasks are running according to schedule. When running ahead of schedule we increase P/E, when running behind we decrease P/E. Due to these dynamic time constraints, our system may need to begin execution before planning is complete. Faced with the need to act immediately, but having only an incomplete plan, our agent must start executing the beginning of a plan while continuing to reason hypothetically about the rest of that plan. This requires that our plan synthesis algorithms possess significant anytime properties in the sense of Dean and Boddy[DB88]. To address this requirement, we focus on the question of when decisions are made.

We define DECISION PHASES as the time intervals when decisions are made. A DECISION to us primarily means selecting actions, variable bindings, and action orderings. Our system is particularly concerned with runtime projection because our aim is to eventually handle highly procedural actions with conditional effects. We identify 3 decision phases for autonomous behavior in an agent: PRE-RUN PREDICTION: The decision is made by a human designer or an off-line computation which models the future before the agent has executed any behavior at all (before the agent is "alive"). During this phase, external data are static or uninstantiated. RUNTIME PREDICTION: The agent itself makes the decision by modeling the future while instantiated external data are changing. These decisions are part of the planning component of our response cycle. EXECUTION: The agent makes the decision by modifying external data without consulting any model of future. The autonomy and uncertainty in our domain requires the flexibility of using all 3 decision phases. Our system will provide coverage in all phases, although we are primarily interested in studying decision making during the runtime prediction phase. Many of the new reactive systems combine pre-run prediction with execution, while the classical planning systems have minimal execution phase competence. To study the mechanics of our system further, we introduce a simplified domain example.

## 2 The Domain Description

Our project is concerned with control of a Soil Lab Experiment Director (SLED) which can operate autonomously in an uncertain and unpredictable environment. SLED will provide the interpretation, planning and execution components for controlling the mineral analysis equipment on a planetary rover. SLED designs and coordinates experiments which use multiple modules of analysis hardware to describe an unknown mineral compound. Although the compound may be unknown on Earth, SLED will design experiments which attempt to disambiguate between possible descriptions. SLED will contend with the unpredictable behavior of the following agents:
*ROVER* - The rover executive in which SLED is embedded. This is SLED's boss and it dynamically controls SLED's time, power and goal requirements.
*DTA* - The Differential Thermal Analyzer. Analysis hardware which can detect phase changes in a soil sample as it is heated according to a "heating program". This agent can malfunction unpredictably.
*GC* - The Gas Chromatograph. Analysis hardware which can sniff gas that evolves during the heating process of the DTA (above). This agent can malfunction unpredictably.
*SOIL* - The unknown soil. This agent will always act unpredictably. For example, if the soil heats up faster or slower than expected then it affects time constraints. We may design an experiment assuming the presence of a mineral which is actually not present. If there is much less of some mineral than expected, the DTA heating program may proceed too fast to notice. We are primarily interested in 2 sets of temperatures:
*reference-temperatures* = [0...100]
*soil-temperatures* = the set $[<, >, =]$ where
$>\equiv$ soil temperature is hotter than reference.
$<\equiv$ soil temperature is cooler than reference.
$=\equiv$ soil temperature is equal to reference.

Facts in our example database are represented as predicates.The sensory facts in our database are : *(reference-temp x)*: x $\in$ reference-temperatures. This reference-temp parameter is given as an index into the other three predicates. *(soil-temp x y)* : x $\in$ reference-temperatures, y $\in$ soil-temperatures. *(gc-status x y)* : x $\in$ reference-temperatures, y $\in$ [on, off]. *(oven-speed x y)* : x $\in$ reference-temperatures, y $\in$ [High, Medium, Low, Neutral]. The interpretation component supplies several reference-temp vs. soil-temp curves for previously analyzed minerals which may be present in soil. Each curve is represented as a stream of <soil-temp, reference-temp> coordinate pairs. EXPECTED-CURVES = a set of previously recorded curves for known soil-temps vs. reference-temps.

In addition to the sensory facts, we have facts which are maintained internally within our agent. These memory facts in our database are: *(expected-soil-temp x y z)* : x $\in$ expected-curves, y $\in$ reference-temperatures, z $\in$ soil-temperatures. The interpretation system also supplies abstract descriptions of the expected-curves. Our abstract view of a curve is: *(expected-curve-description x y)* : x $\in$ expected-curves, y $\equiv$ a list of thermal events. A *thermal event* describes an interval of exothermic (>) or an endothermic (<) difference between the soil temperature and the reference temperature. A thermal event, e, is a triple < duration, direction, onset-temp> : duration $\in$ [small, medium, large], direction $\in [<, >, =]$, onset-temp $\in$ reference-temperatures. For e = the event <dur, dir, onset> we have the facts *(expected-event-duration e y)* : y = dur, *(expected-event-direction e y)* : y = dir, *(expected-event-onset-temp e y)* : y = onset. A STATE $\equiv$ w, x, y, z : (reference-temp w)$\wedge$(soil-temp w x)$\wedge$(gc-status w y)$\wedge$(oven-speed w z). In other words, a state $\equiv$ a set of values for the four sensory facts.

## 3 An Example Mineral Analysis Planning Problem

The goal of our simplified DTA-Driver example is to: Predict a sequence of states which burn the soil as fast as possible while slowing down for higher resolution at expected critical recording temperatures. Examples of critical recording temperatures are: 1) short duration thermal events and 2)differentiating between similar expected curves. While trying to predict the road ahead, the DTA-driver will also be reacting to real-time sensory evidence which contradicts expected-curve evidence.

How to predict the state sequences? For each reference-temp from 0 to 1000, our agent must predict 3 values:

Predict x : (soil-temp reference-temp x)

- Choose x's value from $[<,>,=]$
- Can use a heuristic H1 based on w: (expected-soil-temp c reference-temp w): ∀ c ∈ expected-curves.
- The DTA-driver (our autonomous agent) cannot control this value.

Predict y : (gc-status reference-temp y)

- Choose y's value from [ON, OFF]
- 2 actions allow the DTA-driver to deterministically control this value based on expected or sensed values of (soil-temp reference-temp x)

Predict z : (oven-speed reference-temp z)

- Choose z's value from [HIGH, MEDIUM, LOW, NEUTRAL], which correspond to heating rates.
- 4 actions allow the DTA-driver to heuristically control this value.
- Can use a heuristic H2 based on y: (expected-event-duration x y) ∀ x ∈ events occurring at reference-temp, ∀ curve-descriptions, ∀ expected-curves.

Here are the actions in our domain. The format is: *preconditions → action-name → postconditions.* The DTA must be in neutral (isothermal) before changing speeds.

1. (soil-temp reference-temp >) → GC-ON → (gc-status ON)
2. ¬(soil-temp reference-temp >) → GC-OFF → (gc-status OFF)
3. (oven-speed reference-temp NEUTRAL) → SPEED-H → (oven-speed reference-temp HIGH)
4. (oven-speed reference-temp NEUTRAL) → SPEED-M → (oven-speed reference-temp MEDIUM)
5. (oven-speed reference-temp NEUTRAL) → SPEED-L → (oven-speed reference-temp LOW)
6. ¬(oven-speed reference-temp Neutral) → PAUSE → (oven-speed reference-temp NEUTRAL)

The Time constraints in this example are:

1. The agent must fit execution time within a given execution time slot. The agent's expectations and reactions can each lengthen execution time.
2. The agent must fit planning time within a changing execution time interval. The interval is the time until the next program step is needed. The interval length depends on speed: Time = Temperature-change/Heating-rate. The agent's expectations and reactions can each lengthen planning time.
3. The agent must have the GC available and warmed up to catch exothermic (>) thermal events. The agent never really knows how long until the next exothermic event.

## 4 The Situated Planning Society Architecture

SLED will be developed using the EXECUTIVE TOOL KIT (XTK). XTK is a rapid-prototyping tool for developing models of executive behavior in distributed autonomous systems. It is a procedurally expressive multitasking reasoning system originally derived from the STRIPS-based KEE backward chainer. The basic object in XTK is a TASK. XTK tasks are goal-driven symbolic procedures which allow subgoaling. Some tasks in a domain are run as parallel processes (co-routines) while others are called as subroutines. This system and PRS (GE87)share much of the same motivation. However, XTK is aimed at an even higher level of procedural expressiveness than PRS, combined with projection.

In XTK, decisions appear as choice points. A CHOICE POINT is a non-deterministic subgoal expression in XTK. A statement of the form (ACHIEVE <subgoal> USING <supervisor>) can be used if a choice must be made between several subgoal achievement methods. When a task executes this type of subgoal expression, we have "encountered a choice point". In this situation, many planners focus on one choice until it either succeeds or fails, before they try another approach. This is a rather coarse update quantum if we really want our plan to be ready anytime. XTK handles choice points explicitly by providing high level options for controlling search heuristically from these points. Our approach is to get better anytime performance by associating each choice point with softer, heuristic meta goals called SCORING CONDITIONS. These scoring conditions are monitored by a supervisor task, so that decisions can be made continuously throughout the projection of future worlds (runtime projection)– not only at the end. These conditions are predicates designed during pre-run prediction and can include Lisp functions. The scoring conditions refer to changing facts in a hypothetical world model, and they capture domain heuristics for local choice selection.

The planning mechanisms in XTK are embedded within the execution system. Situated, goal-driven behavior is performed by a team of tasks; some of whom perform execution activities while others perform planning activities. Our team consists of REAL WORLD TASKS, HYPOTHETICAL TASKS, SUPERVISORS and EXECUTION MONITORS. A REAL WORLD TASK (○) is the basic task used to model domain actions. The basic XTK scenario is: A real world task is modifying external data (i.e. executing - defined in Section 1), when it encounters a choice point. If the choice cannot be made using less expensive methods (due to uncertainty), then the appropriate choice is selected by simulating the alternatives in hypothetical worlds using hypothetical tasks (RUNTIME PROJECTION). A HYPOTHETICAL TASK (□) is identical to a real world task except that it is running in a hypothetical world - thus simulating the action not executing it. Each leaf in a hypothetical task tree represents a potential sequence of plan decisions, thus each leaf identifies a unique PLAN. A task called a SUPERVISOR (△) is created at each choice point, in order to monitor the scoring conditions that are modified by subordinate hypothetical tasks. The job of a supervisor is to maintain a preference ordering on the plans below it by ranking each plan when it changes with respect to a scoring condition. Additionally, the supervisors will consider the cost and time deadlines in their sorting functions. Whenever a real world task encounters a choice point, an EXECUTION MONITOR (◇) is created as the interface expert between the tasks executing in the real world and the hypothetical tasks which simulate decision alternatives. Execution monitors control the scheduling of hypothetical tasks and decide when planning time runs out. They must then collect the highest scoring hypothetical task tree, and monitor the execution of that partial plan in the real world. It may even need to simultaneously coordinate the efforts of planning tasks with those of execution tasks. Here is a sample fragment of the task tree for our example:

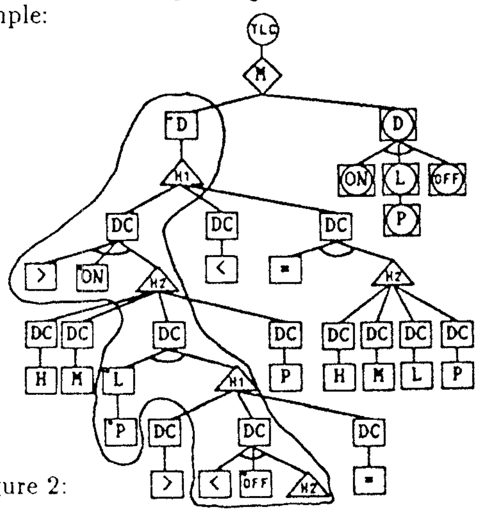


Figure 2:

The Top Level Controller (TLC) is executing in the real world when it encounters the goal : (a DTA program considering expected-curves is ?program). The deadline for that goal permits 5 minutes of planning time before we run the experiment. The execution monitor, M, is created as a subtask of TLC in order to interface between tasks in hypothetical worlds and the real world task TLC. The responsibility of the interface includes first controlling the transition from real world execution into planning, and then most importantly, controlling the transition from planning into real world execution. This means coordinating the communication between the simulation of plans on the left and the execution of plans on the right. After M is created, the dta-driver, D, is run in a hypothetical world. It encounters the choice point : (ACHIEVE (soil-temp reference-temp soil-temp) USING H1). H1 is the name of a supervisor whose scoring conditions use Heuristic-1, based on expected and sensed values of x: (soil-temp reference-temp x)–see section 3. Three subtasks suggest the alternative values >, < or =. The Driver,D, splits into 3 different CPU processes to model 3 different futures based on each possible value. Since the driver,D, split into 3 CPU processes, we copy the computational continuation of D into each choice. These Driver Continuations are labelled DC; they are effectively clones of the driver, D. The heuristics of an H1 supervisor prefer the > future so it is given more CPU. Since it is exothermic (>), the GC is turned on. We then hit another choice point: (ACHIEVE (oven-speed reference-temp speed) USING H2).The DC now splits 4 ways to model the alternative futures–HIGH, MEDIUM, LOW or NEUTRAL. By monitoring facts in their scoring conditions, the H1 and H2 supervisors each know when the leaves below them change . These 2 supervisors rank the plans below them according to different heuristics. They each send their recommendations to M who sorts those local suggestions based on more global information, like changes in the external world and slips in execution time. M maintains an ordering on the alternative sequences of plan decisions below it. When we hit the H2 choice in the bottom row, planning time runs out. The highest ranking plan at that point is selected (encircled subtree). These tasks are then executed by the monitor (shown in circles and squares on the right).

## 5 The Dynamic P/E Balancing Mechanism

We define PLANNING TIME as the time left over when running ahead of our execution deadlines. Our hook into time is the Deadline option on the ACHIEVE statement: (ACHIEVE <subgoal> :DEADLINE <Time V Offset >) This execution time deadline represents a pointer to the time when subgoal execution must begin. The deadline is designed during pre-run prediction or could be probabilistically learned based on previous subtask duration times. This is the time when planning must stop, a choice is selected, and execution begins. When real-world tasks achieve subgoals, they calculate the closeness of their time constraints : *Planning time* ≡ *subgoal-execution-deadline - current-time.* Positive planning time indicates we are ahead of schedule, and can spend more time thinking. Negative planning time indicates we are behind schedule, and must spend more time doing. We currently have 3 mechanisms based on this deadline information:

1. When a task encounters a choice point, it passes the planning time to an execution monitor who counts down till the execution deadline. When the deadline is up, the monitor selects the best partial plan at that time, and starts concurrent planning/execution. A countdown is initiated this way (a) when a choice point is encountered for the first time or (b) when a previous choice point must be replanned due to new sensor evidence (reaction).
2. As subgoals are are achieved in the real world, the execution tasks pass their planning time calculations to their monitor, allowing the monitor to keep abreast of slips in the execution schedule. Positive planning time is added to the monitor's current planning time countdown, and negative planning time is subtracted from the countdown.
3. The subgoal deadline is used by supervisors to compare expected subtask duration with subgoal deadlines while scoring alternative subtasks. This allows the planners to reason about subtasks which may not make their subgoal deadlines.

## 6 Search Control & Communication in The Situated Planning Society

XTK provides a distributed planning approach with search controlled by a combination of local and global decision experts. Each plan actually represents a particular sequence of plan decisions, where each decision is monitored by a different supervisor. The supervisors each sort their subordinate plans based on their own heuristic preference. As a result, supervisors at different levels in the task tree may have different scores for a single plan. The supervisors each send a preferred subset of choices to an execution monitor task, who sorts the suggestions a second time based on the percentage of supervisors that recommended each one. The most popular plans get scheduled for CPU time, allowing them to be further simulated. This provides a parallel-beam search control mechanism.

When the monitor notices that the execution states of the tasks on right differ significantly from the assumptions of the planning tasks on the left, then M adjusts the weight on the local heuristic supervisors. Suggestions from supervisors based on currently true states are weighted more heavily than suggestions based on false states. The local rankings occur independently of actual subgoal success or failure. Instead, the rankings change whenever relevant facts change in the simulated states. This allows each supervisor to independently change its local ranking at any time. Likewise, the global sort changes asynchronously due to continuous sensor information. We call this "anytime" because the plans represent *continuously* sorted chronological sequences of plan decisions. Any time we run out of planning time, we have a locally and globally elected sequence of decisions to execute starting from the initial state. While the beginning of the plan is executing on the right, planning continues on the left. The plans get better as time progresses because they look further into the future; becoming more complete and globally coherent.

The execution monitor provides communication between concurrent planning and execution tasks about "mutually interesting results". The Execution Monitor watches for several types of events, then passes information between planning subtasks (P) and execution subtasks (E). First,the monitor watches for slips in the execution time. When the schedule slips forward or backward, messages are sent from the E tasks to the P tasks. Secondly, the preconditions which support the tasks in a selected plan are monitored as that plan is executed by E tasks. If those planned preconditions are not true during execution time, messages are sent from the E tasks to the P tasks in order to initiate replanning (time permiting). Additionally, these messages allow the planning tasks to prune their search space according to execution choices being made in the real world. As the planning tasks incrementally decide on upcoming choice points, messages are sent in the P to E direction. These messages may involve sending entire planned subtrees incrementally to an execution monitor who then integrates the tree with it's execution tasks. Alternatively, the planning tasks could send Drummond's Situated Control Rules (DR89) to the execution monitor for a more reactive response.

The above approach to planning significantly breaks down the usual dichotomy between planning and execution activities. Since planning and execution are both represented using tasks, the planning activity *is* an execution activity. The only difference is that planning activities reason about hypothetical procedures as opposed to reasoning about attributes in the physical world. They can communicate with each other about mutually interesting results, providing flexibility in the face of dynamic time constraints.

**Acknowledgements**

Thanks to Mark Drummond and Dave Thompson for their feedback during the evolution of this work.

## References

[1] Dean, T., Boddy, M. 1988. An Analysis of Time-Dependent Planning. *Proceedings from AAAI '88.* St. Paul MN.

[2] Georgeff, M., Lansky, A. 1987 Reactive Reasoning and Planning. *Proceedings from AAAI '87.* Los Angeles, CA.

[3] Drummond, M, 1989. Situated Control rules. *Proceedings of Conference on Principles of Knowledge Representation and Reasoning*, Toronto, Canada.

# Task Planning using a Formal Model for Reactive Robot Plans

**D.M. Lyons, R.N. Pelavin, A.J. Hendriks, D.P. Benjamin**

*Autonomous Systems Department*
*Philips Laboratories*
*North American Philips Corporation*
*345 Scarborough Road*
*Briarcliff Manor*
*NY 10510*
dml@philabs.philips.com

Keywords: $\mathcal{RS}$, Planning, Plan Representation, Reactive Planning, Opportunism, Plan Analysis.

## 1 Introduction.

Our work is focused on the automatic generation and verification of plans for uncertain and dynamic environments. An intelligent system cannot explicitly plan out all its actions in advance if it is operating in such an environment. For example, in Sanborn's Traffic World (Sanborn 1989), cars approach at unknown and dynamically changing velocities. The only way to deal with this problem is to construct *reactive plans*: plans that can tune their actions to suit the environment. Reactive plans can be constructed in advance, either manually, as in (Agre & Chapman 1987) Pengi or in the (Brook 1986) subsumption architecture, or automatically, given some knowledge about the uncertainty in the environment. Plan generation in dynamic domains involves identifying relevant (external) conditions to monitor and determining an appropriate control mechanism that links the sensor information to the effectors so that the agent can react properly to external events.

We have developed a model for representing reactive plans as networks of distributed processes, $\mathcal{RS}$ (Lyons & Arbib 1989; Lyons 1988,1989). In addition, this model can also be used to represent the environment in which the plan acts. We have developed formal techniques for analyzing plans and worlds expressed in the model. Our work is novel in that it offers (1) a precise framework for reactive plans, and (2) techniques for looking at the behavior of reactive plans in uncertain and dynamic environments.

## 2 The $\mathcal{RS}$ Model.

The $\mathcal{RS}$ (Robot Schemas) model was developed to express robot plans that could 'tune' their behavior to suit their environment. This is particularly relevant in the robot domain because object locations, identities and behaviors are never precisely known in any realistic task. We chose to build a special model of distributed computation for reasons of efficiency in execution, and because of the formal analysis techniques available (Hoare 1985; Hennessy 1988). A plan in $\mathcal{RS}$ is a network of concurrent processes. The processes can be coupled into networks in two ways: they can communicate messages to each other via *communication ports*, or they can be composed together using several kinds of *process composition operators.* Processes can be defined in terms of networks of other processes, grounding out with a set of primitive, pre-defined, processes.

## 3 Representing Reactive Plans.

The canonical structure for a plan in our model is the *task unit*, which written as a network schema of three concurrent, communicating processes:

$$\mathtt{TU} = [\mathtt{S}, \mathtt{T}, \mathtt{M}]^{c}$$

The process S implements the sensory activities for task TU, the process M implements the motor activities for task TU, and the process T implements the connection between sensing and motor activities that characterizes the task. The connection map $c$ describes the connections between the communication ports on the three processes. Thus, every plan is phrased as action linked to sensing. Additionally, the S component, which implements task-specific sensing, allows us to represent the world in *functional-indexical* terms. That is, objects are only represented in $\mathcal{RS}$ in terms of the role they play in task units.

To represent more explicitly *how* the T process implements the connections between sensing and action, we developed a set of process composition operators. For example, the enable operator ':' combines two processes sequentially but allows the first process to parameterize the second. The following network uses this operator to implement a plan to assemble a box from two sides, a top and a base, in a reactive fashion:

$$\begin{aligned}\texttt{Plan} = \ & \texttt{Locate}_{base}\langle p\rangle : \texttt{Place}_p;\\ & [\ \texttt{Locate}_{s1}\langle p\rangle : \texttt{Place}_p,\\ & \ \ \texttt{Locate}_{s2}\langle p\rangle : \texttt{Place}_p\ ];\\ & \texttt{Locate}_{top}\langle p\rangle : \texttt{Place}_p\end{aligned}$$

$\texttt{Locate}_m$ does a sensory search of the environment for an object of class $m$ and returns a pointer to it in $p$ when it is found and terminates. The value $p$ is then used to enable the next process Place. $\texttt{Place}_p$ places part $p$ in the assembly and terminates. The ':' operation links these two processes so that each part is searched for and then placed — it 'synchronizes' the Place process with the environment. The ';' operator implements sequential composition; it 'imposes' a strict ordering on some actions in the plan. However, the concurrency in searching for, and placing, the sides ($s1$ and $s2$) means that these operations will be done in whatever order the environment enables. This sort of reaction is sometimes called *opportunism* (Fox & Kempf 1985).

## 4 Representing Uncertain and Dynamic Environments.

The $\mathcal{RS}$ model can also be used to represent dynamic and uncertain worlds. The composition operators can be used to build temporal descriptions. The recurring enable composition operation, written ':;', produces a process that is like an enable composition in an infinite loop. Let Ran be a process that terminates in some random, finite time. Let the Car process represent a car. The network

$$\texttt{OneCarWorld} = \texttt{Ran} :; \texttt{Car}$$

repeatedly generates a Car process at a random time.

The transcondition operation, '#', allows us to use one process to limit the execution of another: it produces a process that acts like the concurrent network of both its argument until either argument terminates, then the composition terminates. In a clean room domain, wafer batches have to be heated in ovens to facilitate various chemical processes. Unfortunately, these ovens go down a lot. Let the process Oven represent the operating oven. Let the *termination* of IfError represent the event that causes the oven to go out of operation. Let OvenDown represent the oven being down. Finally, let the termination of IfRepaired represent the successful conclusion of the activities necessary to fix the oven. The following network is a causal model of this uncertain and dynamic world:

$$\texttt{CleanWorld} = [\texttt{Oven}\#\texttt{IfError}] :; [\texttt{IfRepaired}\#\texttt{OvenDown}]$$

This accurately represents what is know about the world (the causal structure) but doesn't say anything about when specifically, or how often, the oven will go down.

## 5 Plan Analysis.

The formal semantics of the $\mathcal{RS}$ model is constructed using *Port Automata* (Steenstrup et al. 1983), a special automata theoretic model for distributed computation. The process equality, composition operators and port communication are defined in terms of port automata. We use the methodology of *process algebra* (Hoare 1985) to analyze the behavior of plans in worlds. In addition to using the algebraic properties of the composition operators we have discussed, we have also developed a plan analysis operator, 'evolves,' defined as follows. We say that process A 'evolves' into process B under condition $\Omega$ if A becomes equal to B when condition $\Omega$ occurs; we write this as $\texttt{A} \xrightarrow{\Omega} \texttt{B}$. Let Stop be a special process that immediately terminates. Asking $[\texttt{Plan}, \texttt{World}] \xrightarrow{\Omega} \texttt{Stop}$ is equivalent to asking under what conditions will Plan terminate in environment World. The environment, World, is a network of models, such as CleanWorld or OneCarWorld above, that describes the environment in which the plan is to be carried out. We can also ask under what conditions some Goal network is produced, $[\texttt{Plan}, \texttt{World}] \xrightarrow{\Omega} \texttt{Goal}$. The evolves operator can be easily implemented as follows: The termination conditions for each primitive process must be specified. The definitions of the individual composition operators can be used to deduce how composite processes evolves.

## 6 Autogeneration of Reactive Plans.

We are developing a formalism that embraces both the school of reactive planning (Agre & Chapman 1987) and the hierarchical top-down planning methodology (Sacerdoti 1974; Fikes & Nilsson 1971). Like (Agre and Chapman 1978) we recognize the computational expense of using classical techniques to generate plans, and see the need for reactive mechanisms. However, we argue that it is always advantageous to have some *a priori* plan generation component, even in the case when the plan itself is highly reactive. Our objective is to develop a uniform framework that can handle a broad spectrum of tasks, ranging from short term reactions to long term strategic planning, based on the $\mathcal{RS}$ model. We summarize the keys features of our approach to plan generation as follows.

- In our framework, a planner produces an abstract plan in response to a specified goal and typically based on a incomplete view of the world. This abstract plan may contain actions at various levels of detail. The one important constraint we impose on this plan is that the first step must be executable. This enables the agent to undertake execution of this step, while concurrently either refining or updating the plan.

- To determine the appropriate steps to include in an abstract plan, it is important to determine which features in the external world are *relevant* features that the agent must monitor and respond to. Previous work has considered only manual plan construction; the user must determine the appropriate actions (Fikes & Nilsson 1971) and set of features to monitor (Sacerdoti 1974). In contrast, we are developing methods for automatically determining which features are relevant. In (Benjamin et al. 1989), we show how to automatically decompose a task described by a network into a hierarchy of subtasks, such that each level of the hierarchy has an associated feature identifying the aspect of the world to be monitored.

- To automate the process of coordinating processes, we are extending the concepts of (Pelavin 1988) to connect the ports of $\mathcal{RS}$ processes and to handle all of the $\mathcal{RS}$ process combination operators (temporal coordination). Meeting timing constraints is also an important issue. Work in progress on real-time scheduling in $\mathcal{RS}$ (Lyons & Mehta 1989; Pocock 1989) starts to address this issue.

- We plan to move away from "airtight planning" by incorporating decision theory and adapting our methods to work with probabilistic statements and statements about utilities. Decision theory provides a formal framework for choosing the plan that best achieves the agent's (possibly competing) objectives in an uncertain world. The use of decision theory for planning in uncertain worlds has been previously advocated by (Langlotz et al. 1986).

## 7 Implementation.

We have implemented an $\mathcal{RS}$ kernel and front-end on an inhouse multiprocessor and are using it for experiements in intelligent robot control (Lyons & Mehta 1989). We have simulated a subset of the "evolves" operator in PROLOG on a SUN, and are now building a first version of the plan generation software.

## References

[1] P. Agre and D. Chapman. Pengi: an implementation of a theory of action. In *Proc. AAAI-87*, Santa Cruz CA, Oct. 1987, pp.123–154.

[2] Allen, J.F. Towards a General Theory of Action and Time. In *Artificial Intelligence* 23,2, 1984, pp. 123–154

[3] Benjamin, D.P., Dorst, L., Mandhyan, I. and Rosar, M., An introduction to the decomposition of task representation in Autonomous Systems. In *Change of Representation and Inductive Bias* Kluwer Academic Publishers, 1989.

[4] R. Brooks. A robust layered control system for a mobile robot. *IEEE J. Rob. & Aut.*, RA-2(1):14–22, Mar. 1986.

[5] Fikes, R.E. and Nilsson, N.J. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving *Artificial Intelligence* 2, 1971, pp.189–208.

[6] B.R. Fox and K.G. Kempf. Opportunistic scheduling for robotic assembly. In *IEEE Int. Conf. Robotics & Automation*, pages 880–889, St.Louis MO, 1985.

[7] Langlotz, C., Fagan, L., Tu, S., Williams, J., and Sikic, B. ONXY: An Architecture for Planning in Uncertain Environments. *Proc. AAAI-86*, 1987 pp.447–449

[8] M. Hennessy. *Algebraic Theory of Processes.* MIT Press, 1988.

[9] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International Series in Computer Science, 1985.

[10] D.M. Lyons. On-line allocation of robot resources to task plans. In *SPIE Symp. on Advances in Intelligent Robotic Systems; Expert Robots for Industrial Use (Vol. 1008)*, pages 215–222, Nov. 1988.

[11] D.M. Lyons. A process-based approach to task-plan representation. In *Submitted to the IEEE Int. Conf. Robotics & Automation*, Cincinatti, Ohio, May 1990.

[12] D.M. Lyons and M.A. Arbib. A formal model of computation for sensory-based robotics. *IEEE Trans. on Robotics & Automation*, 5(3):280–293, June 1989.

[13] D.M. Lyons and S. Mehta. A distributed computing environment for the multiple robot domain. In *Fourth International Conference on CAD, CAM, Robotics and factories of the future*, New Delhi, India, Dec. 19–22 1989.

[14] McDermott, D., A Temporal Logic for Reasoning about Process and Plans. *Cognitive Science* 6,2 (1982), 101-155.

Pel88 Pelavin, R. N., A Formal Approach to Planning with Concurrent Actions and External Events. *PhD Thesis*, University of Rochester, 1988.

[15] Pocock, G. A Distributed Real-Time Programming Language for Robotics. *IEEE 1989 Int. Conf. on Rob. & Aut.* May 14–19th 1989, Scottsdale Arizona, pp. 1010–1015.

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

[17] J. Sanborn. Dynamic reaction: Controlling behavior in dynamic domains. Technical Report CS-TR-2184, University of Maryland, Dept. of Computer Science, College Park, MD 20742, 1989.

[18] M. Steenstrup, M.A. Arbib, and E.G. Manes. Port automata and the algebra of concurrent processes. *Journal of Computer and System Sciences*, 27(1):29–50, Aug. 1983.

# Reactive Planning using a "Situation Space"

Stacy C. Marsella and Charles F. Schmidt

Department of Computer Science
Rutgers University
New Brunswick, NJ

## Introduction

Consider the problem of how a single agent, the actor, can plan rationally in a world populated with multiple independent intelligent agents. Each agent can bring about changes in the actor's world which could frustrate or facilitate some course of action that the actor might entertain as part of a possible plan for achieving a goal. In the classical planning paradigm the actor is the sole source of change. This assumption, coupled with an assumption that the actor possesses a complete and correct model of the world, allows us to adopt a fairly straightforward definition of rationality for such a planning system. The system behaves rationally if it creates correct plans which are in a certain sense also minimal. A plan is correct if the goal can be proved to follow from the initial state and the plan. It satisfies a sense of minimality if no part of the plan can be removed and the proof remain valid. This is an intuitively reasonable criterion for rationality in this classical paradigm. However, research within this paradigm has led to a deeper understanding of the difficulties that stand in the way of creating an efficient computational model of planning that is guaranteed to satisfy this notion of rationality when the world is even of moderate complexity.

More recently, there has been a good deal of research in AI planning on the problem of how to extend the investigation of planning to contexts where the assumptions of the classical model are relaxed (e.g. Hendler and Sanborn, 1987; Firby, 1987; Schmidt et al., 1989). The relaxation ranges from withdrawing the assumption that guarantees success of action execution to the withdrawal of the assumption that the planner is the only source of change in the planning environment. Not surprisingly, the notion of rationality assumed in classical planning is difficult to realistically apply to most non-classical planning environments. If complete and certain knowledge is either lacking or of a complexity that precludes its effective use, then it may be the case that the success of any plan cannot be guaranteed.

In this paper we will tentatively put forward a hypothesis about a way in which to implicitly structure the space of states associated with a domain such that a weaker form of rationality is enforced when a planner uses this structure to guide its planning activity. We will term this structuring of the space of states a *situation space*.

## An Example Domain - PWorld

It is useful to define an example domain within which the idea of a situation space can be motivated and the intuitions behind this structure discussed. The domain that is used for this purpose is referred to as PWorld. It was loosely defined by analogy to the kind of goals that might govern the movement of persons congregrated at a party. In this world there is a set of agents and a set of cells which are spatially structured as a grid. At any point in time each agent is located at some cell and no two agents may occupy the same cell.

The primitive actions available to each actor are of two types. Either staying in the current location or movement to a rectilinearly adjacent cell. Staying in a location can always be carried out. The rules of movement are basically those of rectilinear motion with constraints which depend upon the location of other agents. First, an agent cannot move to a cell occupied by another agent since this would violate the constraint that only one agent can occupy any cell. More interestingly, when two agents are adjacent only to one another and no other agent then neither agent can move to a cell that would cause the other agent to no longer be adjacent to some agent. Here adjacency holds between two agents if they occupy cells which are immediately rectilinearly or diagonally adjacent. Call this the *politeness constraint*. Note that if the agent is adjacent to more than one other agent, then it is possible for an agent to move to a cell that breaks this adjacency relation and possibly causes the agent not to be adjacent to any other agent. This politeness constraint is sufficient to yield interesting "group" behavior when a sufficient number of agents are "packed" into a fixed set of cells.

These minimally interesting agents can have goals of being adjacent to other agents, not being adjacent to other agents, or some conjunct of these possibilities. We assume that there is no communication about goals among the agents and that no agent's goal involves intentionally blocking or facilitating the goal achievement of another agent. It is a benign though possibly chaotic world. We further assume that the agents' goals are sufficiently diverse to make it unlikely that the true situation is a competitive one. For example, where more than eight agents had the goal of being adjacent to the same other agent.

The agents are assumed to plan and act asynchronously. Whenever, two or more agents attempt to move to the same free location at the same time, we assume a world monitor that indifferently decides which agent's action succeeds. In the case where an agent attempts to move to a cell occupied by an agent who is executing the action of staying in that cell, then the world monitor always rules in favor of the agent that is staying in the cell.

Now let us distinguish one particular agent, called the actor, and consider the problems encountered by the actor in attempting to achieve the goal of being aligned with some specific other agent, the attractor. One agent is said to be aligned with another agent if the agent occupies a cell that is immediately north, south, east, or west of the other.

## Some Observations about PWorld

What are some of the aspects of PWorld that make planning particularly difficult? First, simply consider some estimates of aspects of the state space associated with a PWorld problem. Assume there are 15 agents at a party. Recall that there are five primitive actions that an agent might attempt. A reasonable estimate of the average number of these actions that an agent can actually perform at a point in time is three. With 15 agents there are roughly $3^{15}$ or over 14 million possible next states. Even if each agent could only choose between two actions, then the average branching factor of the state space would be $2^{15}$ or over 33 thousand. Consequently, reasoning through this space of possible next states to find a sequence of actions that guarantees goal achievement regardless of the actions of the other agents would be a formidable search task. The problem is particularly difficult if the plan involves many actions.

Another possibility would be to attempt to accurately model each agent's plan. However, if each agent's plan depends on modelling the planning of other agents, the combinatorics of these models of models, ..., becomes prohibitive, if not actually inconsistent. What seems to be required is some means of forming abstractions or reformulations of the primitive state space useful to the planner. An obvious candidate is to aggregate agents that are adjacent to each other into groups based on the transitive closure of the adjacency relationship. With 15 agents we might have states which involve one group of all agents, states which involve 2 groups of 1 and 14 agents, or 2 and 13, and so on, to the state where there are 15 groups, that is, each agent is isolated. From the rules of motion for individuals, it is possible to determine the rules of group motion and the possibilities of group formation and dissolution for groups of different size and spatial distribution. For example, a two agent group can undergo an identity transformation, a rigid translation in any of the four directions, a clockwise or counter-clockwise circling of one agent about the other, and so on. A three agent group has a set of group transformations unique to it such as deformation from horizontally aligned to a triangular configuration, and so on, together with the two agent group transformations possible when two agents actions are joined and the third allowed to vary independently, and so on. The advantage of such a reformulation accrues from the ability to collapse states based on the symmetries discovered. Unfortunately, our actor would have to be rather well-versed in the mathematical theory of groups to accomplish this reformulation and there is no guarantee that such a reformulation would simplify the planning for achievement of the actor's goal.

These observations about PWorld are consistent with the following conclusions. First, a classical planning formulation of this type of domain is very unlikely to be of practical use. Second, it is unlikely that a coherent formulation that allows the planner to reason from "correct" models of each agent to deterministic predictions of each agent's actions is possible. And, even if possible, it is unlikely that the use of such models is tractable. Third, even if a reformulated space describing aggregate motion could be derived from the laws of individual motion and states, the resulting state would still be highly complex and carry no guarantee that it could be usefully employed to significantly simplify the planning problem. Thus, we seem to find ourselves in the world of problems which require the paradoxical notion of "reactive planning."

## A Plan-Derived Situation Space

One suggestion for dealing with unpredictable environments is to create a universal plan which "specifies appropriate reactions to every possible situation within a given domain..." (Schoppers, 1987). This suggestion is less than helpful in a domain such as PWorld. Agre and Chapman (1987) have studied the problem of selecting actions in an arcade game, Pengo, that shares some of the characteristics of our PWorld. We share their belief that a "state collapsing" mechanism to control the selection of action in domains such as these is required. Their hope is that such a mechanism emerges from an appropriate coupling or "situated" interplay between a "vision" system and concrete activity. In constrast, the hypothesis advanced here is that the traditional representations available in planning systems may provide a basis for usefully collapsing the states associated with a problem to yield a kind of "situated planner."

We refer to this basis for controlling planning as a situation space. The intent is that a situation space provide an efficient representation of an abstract partial plan that can be used to monitor the changing state of the world, provide the planner information about the appropriate goal to pursue in the "current situation", and enforce a goal-oriented coherence to a planning system that typically attempts to achieve and maintain subgoals in a local fashion.

The problem in realizing this intent is that of determining how to identify an abstract partial plan for a problem without exhaustively planning in the primitive problem space. Our proposed solution involves two basic moves. First, note that for a complex domain such as PWorld there will often be many differing plans that might yield a solution. The representation of an abstraction of this "OR-space" of plans can yield a situation space that itself would be difficult to use and will make the identification of a construction procedure more problematic. What is desired is the imposition of a simple structure on the abstract partial plan. This will result in curtailing the possible plans that a planner controlled by this space can produce. And, it can mean that a plan is not found for some specific problem despite the fact that one exists. The second move is to construct this abstract partial plan in as egocentric a fashion as possible. That is, the idea is to exploit the simplification that results by considering the affect that the world can have on the actor's ability to produce actions within a local or egocentrically defined portion of the space rather than to emphasize reasoning about the global properties of the space. An additional problem that must be faced by any constructive procedure is that of the choice of parameters for the basic objects of the space. Clearly, a situation space for a PWorld with a suitably fixed number of cells and involving 15 agents will differ and be largely irrelevant for one that involves only four agents.

The current ideas concerning a plausible method for constructing such a situation space will be presented by sketching a possible situation space for our PWorld problem. The basic strategy is inductive in form but the induction is controlled to yield partial state descriptions that collapse the space into action and goal related classes.

Figure 1 presents an example of a situation space for PWorld. The construction begins with a tentative set of situation predicates, **Free**, **Paired**, and **Surrounded**. This tentative set is based on a classification of partial state configurations that differentially constrain the actor's actions. Then we show how these set of situation predicates might be modified when fit into a situation space.

Consider first the Surrounded predicate, which returns true when there are other agents in the four cells that are vertically and horizontally adjacent to the actor's cell location. The effect of being in such a state is to make the actions of the actor conditional on those four agents. In particular, if the surrounding agents do not move, the actor cannot move. The Paired predicate is when there is exactly one other agent adjacent to the actor (i.e. in just one of the 8 vertically, horizontally, or diagonally adjacent cells) and none of the actor's immediate legal moves bring additional agents into the group. The effect is to place the actor in a two person group, whereby the *politeness constraint* restricts the actor's actions. If the other agent does not move, the actor can take at most 2 different moves and if the other agent continues not to move over some sequence of the actor's actions there will be a a cyclic structure to that action sequence. Free is true when the actor is not adjacent to any other agents or alternatively is in a group of more than 2 persons and is not surrounded. Thus the actor is either not in a group and can thus immediately take any of four different actions (walls permitting) or is capable of leaving the group and thus eventually will be able to take any of four different actions, again assuming the other agents do not move.

These three situation predicates coincide with different restrictions on the Actor's view of its ability to compose actions. This view is egocentric in the sense that it ignores the other agents' actions. Furthermore, it is shortsighted because it ignores the actor's goals. Thus the task remains to coalesce this action based view of what an actor can do with a goal based view of what the actor wants to do.

Consider now a fourth situation, **Aligned**, which is when the actor is vertically or horizontally adjacent to the Attractor. States satisfying this predicate are the desired goal states. Being Aligned does not coincide with a unique or uniform restriction on action. Aligned can restrict the Actor's actions, and can do so to different degrees depending on whether there are other members in the group. For instance, one can be aligned in a two or three person group. The distinction here is that when alignment has been achieved restriction on action is no longer a major hindrance vis-a-vis goal achievement and can even be beneficial.

Clearly, Aligned is important for goal achievement, yet it is not simply defined in terms of restrictions on actions. This is why situations cannot only express restrictions on the ability of the actor to act. A situation space must also overlay onto or otherwise distinguish situations based on some incomplete overall plan or plans to achieve the goal and how subgoals in those plans are associated with situations. So, whereas situations are in part "locally" defined in terms of the actor's capabilities they must also fit into a situation space that coherently relates a situation to the overall goal. This suggests certain modifications to the situation predicates.

For instance, the most important goal is to achieve and maintain alignment. As we have seen, the various states in which the goal is true do not uniquely or uniformly satisfy any of the three situations based on action restriction. So an Aligned situation predicate is defined and associated with the overall goal of achieving and maintaining alignment. However, in order to distinguish the Aligned situation from Surrounded, we further constrain Surrounded so that none of the other, surrounding, agents is the Attractor. Likewise, we constrain Paired so that the other agent is not the Attractor. See Figure 2 for examples that satisfy these modified situation predicates.

Now that we have the situations we must fit them into a situation space that guides the actor's planning and action execution. This involves adding additional structure. First, associated with each situation predicate is a situationally appropriate goal to pursue when the predicate is true. The goal associated with Surrounded is to achieve a state that satisfies Free. The goal associated with Paired is also Free. When in the situation Free, the goal is either "Approach Attractor and Maintain Free" or, failing that just "Maintain Free". This disjunctive goal is due to possibility of approach paths being blocked by other agents.

In addition to the situation predicates and goals, Figure 1 represents possible transitions from one situation to another as arcs connecting situations. Although not explicitly represented in Figure 1, each of these arcs has an associated predicate to be monitored that determines when the transition occurs. There are two kinds of transition arcs, solid and dotted. The solid arcs represent transitions that are associated with the successful achievement of the goal associated with the situation at the tail end of the arc. In contrast, the dotted arcs mark unexpected transitions. A path along solid arcs encodes a possible sequence of situations that represent various "expected" paths for the actor in the situation space. Any such sequence from some current situation to the Aligned situation is a decomposition into a collection of planning islands with the following characteristics:

- there is an order on the planning for the islands and on the execution of the solutions as noted by the arcs,
- an island's goal ignores future islands,
- an island's goal includes some maintenance goals.

This situation space denotes the limited rationality of the Actor. Due to the instability of PWorld it doesn't make sense to make complete plans that plan across future islands. Nevertheless the situation space itself insures that the planning for the present situation coincides with at least one formulation of an abstract decomposition structure for the goal. Roughly speaking, we can characterize this as get Free, Approach Attractor, and Align. Fortuitous transitions simply bypass part of the structure. The maintenance goals serve to guard against catastrophic transitions that shift the actor into a less advantageous situation in terms of the situation space's partial order.

## Concluding Remarks

The further development and evaluation of these ideas of how to construct and use situation spaces in planning requires the pursuit of two directions of research. One is to find an appropriate formal characterization of the construction procedure in order to determine its appropriateness and generality. A second is to experimentally explore the behavior of possible situated planners that can be designed to use the situation space to control planning activity.

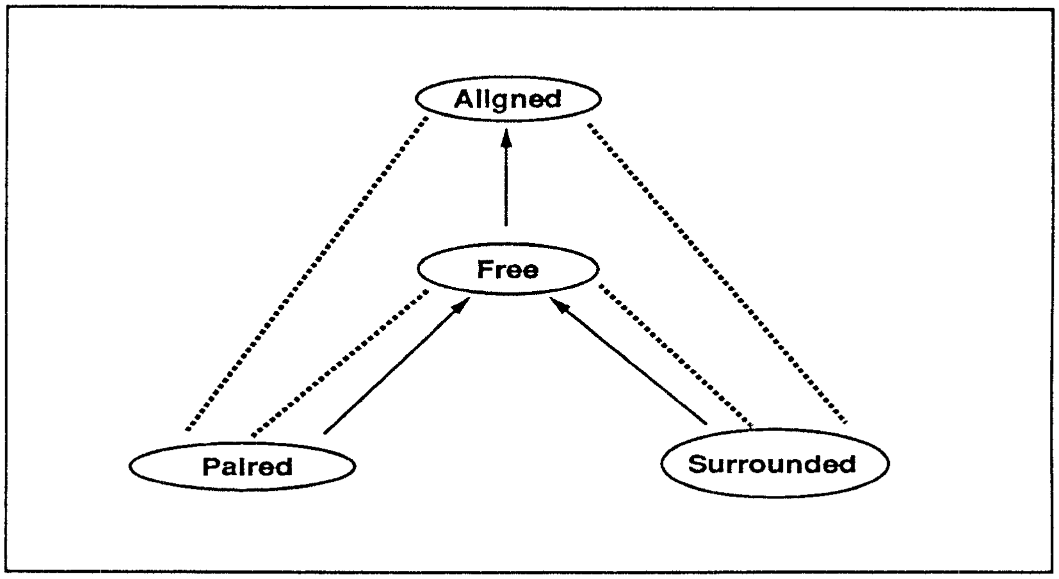


Figure 1: Example Situation Space

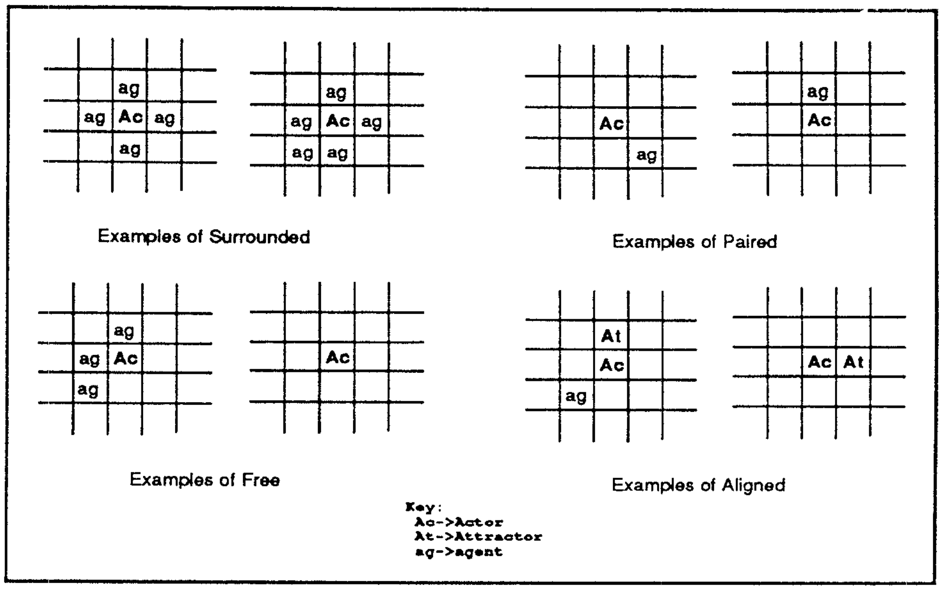


Figure 2: Example Situations

## References

[1] Agre, P.E. and Chapman, D. Pengi: An implementation of a theory of activity. In Proceedings of the National Conference on Artificial Intelligence, AAAI, 1987, 268-272.

[2] Firby, R.J. An investigation into reactive planning in complex domains. In Proceedings of the National Conference on Artificial Intelligence, AAAI, 1987, 202-206.

[3] Hendler, J.A. and Sanborn, J.C. A model of reaction for planning in dynamic environments. In Proceedings of the Knowledge-Based Planning Workshop, DARPA, 1987, 24-1 - 24-10.

[4] Schmidt, C.F., Goodson, J.L., Marsella, S.C., Bresina, J.L., Reactive Planning Using a Situation Space. In Proceedings of the 1989 AI Systems in Government Conference, IEEE, 1989.

[5] Schoppers, M.J. Universal plans for reactive robots in unpredictable environments. In Proceedings of the Eighth International Joint Conference on Artificial Intelligence, IJCAI, 1987, 1039-1042.

# A Logic for a Non-monotonic Theory of Planning

**Leora Morgenstern**
IBM T. J. Watson Research Center
P.O. Box 704, Mail Stop H1N08
Yorktown Heights, NY 10598
(914)784-7151
E-mail: leora@IBM.COM

## 1 Abstract

We argue that a realistic theory of planning must be based on a non-monotonic logic. We present MANML, a multiple-agent non-monotonic logic, which is an extension of Moore's autoepistemic logic. Several inference rules are suggested for MANML. While these allow the sort of reasoning that one needs in multi-agent planning, they are shown to be too permissive. Finally, we show that a restriction of one of these inference rules gives intuitive results for temporal projection.

## 2 Introduction

AI researchers have long recognized that agents who operate in complex environments often don't know enough to perform desired actions or to achieve their goals. During the past two decades, researchers have responded to this problem via two distinct approaches. On the one hand, formalists have developed theories that allow an agent to reason about his ability to perform an action ([McCarthy and Hayes 1969], [Moore 1980]. and to execute complex plans even in the absence of knowledge [Morgenstern 1987]. On the other hand, more practically oriented AI researchers have investigated *reactive* planners ([Kaelbling 1986],[Schoppers 1987]) where little advance planning is performed, actions - reactions to a particular state of the world - are hard-wired into the system, and the effect of incomplete knowledge is minimized.

For many reasonably complex planning problems, however, both of these approaches are inadequate. Reactive planning systems may be a good solution for simple planning scenarios, especially for those situations in which an agent will not have the necessary information ahead of time, but will have that information at run time. Nevertheless, there are many situations in which stragetic planning is both necessary and natural. If an agent needs a piece of information, and the only way to get that information is to ask another agent, waiting until run time won't get him anywhere. The point, clearly, is that the agent must explicitly plan to ask the second agent for the information, and must somehow reason that this plan will work. On the other hand, the formalist approach as it has so far been developed is too rigid to model reasonably complex planning problems. Consider for example, the Chain Request Problem:

Suppose Alice wants to open a safe. She knows that the combination of the safe is either some sequence N1 or N2, but she doesn't know which. It is crucial to know the correct combination before attempting to dial, because dialing the wrong combination causes the safe to blow up. Furthermore, various authorized individuals may change the combination of the safe from N1 to N2 and vice versa. Alice knows that both Jim and Lisa know the combination of the safe, but she is not friendly enough with either of them to ask. However, she knows Susan quite well, and she knows that Susan is friendly with either Lisa or Jim. Alice therefore constructs the following plan:

- Alice asks Susan to ask either Jim or Lisa for the combination
- Susan asks Jim or Lisa for the combination
- Jim or Lisa tells Susan the combination
- Susan tells Alice the combination

Despite the apparent simplicity of this plan, Alice cannot predict with any certainty that this plan will work. She can assume that Jim or Lisa will give Susan the information, but she cannot be absolutely certain of it; she can be reasonably confident that the combination of the safe won't change during this process - and that Jim and Lisa will know this - but this is not an absolute certainty.

The standard formalist approach has concentrated on iron-clad proofs showing that an ignorant agent could do something to acheive his goals. Often such proofs have relied on axioms that were drastic oversimplifications of commonsense knowledge. In contrast, we aim to develop a theory with realistic axioms, default rules, and defeasible conclusions. That is, we would like to develop a theory of planning that is grounded in a non-monotonic logic.

Given that it is necessary to ground a robust theory of planning in a non-monotonic logic, why not simply fuse an existing non-monotonic logic with an existing theory of knowledge and action? Such an approach would be inadequate, primarily because existing non-monotonic logics are intended for modelling how a single agent reasons with partial information and with default rules. In a non-monotonic theory of planning, however, the single-agent case no longer suffices. When an agent plans, he will often take into account the actions that he believes another agent will perform. In order to predict the actions of another agent, he must know how that other agent plans, i.e., he must know how that other agent reasons non-monotonically. Thus, we need a multiple agent non-monotonic logic.

## 3 MANML

MANML is a Multiple Agent Non-Monotonic Logic that has been developed in recent work [Morgenstern 1990]. The core of MANML is an extension of Moore's autoepistemic logic (AEL) to the multiple agent case.

AEL was designed to formalize how an agent reasons about his own beliefs. Sentences of AEL are defined by the following rules: (1) if $\phi$ is a formula of the predicate calculus, $\phi \in$ AEL; (2) if $\phi \in$ AEL, $L\phi \in$ AEL, where $L$ is the standard belief operator; (3) if $\phi$ and $\psi$ are sentences of AEL, so are $\phi \wedge \psi$ and $\neg\phi$.

We say a theory $T$ of AEL is a stable set if it obeys the following three rules:
[1] $T$ is closed under logical consequence
[2] if $P \in T$, then $LP \in T$
[3] if $P \notin T$, then $\neg LP \in T$.

To extend this logic to multiple agents, we index the belief operator $L$. We thus state the formation rules of MANML as follows:
(1) if $\phi$ is a sentence of the predicate calculus, $\phi$ is a sentence of MANML.
(2) if $\phi$ is a sentence of MANML, $L_a\phi$ is a sentence of MANML, where $a$ is a constant of the language that represents an agent
(3) if $\phi$ and $\psi$ are sentences of MANML, so are $\phi \wedge \psi$ and $\neg\phi$

Once we introduce multiple agents into the theory, the stable set formation rules of AEL no longer hold. If $P$ is in $T$, we do not necessarily want to say that $L_a P$ is in $T$, for any $a$. Nevertheless, we wish to get the effect of these rules, so that agents can reason autoepistemically and reason about other agents reasoning autoepistemically. We alter the stable set formation rules by adding an explicit level of indexing in the obvious way. This yields the following set of rules:

1. if $L_a P_1, ..., L_a P_n \in T, P_1 ... P_n \vdash Q$, then $L_a Q \in T$
2. if $L_a P \in T$, then $L_a L_a P \in T$
3. if $L_a P \notin T$, then $L_a \neg L_a P \in T$

Default rules must also be indexed appropriately. Bill's belief that he would know if he had an older brother is represented as $L_{Bill}(P \Rightarrow L_{Bill}P)$, where P stands for the sentence: Bill has an older brother. Suppose $L_{Bill}(P \Rightarrow L_{Bill}P)$ is the only sentence of the form $L_{Bill}\phi$ in $T$. By 1., $L_{Bill}(\neg L_{Bill}P \Rightarrow \neg P)$. But, since $L_{Bill}P \notin T$, by 3., $L_{Bill}\neg L_{Bill}P \in T$. Thus, by 1., $L_{Bill}\neg P$.

We have the following **Theorem:** Let $T$ be a set of sentences of MANML. Let $T_\alpha = \{ \phi | L_\alpha \phi \in T\}$. Then, $L_\alpha P$ is a MANML stable set consequence of $T$ iff $P$ is an AE stable set consequence of $T_\alpha$.

Note that, in restricted cases, MANML seems to permit *other* agents to reason about an individual agent's autoepistemic reasoning. Suppose, for example, that $T$ contains the following axioms: $L_{Bill}L_{Alex}(P \Rightarrow L_{Alex}P))$ (Bill believes that if Alex had an older brother, Alex would know about it) and $L_{Bill}(\neg L_{Alex}P)$. Then by rule 1. of MANML, we get $L_{Bill}L_{Alex}\neg P$.

In the foregoing example, however, Bill did not really reason about Alex's autoepistemic reasoning abilities at all. He explicitly knew that Alex didn't believe that he had an older brother. This goes against the spirit of autoepistemic reasoning. The point is to start out from the positive beliefs in one's data base, use the stable set principles to conclude what beliefs one doesn't have, and to go from there to negative beliefs.

Similarly, recall the Chain Request Problem of Section 1, and suppose that Susan believes that friendly agents typically give over information when they are requested to do so. Using a simple temporal logic where actions take unit time (as in [Morgenstern 1989]), following Konolige's [1987] suggestion for representing default rules in autoepistemic logic, and adding the appropriate indexing, we can represent this rule as

$L_{Susan}$
$[L_{Susan}(True(t, Occurs(do(a, request(b, tellw(d)))$
$\wedge True(t, friendly(a, b)))$
$\wedge \neg L_{Susan}\neg True(t+1, Occurs(do(b, tellw(d))))$
$\Rightarrow True(t+1, do(b, tellw(d)))]$

tellw describes the action of an agent taking a disjunction and telling which disjunct is true.

Suppose also that Susan belives that $a$ and $b$ are friendly and that $a$ has just requested $b$ to tell him which of the disjuncts in d is true, i.e.
$L_{Susan}True(t, Occurs(do(a, request(b, tell(d))))$
$\wedge True(t, friendly(a, b)))$
and that Alice knows that Susan has these beliefs. In order for Alice to believe that Susan believes that $b$ in fact will do the telling action, Alice must also believe that Susan does *not* believe that the telling action will not happen. Again, these constraints go against the spirit of non-monotonic reasoning. The whole point is that agents need not have explicit knowledge of the conditions that are assumed to be true by default.

How can one agent reason about the way in which another agent uses default rules? The core of the answer is this: Agents reason about how other agents reason non-monotonically by making default assumptions about what these agents do *not* believe. Note that these default assumptions are quite different from the assumption of perfect introspection that characterizes autoepistemic reasoning in the single-agent case. In the single-agent case, the given theory was a complete description of the mind of the agent. In the multi-agent case, agents have at best a partial description of other agents' beliefs.

Note that the default assumptions that one agents makes about another agent's lack of beliefs is defeasible. In fact, one agent may be wrong in what he thinks another agent does not believe. Moreover, this principle embodies a certain amount of arrogance. An agent who reasons about how a second agent uses non-monotonic reasoning must be arrogant with respect to his beliefs about the *limitations* of the second agent's beliefs. He must in some sense believe that he knows all that is important to know about the second agent's beliefs.

We aim to limit this arrogance as much as possible. For any default rule of the form $L_a\alpha \wedge \neg L_a\beta \Rightarrow L_a\gamma$ let us call $\neg La\beta$ the *ignorant* part of the rule, since it deals with an agent's negative beliefs. To enable multi-agent non-monotonic reasoning, we need only assume that agents are arrogant with respect to the negative parts of the default rules.

A first step at a principle of inference for MANML might therefore be:
If an agent X believes that a second agent Y believes some default rule $L_Y\alpha \wedge \neg L_Y\beta \Rightarrow \gamma$, and X believes that Y believes $\alpha$ and has no reason to believe that Y believes $\beta$, X can conclude that Y believes $\gamma$. Formally, suppose:

$L_XL_Y(L_Y\alpha \wedge \neg L_Y\beta \Rightarrow \gamma) \in T$, $L_XL_Y\alpha \in T$, $L_XL_Y\beta \notin T$. Then, $L_XL_Y\gamma \in T$.

We will call the above principle the Principle of Moderate Arrogance (PMA).

It can easily be seen that the Principle of Moderate Arrogance allows us to model in a rational manner how Alice comes to conclude that Susan will believe that Lisa or Jim will tell her what the combination is.

Often, even an arrogant agent finds it worthwhile to be more circumspect about ascribing negative beliefs to other agents. This is particularly the case when the agent *does* believe the negative part of some default rule. We call this rule of inference the Principle of Cautious Arrogance (PCA). It is formalized as follows: Suppose $L_X L_Y(L_Y\alpha \wedge \neg L_Y\beta \Rightarrow \gamma) \in T$, $L_X L_Y\alpha \in T$, $L_X L_Y\beta \notin T$, and $L_X\beta \notin T$. Then $L_X L_Y\gamma \in T$.

The PCA may be too cautious at times. In general, however, both the PMA and the PCA are much too permissive. Below, we discuss a restriction of the PMA that works well in temporal projection.

## 4 Epistemic Motivated Action Theory in MANML

Epistemic Motivated Action Theory (EMAT) was originally developed to handle two strange variant frame problems that arise in an integrated theory of knowledge and multi-agent planning ([Morgenstern 1989]). The Chain Request Problem, which we introduced in Section 1, captures the salient features of these problems in a simpler format: [1]

Consider Alice's simple 4-step plan (ask Susan to ask Lisa or Jim for the combination, Susan asks Lisa or Jim, Lisa or Jim tells Susan, Susan tells Alice the combination) It turns out that in a standard monotonic logic with frame axioms, Alice cannot prove that her plan will work. She does not know that Lisa or Jim will tell Susan the combination, since Lisa or Jim will have no way of knowing that the combination of the safe has not changed since the beginning of the planning process. No amount of frame axioms will help. The problem is not a lack of frame axioms, or even a lack of knowledge on the part of Lisa and Jim of the frame axioms. However, since Lisa and Jim may not know what has happened during the planning process, they may have no way of applying these frame axioms.

Surprisingly, standard non-monotonic temporal logics ([Lifschitz 1987], [Haugh 1987], and [Baker and Ginsberg 1988]) are of no help either. Very briefly, the reason these logics won't work is that they are based on the situation calculus, and therefore allow no gaps in any agent's knowledge of what actions are occurring in a chronicle.

Of course, a commonsense reasoning system should be able to conclude that Alice's plan will work. Presumably, the combination of the safe will not change. Lisa and Jim know this. Therefore, as long as Jim and Lisa don't know of anything that would indicate that the combination has changed, they will assume that it hasn't changed. Alice therefore reasons that Lisa or Jim will know the combination when Susan asks.

The basic principle underlying the foregoing reasoning is that actions happen only if they have to happen, or are *motivated*. This principle has been formalized Motivated Action Theory (MAT) [Morgenstern and Stein 1988]. We assume a theory of causal and persistence rules T, and a collection of statements giving a partial description of a chronicle, CD. $CD \cup T = TI$, a particular theory instantiation. A statement is said to be *motivated* is a theorem of TI; a statement is said to be motivated with respect to a particular model if it has to be true, given rules and boundary conditions, within that particular model. We prefer models which minimize unmotivated statements of the form T(t,Occurs(act)).

EMAT extends MAT by parameterizing theory instantiations with respect to agents and times. For example, TI(a,t1,b,t2) = TI(a,t1)(b,t2) is what a at t1 believes b at t2 believes. Agents assume that other agents reason using MAT on the theory instantiations which they ascribe to them.

In the above example, EMAT allows Alice to prove that her 4-step plan will work. The theory instantiations TI(Alice,1,Jim,3) and TI(Alice,1,Lisa,3) both contain the statement that the combination at time 3 is identical to the combination at time 1; thus, Lisa and Jim know the combination.

EMAT provides a simple and elegant solution to the problem of temporal projection in epistemic contexts. The principle embodied in EMAT is quite close to a restricted form of the PMA in MANML. Note that the principle underlying MAT and EMAT - actions happen only if they have to happen - can be captured by the following axiom schema of MANML: [2]

$$L_a(\neg L_a Occurs(act) \Rightarrow \neg Occurs(act))$$

[1] The original variant frame problems and their solutions in EMAT, were developed in a rich logical language that allowed quantification into epistemic contexts. AEL (even Konolige's extended version) and therefore MANML do not allow quantification into epistemic contexts.

[2] the correspondence is close but not exact.

That is, it is assumed by default that unmotivated actions do not occur.

Agents in EMAT implicitly assume that the partial characterization that they have of the other agents' theory instantiations is sufficient for their purposes. This assumption can be made explicit in the following restricted form of the PMA, which is limited to default rules of causal reasoning. This restricted form of the PMA, (EMAT-PMA) can be stated as follows:

Suppose

$L_X L_Y (L_Y \alpha \quad \wedge$
$\neg L_Y True(t, Occurs(act))) \Rightarrow \gamma$

$L_X L_Y \alpha \in T$

$L_X L_Y True(t, Occurs(act)) \notin T$

Then $L_X L_Y \gamma \in T$

This gives us a powerful inference rule for non-monotonic temporal reasoning. EMAT-PMA will allow Alice to reason that Jim or Lisa will give Susan the combination, and that Susan will give her the combination. Naturally, a logic that has EMAT-PMA will not permit much of the reasoning that a commonsense reasoner ought to be able to do. Nonetheless, the reasonableness of this inference rule suggest the possibility that a group of rules of this sort, each expressing a restriction of PMA for some sort of reasoning, is a good first step toward building a general purpose theory of non-monotonic planning.

## 5 Conclusion

We have argued that a theory of non-monotonic planning is crucial for a realistic theory of commonsense reasoning. We have presented MANML, a logic of multiple-agent non-monotonic reasoning, and have showed that it is a good basis for a non-monotonic theory of planning. We have suggested several inference rules for MANML, based on the concept of an agent's arrogance towards his knowledge of another agent's ignorance. These rules were shown to be overly permissive. We demonstrated that an existing theory of temporal reasoning, which allowed for limited multiple-agent non-monotonic reasoning, could be duplicated by a restricted form of one of the principles of arrogance.

## 6 References

Baker, Andrew and Matthew Ginsberg: Some Problems in Temporal Reasoning, 1988

Haugh, Brian: Simple Causal Minimizations for Temporal Persistence and Projection, Proceedings, AAAI 1987

Kaelbling, Leslie: An Architecture for Intelligent Reactive Systems, in Georgeff, Michael and Amy Lansky, eds: Reasoning About Actions and Plans, Proceedings of the 1986 Workshop at Timberline, 1987

Konolige, Kurt: On the Relation Between Default Theories and Autoepistemic Logic, Proceedings, IJCAI 1987

Lifschitz, Vladimir: Formal Theories of Action, Proceedings, IJCAI 1987

McCarthy, John and Patrick Hayes: Some Philosophical Problems from the Standpoint of Artificial Intelligence, in Bernard Meltzer, ed. Machine Intelligence, 1969

Moore, Robert: Reasoning About Knowledge and Action, SRI TR 191, 1980

Moore, Robert: Semantical Considerations on Nonmonotonic Logic, AIJ, Vol.25, 1985

Morgenstern, Leora: Knowledge and the Frame Problem, Proceedings, Workshop on the Frame Problem, Pensacola, 1989

Morgenstern, Leora: Knowledge Preconditions for Actions and Plans, IJCAI 1987

Morgenstern, Leora: Preliminary Investigations into a Theory of Multiple Agent Nonmonotonic Reasoning, submitted to 1990 Workshop on Nonmonotonic Reasoning

Morgenstern, Leora and Lynn Andrea Stein: Why Things go Wrong: A Formal Theory of Causal Reasoning, Proceedings, AAAI 1988

Reiter, Ray: A Logic for Default Reasoning, AIJ, Vol. 13, 1980

Schoppers, Marcel: Universal Plans for Reactive Robots in Unpredictable Environments, IJCAI, 1987

# INTRODUCING THE TILEWORLD: EXPERIMENTALLY EVALUATING AGENT ARCHITECTURES

Martha E. Pollack
Artificial Intelligence Center *and*
Center for the Study of Language and Information
SRI International
Menlo Park, CA
pollack@ai.sri.com

Marc Ringuette
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA
mnr@cs.cmu.edu

## Introduction

Recently there has been a surge of interest in systems that are capable of intelligent behavior in dynamic, unpredictable environments. Because agents inevitably have bounded computational resources, their deliberations about what to do take time, and so, in dynamic environments, they run the risk that things will change while they reason. Indeed, things may change in ways that undermine the very assumptions upon which the reasoning is proceeding. The agent may begin a deliberation problem with a particular set of available options, but, in a dynamic environment, new options may arise, and formerly existing options disappear, during the course of the deliberation. An agent that blindly pushes forward with the original deliberation problem, without regard to the amount of time it is taking or the changes meanwhile going on, is not likely to make rational decisions.

One solution that has been proposed eliminates explicit execution-time reasoning by compiling into the agent all decisions about what to do in particular situations [Agre and Chapman, 1987, Brooks, 1987, Kaelbling, 1988]. This is an interesting endeavor whose ultimate feasibility remains an open question, but we and others believe that in complex domains, the exclusive use of compilation techniques is impractical [D'Ambrosio and Fehling, 1989, Doyle, 1988, Pollock, 1989].

An alternative is to design agents that perform explicit reasoning at execution time, but manage that reasoning by engaging in *meta-level reasoning.* Within the past few years, researchers in AI have provided theoretical analyses of meta-level reasoning, often applying decision-theoretic notions to it [Boddy and Dean, 1989, Russell and Wefald, 1989, Horvitz, 1987]. In addition, architectural specifications for agents performing meta-level reasoning have been developed [Bratman *et al.*, 1988], and prototype systems that engage in meta-level reasoning have been implemented [Cohen *et al.*, 1989, Georgeff and Ingrand, 1989]. The project we describe in this paper involves the implementation of a system for experimentally evaluating competing theoretical and architectural proposals.

More specifically, we have been constructing a system called Tileworld, which consists of a simulated robot agent and a simulated environment which is both dynamic and unpredictable. Both the agent and the environment are highly parameterized, enabling one to control certain characteristics of each. We can thus experimentally investigate the behavior of various meta-level reasoning strategies by tuning the parameters of the agent, and can assess the success of alternative strategies in different environments, by tuning the environmental parameters. Our hypothesis is that the appropriateness of a particular meta-level reasoning strategy will depend in large part upon the characteristics of the environment in which the agent incorporating that strategy is situated. We shall describe below how the parameters of our simulated environment correspond to interesting characteristics of real, dynamic environments.

In our initial experiments using Tileworld, we have been evaluating a version of the meta-level reasoning strategy proposed in earlier work by one of the authors [Bratman *et al.*, 1988]. However, we will also describe how Tileworld can be used to evaluate a range of competing proposals, such as the ones mentioned above.

## The Tileworld Environment

The Tileworld is a chessboard-like grid on which there are agents, tiles, obstacles, and holes. An agent is a unit square which is able to move up, down, left, or right, one cell at a time. A tile is a unit square which behaves like a tile: it slides, and rows of tiles can be pushed by the agent. An obstacle is a group of grid cells which are immovable. A hole is a group of grid cells, each of which can be "papered over" by a tile when the tile is moved on top of the hole cell; the tile and hole cell disappear, leaving a blank cell. If a hole becomes completely filled, the agent gets points for filling it in. The agent knows ahead of time how valuable the hole is; its overall goal is to get as many points as possible by filling in holes.

```
# # # # # # # # # # # # # # # # # # # #
#     T   T         T       T         #
#       #                   2 2     T #
#       #                   2         #
#       # # 5             T           #
#     # # # 5 T                       #
#         # 5   T         a         T #
#       T                             #
#         T     T             T T     #
#               # T # T   T           #
#       T       # # # #               #
#     #           # #         T       #
#     #   T   T   T       T           #
#     #                       #       #
#                         T   #   #   #
#     T                   # # # # #   #
#     # # # # #                       #
#     #                     T T       #
#                   T         T     T #
# # # # # # # # # # # # # # # # # # # #
```

*A Typical Tileworld Starting State*

a = *agent*, # = *obstacle*, T = *tile*, <*digits*>=*hole*

A Tileworld simulation takes place dynamically: it begins in a state which is randomly generated by the simulator according to a set of parameters, and changes continually over time. Objects (holes, tiles, and obstacles) appear and disappear at rates determined by parameters set by the experimenter, while at the same time the agent moves around and pushes tiles into holes. The dynamic aspect of a Tileworld simulation distinguishes it from many earlier domains that have been used for studying AI planning, such as blocks-world.

The Tileworld is a rough abstraction of the Robot Delivery Domain, in which a mobile robot roams the halls of an office delivering messages and objects in response to human requests.[1] We have been able to draw a fairly close correspondence between the two domains (i.e., the appearance of a hole corresponds to a request, the hole itself corresponds to a delivery location, tiles correspond to messages or objects, the agent to the robot, the grid to hallways, and the simulator time to real time).

Features of the domain put a variety of demands on the agent. Its spatial complexity is nontrivial: a simple hill-climbing strategy can have modest success, but when efficient action is needed, more extensive reasoning is necessary. But the time spent in reasoning has an associated cost, both in lost opportunities and in unexpected changes to the world; thus the agent must make tradeoffs between speed and accuracy, and must monitor the execution of its plans to ensure success. Time pressures also become significant as multiple goals vie for the agent's attention.

The Tileworld can be a good test of an agent's abilities to behave intelligently in a dynamic, unpredictable environment. But a single Tileworld simulation, however interesting, will give only one data point in the design space of robot agents. To explore the space more vigorously, we must be able to vary the challenges that the domain presents to the agent. We have therefore parameterized the domain, and provided "knobs" which can be adjusted to set the values of those parameters.

The knob settings control the evolution of a Tileworld simulation. Some of the knobs were alluded to earlier, for instance, those that control the frequency of appearance and disappearance of each object type. Other knobs control the number and average size of each object type. Still other knobs are used to control factors such as the shape of the distribution of scores associated with holes, or the choice between the instantaneous disappearance of a hole and a slow decrease in value (a hard bound versus a soft bound). By adjusting the knobs, one can allow conditions to to vary from something resembling an unconstrained football field to something like a crowded maze, or from fixed puzzle to constantly changing chaos. For each set of parameter settings, an agent can be tested on tens or hundreds of randomly-generated runs automatically. Agents can be compared by running them on the same set of pseudorandom worlds; the simulator is designed to minimize noise and preserve fine distinctions in performance. We will describe the form of an experiment more precisely in a later section.

## An Agent Architecture

In our initial experiments with Tileworld, we are investigating a particular architecture for meta-level reasoning [Bratman *et al.*, 1988], which we briefly describe here. This architecture builds on observations made by Bratman [Bratman, 1987] that agents who are situated in dynamic environments benefit from having plans because their plans can constrain the amount of subsequent reasoning they need to perform. Two constraining roles of plans will concern us here [2]:

- An agent's plans focus subsequent means-end reasoning so that the agent can, in general, concen-

[1] Various projects at SRI have employed this domain, some of them also employing an actual mobile robot, Flakey.

[2] Additional constraining roles have also been postulated [Bratman, 1987, Pollack, 1990].

trate on elaborating its existing plans, rather than on computing all possible courses of action that might be undertaken.

- An agent's plans restrict the set of further potential courses of action it needs to give full consideration to, by filtering out options that are inconsistent with the performance of what it already plans to do.

The first role of plans has always been at least implicit in the standard models of AI planning: AI planners compute means to goals that the agent already has. The second has a more dramatic effect on the architecture we are investigating: it leads to the introduction of a *filtering mechanism*, which manages execution time reasoning by restricting deliberation to options that are compatible with the performance of already intended actions. (To have the desired effect of lessening the amount of reasoning needed, the filtering mechanism must be computationally inexpensive, relative to the cost of deliberation.)

Of course, a rational agent cannot *always* remain committed to its existing plans. Sometimes plans may be subject to reconsideration or abandonment in light of changes in belief. But if an agent constantly reconsiders its plans, they will not limit deliberation in the way they need to. This means that an agent's plans should be reasonably stable, i.e., they should be relatively resistant to reconsideration and abandonment.

To achieve stability while at the same time allowing for reconsideration of plans when necessary, we include two components in the filtering mechanism. The first checks a new option for compatibility with the existing plans. The second, an override mechanism, encodes the conditions under which some portion of its existing plans is to be suspended and weighed against some other option. The filter override mechanism operates in parallel with the compatibility filter. For a new option to pass through the filter, it must either pass the compatibility check or else trigger an override by matching one of the conditions in the override mechanism.

An agent's filter override mechanism must be carefully designed to embody the right degree of sensitivity to the problems and opportunities that arise in its environment. If the agent is overly sensitive, willing to reconsider its plans in response to every unanticipated event, then its plans will not serve sufficiently to limit the number of options about which it must deliberate. On the other hand, if the agent is not sensitive enough, it will fail to react to significant deviations from its expectations.

The options that pass through the filter are then subject to deliberation. The deliberation process is what actually selects the actions the agent will form intentions towards. In other words, it is the deliberation process that performs the type of decision-making that is the focus of traditional decision theory. The filtering mechanism thus serves to frame particular decision problems.

## The Tileworld Agent

The Tileworld agent embodies the architecture described in the previous section. It simultaneously reasons about what to do, and performs actions and perceives changes in its environment. Our model is of a robot with two sets of processing hardware. One processor executes a short control cycle, acting on previously formulated plans and monitoring the world for changes; the second processor executes a longer cycle which permits computations with lengths of up to several seconds. Although this model incurs a certain cost in the complexity of synchronizing the two processes, it allows for a balance of computational flexibility and reactivity. We feel that this is a realistic choice for robot design, although in our current system, we simulate the concurrency for the sake of convenience.

The act cycle is straightforward; the agent performs those acts that have been identified during the previous reasoning cycle, monitoring for limited kinds of failures. The reasoning cycle is more interesting, and the portion of the agent architecture that controls reasoning is sketched below.

```
                                Environment
                                     |
                                     v
            ----------           ------------
           | Means-   |         | Perception |
      ---> | Ends     |          ------------
      |    | Reasoner |               |
      |     ----------                |
      |         |                     |
      |         v                     v
     ---                ---------------------
    | P | -->          | Filtering Mechanism |
    | L |               ---------------------
    | A |                       |
    | N |                       |
    | S |                       v
     ---                 ----------------------
      ^                 | Deliberation Process |
      |                  ----------------------
      |_____________________|
```

The task of the reasoning cycle is to make decisions about what goals to pursue and how to pursue them. New options for consideration can come from two sources. First, the agent may perceive environmental changes that suggest new options—in Tileworld, this occurs when new holes or tiles appear. Second, options may be suggested by a means-end reasoner. This is currently a fairly standard backwards-chaining planner, augmented with special-purpose routines for route-planning. The means-end reasoner suggests options that can serve as means to already intended ends. For example, it may suggest moving to a certain location in order to push a particular tile into some hole, when the filling of that hole is a component of the agent's current plans.[3]

Options from both sources are then subject to filtering. Most of our efforts so far have concerned the filtering of top-level options, i.e., options to fill a particular hole, and we focus on that here. However, at least some extensions to subordinate options are obvious: for example, the use of particular tile to fill one hole should be filtered as incompatible when there already exists an plan to use that tile for a different hole.

Compatibility checking of top-level options is straightforward. If the agent has a current intention to fill a particular hole (say, hole $N$) right now, then it is the case that filling any other hole $M$ right now will be incompatible with the existing intention. Consider then what happens if the agent is filling hole $N$ when hole $M$ appears. The option to fill hole $M$ now will *not* survive the compatibility filter. Thus, deliberation about whether to abandon, at least temporarily, work on $N$, and instead work on $M$, will depend upon the override mechanism. In the simplest version of the override mechanism, a threshold level is set to some constant $v$, which represents the marginal increase in potential value that the new hole must have over the old one to be worthy of further consideration. In other words, if $(score(M) - score(N)) > v$, deliberation about whether to work on $M$ will ensue. (The function $score(X)$ denotes the number of points the agent will receive for filling $X$.) Recall that deliberation will not necessarily result in the agent's abandoning its currently executing plan; that depends upon the details of the deliberation component, described below. However, if $(score(M) - score(N)) <= v$, then the agent will not even consider abandoning its filling of $N$, and will defer without further consideration attempts to fill $M$. Notice that if we set the threshold value to $-\infty$, deliberation will occur whenever the environment changes in such a way as to provide a new potential option.

The deliberation process is used to decide amongst competing options. In the example we have been describing, assume that filling $M$ now survives the filtering mechanism. Then it is necessary to deliberate about whether in fact to adopt that intention, and begin work on filling $M$, or whether to continue with the current plan of filling $N$. Alternative deliberation strategies can be chosen in Tileworld by the setting of a parameter. We currently have implemented two deliberation modules.

The simpler deliberation module evaluates competing top-level options by selecting the one with the higher potential score. Thus, when the threshold parameter for the override mechanism is nonnegative, this mode of deliberation will always select the new competing option over the one that was previously held. This illustrates a general point: if deliberation is extremely simple, it may be redundant to posit separate deliberation and filtering processes.

A slightly more sophisticated deliberation strategy estimates the subjective expected utility (SEU) of a top-level goal. For a given option to fill a hole $M$, SEU is estimated as a function of $score(M)$, time available to fill $M$, distances of the agent and available tiles from $M$, and the size of $M$; these factors can be combined into an improved measure of the likelihood of success of filling $M$ in the time allotted. With this mode of deliberation, the agent may decide to continue with its current plan to fill $N$, even if filling $M$ has a higher potential score (which would be necessary for filling $M$ even to survive the filter); this will occur if the filling of $N$ has a significantly higher likelihood of success.

We intend to design additional deliberation modules, including one that simulates complete means-end reasoning for options under consideration.

## Experiments With Our Agent

With both the simulator and the agent in place, we are in a position to conduct experimental studies of the the performance of the agent and thereby illuminate the tradeoffs inherent in some of our design decisions.

By adjusting the Tileworld "knobs", we can control a number of domain characteristics. We can vary what we call *dynamism* (the rate at which new holes appear), *hostility* (the rate at which obstacles appear), *variability of utility* (differences in hole values), *variability of difficulty* (differences in hole sizes and distances from tiles), and *hard/soft bounds* (holes having either hard timeouts or gradually decaying in value). There are also variables we can adjust in the agent: *act/think rate* (the relative speeds of acting and thinking), the filter's *threshold level*, and the *sophistication of the deliberation mechanism*.

To perform an experiment, we begin by holding fixed all but a single parameter. We then run many simula-

[3]By the agent's "current plans" we mean those courses of action it has already formed an intention to perform.

tions, varying the setting of the parameter of interest, and recording the score received in each simulation. For example, by varying the dynamism of the domain, we can examine how a particular Tileworld agent with fixed settings reacts to more and less rapidly changing environments. It is also useful to vary two dimensions at once: for instance, we are interested in seeing the scores achieved by our agent as we vary both the *threshold level* and the *act/think rate*. Some other pairings of variables which we currently consider interesting are *threshold* vs. *variability of utility*, *threshold* vs. *sophistication of deliberation*, and *act/think rate* vs. *sophistication of deliberation*. Extensive experiments of this kind are made much more feasible by the capability of the simulator to generate automatically an unlimited number of test runs.

## Experiments With Other Agents

We also see the Tileworld testbed as a good basis for comparison of other agent architectures proposed in the literature. We intend not only to mix and match components of our own agent, but also to investigate the performance of entirely different architectures in our domain.[4]

The goal of our experiments is an improved understanding of the relation between agent design and environmental factors. In the future, when faced with a performance domain for an agent, one should be able to draw on such an understanding to choose more wisely from the wide range of implementation possibilities available.

## Acknowledgements

This research was supported by the Office of Naval Research under Contract No. N00014-85-C-0251, by a contract with the Nippon Telegraph and Telephone Corporation and by a gift from the System Development Foundation.

## References

[Agre and Chapman, 1987] Philip E. Agre and David Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, Wa., 1987.

[Boddy and Dean, 1989] Mark Boddy and Thomas Dean. Solving time-dependent planning problems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, 1989.

[4]We strongly encourage other researchers to demonstrate their agents in our domain: it is relatively clean and portable, is written in CommonLisp, and is available electronically over the Internet from Marc Ringuette at `mnr@cs.cmu.edu`.

[Bratman *et al.*, 1988] Michael E. Bratman, David J. Israel, and Martha E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4(4), 1988.

[Bratman, 1987] Michael E. Bratman. *Intention, Plans and Practical Reason.* Harvard University Press, Cambridge, Ma., 1987.

[Brooks, 1987] Rodney A. Brooks. Planning is just a way of avoiding figuring out what to do next. Technical Report 303, MIT, 1987.

[Cohen *et al.*, 1989] P. R. Cohen, M. L Greenberg, D. M. Hart, and A. E. Howe. Real-time problem solving in the phoenix environment. In *Proceedings of the Workshop on Real-Time Artificial Intelligence Problems*, Detroit, MI, 1989.

[D'Ambrosio and Fehling, 1989] B. D'Ambrosio and M. Fehling. Resource bounded-agents in an uncertain world. In *Proceedings of the AAAI Symposium on Limited Rationality*, pages 13–17, Stanford, Ca., 1989.

[Doyle, 1988] J. Doyle. Artificial intelligence and rational self-government. Technical Report CS-88-124, Carnegie Mellon University, Pittsburgh, Pa., 1988.

[Georgeff and Ingrand, 1989] M.P. Georgeff and F.F. Ingrand. Decision-making in an embedded reasoning system. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Detroit, Mi., 1989.

[Horvitz, 1987] Eric J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In *Proceedings of the 1987 Workshop on Uncertainty in Artificial Intelligence*, Seattle, WA, 1987.

[Kaelbling, 1988] Leslie Kaelbling. Goals as parallel program specifications. In *AAAI-88, Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 60–65, Saint Paul, Minnesota, 1988.

[Pollack, 1990] Martha E. Pollack. Overloaded expectations. In preparation, 1990.

[Pollock, 1989] J.L. Pollock. Oscar: A general theory of rationality. In *Proceedings of the AAAI Symposium on Limited Rationality*, pages 96–100, Stanford, Ca., 1989.

[Russell and Wefald, 1989] Stuart J. Russell and Eric H. Wefald. Principles of metareasoning. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, Toronto, 1989.

# Planning Emergency Response

B.D. Pomeroy, W.E. Cheetham and D.E. Gaucas

GE Research and Development
P.O. Box 8
Schenectady, New York 12301

## Introduction

AI techniques such as hierarchical planning and transformational synthesis have been applied to tactical and mission planning problems for several military applications. In the Pilot's Associate (PA) program [Smith & Broadwell 1987] it has become apparent that similar techniques may also be required for automating the response to emergencies arising from battle damage or random equipment failures. Furthermore, this requirement extends beyond fighter aircraft to crewstation aids for helicopters, submarines and commercial jet-liners.

We have developed examples of F-16 fighter aircraft emergencies in the context of pilot tasks such as achieving mission goals, reacting to tactical situations, and assessing aircraft status. An equipment failure and the emergency procedure responding to that failure interrupt the normal script of pilot actions by blocking some actions and adding others. Multiple failures can cause even more complex changes to the normal sequence if the faults interact to change not only the normal operations but also the emergency procedures themselves. This paper presents an example emergency script, and describes the problems of script modification and action monitoring relative to a changing pilot environment that have motivated our search for planning-based solutions.

## The Pilot's Environment

We view the pilot's environment in the context of the Pilot's Associate [Lockheed 1988], an advisory system for managing pilot tasks including:

- assessment of the external threat and target environment
- assessment of the internal status of the aircraft systems
- planning the mission route, and replanning in response to changes in threats/targets or to accommodate equipment faults
- planning optimal tactics to achieve mission goals within the constraints of threat/target behavior and the aircraft performance limits
- planning emergency action to correct faults or mitigate their effects

The integration and coordination of such tasks under the dynamic constraints of the environment is a major challenge; the example we present below focuses on the last task of addressing equipment failures relative to the normal operations during an offensive engagement.

When an emergency occurs, pilot response must be consistent with the context, situation and environment variables of the operation. For example, the context is characterized by variables such as the mission phase (e.g., takeoff, cruise, commit, engaged, recovery or landing). The situation is represented by variables such as aircraft altitude, airspeed, bank angle, thrust and attack angle. The environment is captured by variables such as runway accessibility, weather and wingman availability.

## Pilot Procedures

### Representation

Normal and emergency procedures in the pilot domain are currently expressed as and/or trees in a flight manual [General Dynamics 1983] and, in a more abbreviated form, in a booklet strapped to the pilot's leg in flight. These trees are limited by their printed format in that they are brief for clarity and do not cover many contingencies. In particular, these trees are almost useless for dealing with multiple system failures.

Several of these trees have been expanded and encoded for on-line access in the Pilot's Associate program [Lockheed 1988]. We found that computerization of the procedure trees helped to cover a wider range of contingencies. It also provided a better focus of attention for the pilot by hiding information until it was required. However, we also found that the tree format imposed severe limits on the number of faults and contingencies that could be considered, and that it was difficult to express action monitoring in this format.

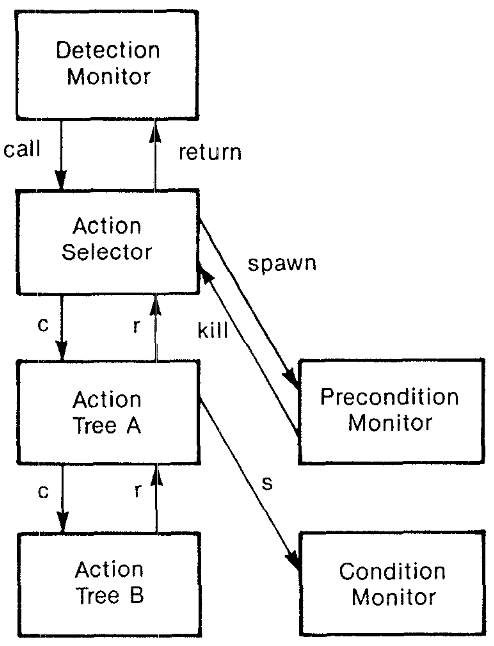
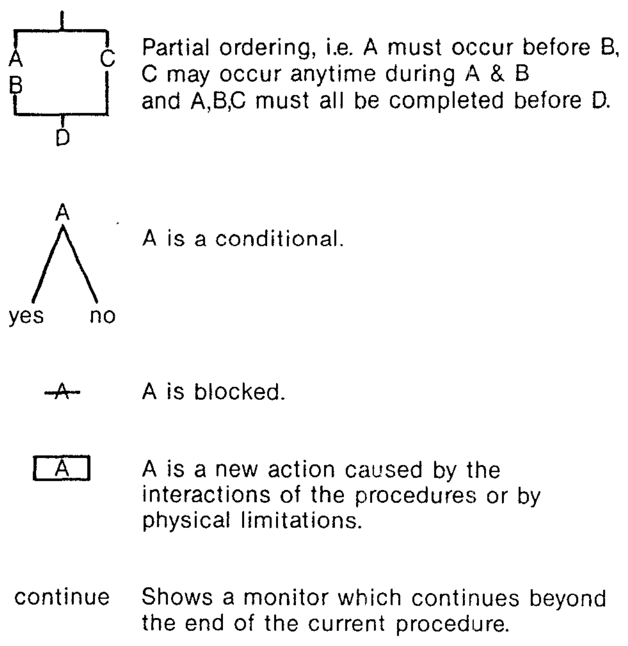


Figure 1: Architecture of Procedures



Partial ordering, i.e. A must occur before B, C may occur anytime during A & B and A,B,C must all be completed before D.

A is a conditional.

A is blocked.

A is a new action caused by the interactions of the procedures or by physical limitations.

Shows a monitor which continues beyond the end of the current procedure.

Figure 2: Tree Notation

We have begun a search for better solutions to this problem of representing actions and control. The initial step has been to develop a library of F-16 emergency procedures which display the various plan and goal interactions observed informally in Pilot's Associate. We extended our conceptual tree language for procedures to include calls to (returns from) sub-trees and spawning (killing) of parallel monitors.

The procedures developed thus far have the architecture shown in Figure 1. There are top level event detection monitors which run continuously and trigger the appropriate action selection tree when a fault occurs such as an equipment failure. The selection trees are classification processes which select an action tree based upon the context, situation and environmental variables. An action precondition monitor is always spawned with an action tree; its purpose is to stop the actions if the context/situation/environment changes. An aborted action restarts the selection process. An action tree may call more specialized action trees and spawn additional condition monitors as required, e.g., during generator failure, a procedure is started to monitor the emergency power unit in case the generator fails again. Parallel execution can occur when actions are monitored during a single fault, and when several action trees and monitors are running in response to multiple faults.

The action trees themselves represent relationships among actions which include partial ordering, conditional branching and preservation conditions. They also capture merged procedures by representing equivalence of two actions, merging of two similar actions with compatible parameter scopes, blockage of an action, and addition of new actions not present in the parent procedures.

Our case studies to date include:

- **single faults** covering procedures for hydraulic, electrical and flight control system failures showing partial ordering of actions, conditional branching and preservation conditions.
- **multiple faults** describing the interactions between the single fault procedures such as equivalence of actions, merging of similar actions with compatible parameter scopes, blockage of actions, and addition of new actions.
- **multiple contexts** showing the backdrop of tactical actions that must be merged with the emergency procedures, e.g., the actions during take-off, offensive engagement, and landing; multiple contexts showing the tailoring of the procedures within the various contexts to suppress inappropriate actions.
- **real-time events** describing the effect of a context change on an incomplete procedure and, similarly, the effects of sequential faults on emergency procedure generation and monitoring.

For this short paper we have selected one example showing the normal operations involved in offensive engagement, the procedure for dealing with a generator failure, and the interactions between the two. Action relationships in this example are expressed using the notation in Figure 2.

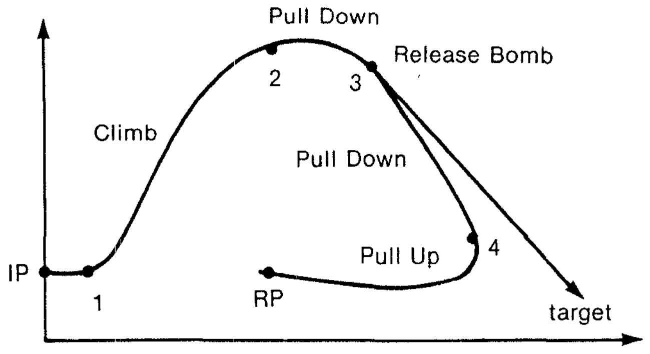


Figure 3: Offensive Engagement

### Normal Operations in Offensive Engagement

An example of a bombing run developed by a Tactical Planner such as the one in Pilot's Associate consists of a low altitude approach to an initial point (IP), followed by a sequence of maneuvers to enable a target hit and terminating with arrival at a rendezvous point (RP). See Figure 3. The actions in this procedure, shown graphically in Figure 4, are defined as follows:

**N1** Begin monitor for threats; if threats appear then change the current context to engaged defensive and replan.

**N2** Begin monitor for arrival at point 1; when point 1 is achieved then alert pilot.

**N3** Set throttle at full power without afterburner.

**N4** Begin monitor for climb angle and speed and arrival at point 2; if deviate from climb path or arrive at point 2 then alert pilot.

**N5** Set throttle at 50% afterburner.

**N6** Move stick to pitch-up into climb.

**N7** Begin electronic targetting and monitor for dive angle and arrival at point 3; if deviate from dive path or arrive at point 3 then alert pilot.

**N8** Set throttle at full power without afterburner.

**N9** Move stick to invert aircraft and pitch-down into dive.

**N10** Release chaff (radar reflective decoy).

**N11** Begin monitor for turn rate and arrival at point RP; if deviate from time constraints or RP is reached then alert pilot.

**N12** Set throttle at 100% afterburner.

**N13** Move stick to roll pitch-up, roll upright, and turn away from target.

**N14** Release chaff.

**N15** Release bombs.

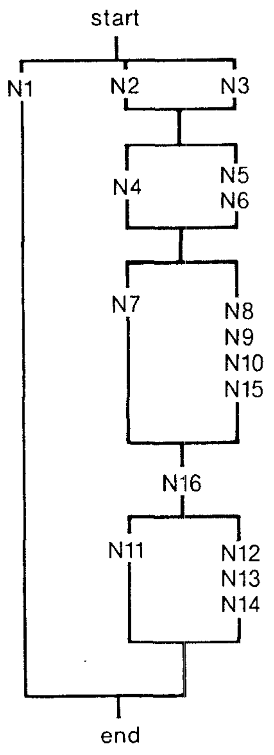


Figure 4: Normal Procedure for Offensive Engagement

**N16** Begin monitor for dive angle and arrival at point 4; if deviate from dive path or arrive at point 4 then alert pilot.

### Generator Failure Emergency

The F-16 has a single electrical generator driven by a power take-off on its jet engine. When this generator fails, an emergency power unit (EPU) driven by an auxiliary gas turbine is started to provide back-up power to most aircraft loads. If the EPU also fails, the flight and engine controls will continue to function on battery power so the pilot can land the aircraft.

Pilot response to a generator failure depends upon the current context, situation, and environment as determined by the action selector in Figure 1. The action tree considered here is appropriate for the engaged offensive context, Figure 5. Some conditional branches and sub-trees for other contexts have been suppressed for clarity. The actions Figure 5 are defined as follows:

**G2** Advise pilot of loss of main generator.

**G7** Advise pilot to limit angle of attack to 12 degrees and bank angle change to 90 degrees maximum, and avoid rapid roll rates.

**G25** Warn pilot that leading edge flaps may be locked and will require reset.

**G26** Begin or continue monitor for repeated failures of main generator; if begin then repeat=1 otherwise repeat=repeat+1. (This monitor continues even after the generator procedure is completed.)

**G27** GOTO start EPU script.

**G28** Set EPU switch to ON.

**G30** Begin monitor for continued EPU operation; if EPU fails then abort main generator procedure and replan for double failure of generator and EPU.

**G31** Set main power switch to BATTERY then to MAIN POWER.

**G33** Advise pilot to check whether main power switch is in MAIN POWER position.

**G37** Set EPU switch OFF the set to NORMAL.

**G40** Press servo electric reset switch. (Resets leading edge flaps control.)

**G42** When speed is subsonic set throttle to mid-range then set electronic engine control backup control (EEC BUC) switch to OFF, and then set switch to EEC.

**G43** When speed is subsonic then set afterburner (AB) switch to AB RESET, and then set switch to NORMAL.

**G44** Advise pilot that all electrical systems are back to normal.

**G46** Alert pilot to systems lost, i.e., lights, master power switch, fuel pumps, fuel feed control, code transmitter, missile launchers, electronic bomb sighting, electronic countermeasures, chaff.

**G47** Alert Tactical Planner to unavailable resources.

**G50** Warn pilot not to retard throttle below full power until subsonic speed is achieved, to avoid engine stall.

**G54** Alert pilot: chaff and electronic bomb sighting unavailable.

**G56** Alert pilot to failure of EPU, and declare EPU failure to the emergency planner database. (This will trigger the detection monitor for EPU failure.)

**G57** (Null action for readability.)

**GQ32** Is the main power switch in the MAIN POWER position?

**GQ34** Is main generator back on-line, i.e., is main generator light green?

**GQ35** Is this the first time the main generator has failed, i.e., is repeat=1?

**GQ41** Is engine model PW200?

**GQ42** Is engine model PW220?

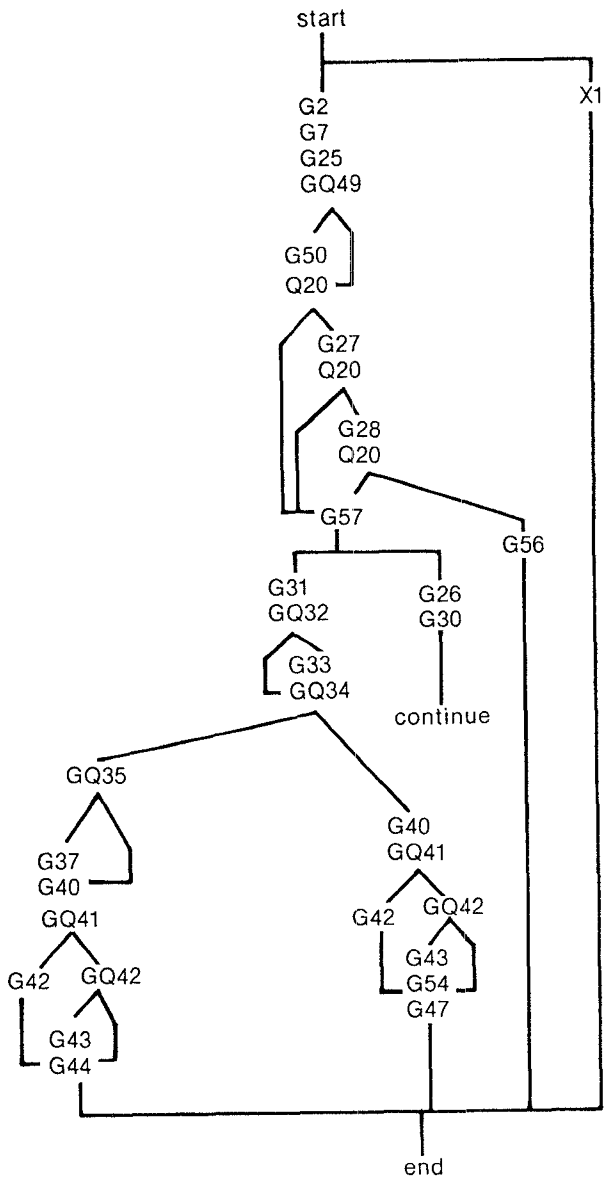


Figure 5: Emergency Procedure for Generator Failure

**GQ49** Is airspeed supersonic?

**Q20** Is the EPU running?

**X1** Begin monitor for fault and context; if fault is corrected or context changes then abort the main generator emergency procedure.

## Merging Normal Operations and Emergency Response

Assume that the generator fails just as the pilot reaches the IP, and that he has a few moments during his transit to point 1 in which to attempt a fix. The merged procedure, Figure 6, shows that the pilot should try to accomplish the entire generator procedure before starting his climb. If the generator reset were successful, then he could proceed with the mission as before. However, if the reset fails, then he has lost power to his weapons computer which blocks the electronic targeting action, N7, and he has lost his chaff dispenser which blocks N10 and N14. There is no substitute for N10 and N14, but he can shift over to manual targeting, N17, for the bomb release. N17, a new action not present in either of the parent procedures, is defined as follows:

**N17** Begin manual targetting monitor for dive angle and arrival at point 3; if deviate from dive path or arrive at point 3 then alert pilot.

If the generator fault occurs later in the run, it changes the context to disengaged, i.e., the Tactical Planner aborts the attack.

We also have considered the multiple emergency example of generator and hydraulic failure in the context of mission recovery. This example displayed the additional concepts of equivalent actions and merged actions not shown here.

## Conclusion

Providing automated emergency response for advanced crewstations, whether in military or civilian applications, will require the ability to merge scripts of actions and to reason about interactions over time. This modification of procedures must be relative to multiple goals in a semi-predictable world where fast reactivity is often important, but where strategic planning is also needed to avoid resource depletion and other unrecoverable mistakes.

The rules for combining two or more procedures are emerging slowly from our pilot domain examples; however, it is still not clear how these combinations should be accomplished in all cases. Clearly the parent procedures need to be more rigidly structured into segments such as fault alerts, critical warnings, corrective actions, and Tactical/Mission replanning alerts, so that actions can be merged in groups according to their importance to the pilot. However, additional mechanisms are needed to handle unforeseen interactions between procedures such as blocked actions and new actions.

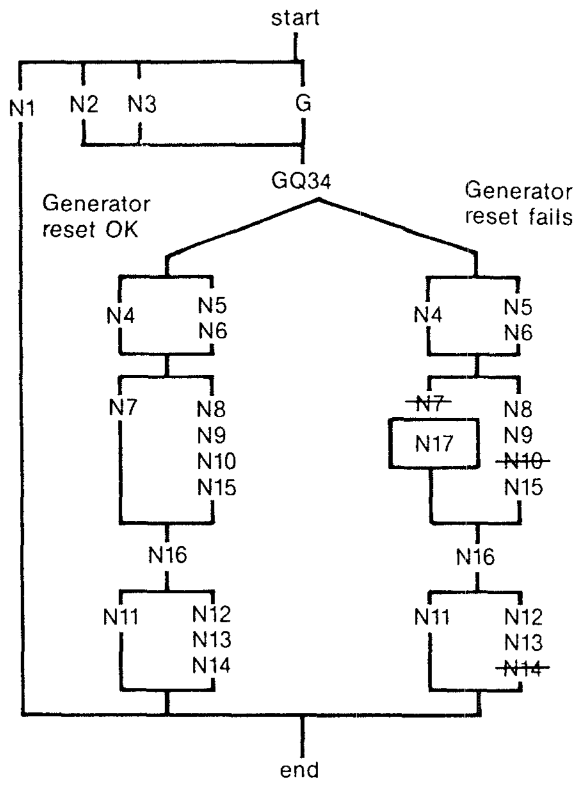


Figure 6: Merge of Offensive Engagement and Generator Emergency (node G refers to entire procedure in Figure 5)

## References

[General Dynamics 1983] Flight Manual USAF/EPAF Series Aircraft F-16 A/B, Technical Order 1F-16A-1, General Dynamics, Fort Worth Division, change 3, 11 July 1983.

[Lockheed 1988] Phase 1 Interim Report of the Pilot's Associate Program, Document PA-INT/RPT, Contract No. F33615-85-C-3804, Lockheed Aeronautical Systems Company, Georgia Division, Marietta, GA, 5 Aug. 1988.

[Smith & Broadwell 1987] Smith, D. and Broadwell, M. Plan Coordination in Support of Expert System Integration. *Proceedings of DARPA Knowledge-Based Planning Workshop,* Austin, Texas, Dec. 1987.

# Plan Fields and Real-World Uncertainty

**Neil C. Rowe**

*Code 52Rp, Naval Postgraduate School*
*Monterey, CA 93943 USA*

Planning can be considered a many-to-one mapping from the space of possible problems to the space of possible plans. For most interesting problems, the mapping is too complicated to formulate other than procedurally. But we can do it for means-ends analysis, where the notion of the difference between the current state and a goal state determines the next planning action. Focusing on differences makes means-ends analysis a step-at-a-time planner, so even when results unexpected by the planner arise from actions, a solution can still usually be found from the new unexpected states.

Our recent research on robot navigation has tried to find analogous ideas for planning with real-valued quantities like position and orientation of a robot. Our approach has been to define "path fields" over the configuration space for the robot, "fields" in the physics sense of electrical or gravitational ones but now extended to plans as well as vectors for every point in the space. For example, for high-level robot path planning to a particular goal point in some terrain, we can define a "path field" at every point in the terrain which indicates the best path or best direction to head from there to reach the goal point. Then if we accidently wander from the path planned when trying to follow it in the real world, or unexpected accident forces us from it, we can just reassess where we are and look up the best way to go from this new location, analogously to means-ends analysis.

Figure 1 (from (Alexander 1989) which in turn uses the path-planning program of (Rowe 1990)), shows an example path-direction field for two-dimensional terrain with an obstacle (the checkered triangle) and a crossable but costly "river" (the bumpy vertical line in the lower left), where all non-obstacle and non-river areas are considered equal-cost-per-distance to traverse, and where the goal is the small circle on the right just below the middle. The small lines indicate the direction of the optimal path to the goal point from evenly-spaced start points. The smooth thin longer lines indicate boundaries between abrupt changes in optimal-path behavior, determined by methods that will be explained later; for instance, the gently curved line in the lower left distinguishes optimal-path behavior that crosses the river from optimal-path behavior that avoids the river by going around its endpoint.

Construction of plan fields that would tell you what to do next from any state would seem to be a difficult or impossible issue for many problems, because nontrivial search spaces are either enormous or infinite. For instance, robot navigation involves real numbers, so there are an infinity of states. One approach is to impose a regular tessellation on the space of possible states (for a robot, the terrain) and then associate every point within a particular cell with the same field direction (next behavior) as that of the center of the cell. This heuristic approach will not always work, as when

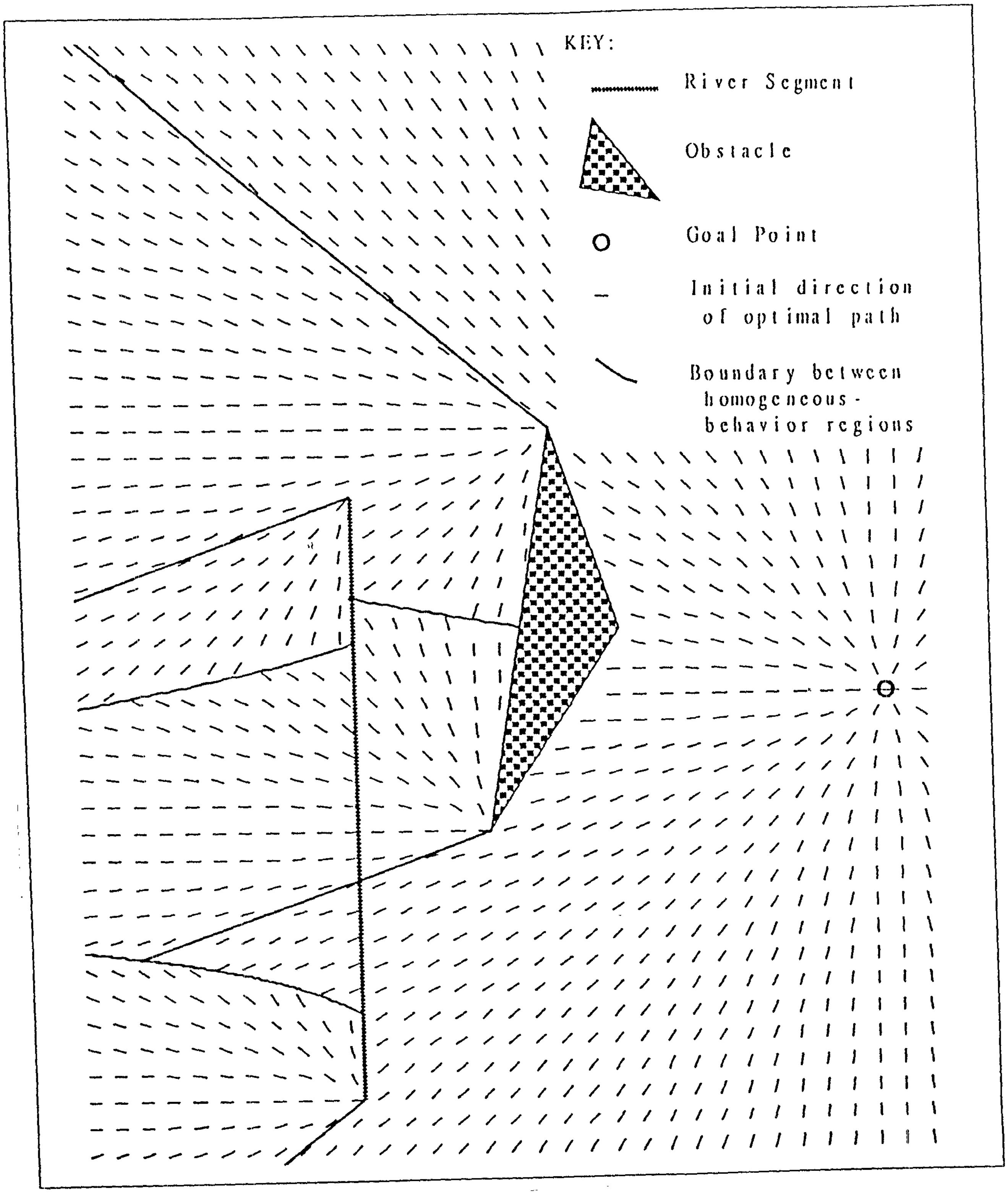

KEY:
River Segment
Obstacle
Goal Point
Initial direction of optimal path
Boundary between homogeneous-behavior regions

Figure 1

Example Optimal-Path Map

an impassable obstacle crosses a terrain cell. Another approach is to define irregular cells of intuitive meaning like the terrain bounded by a ridge and two roads, but this requires expert judgment, and cannot be relied on to give regions of similar path behavior.

Fortunately, a rigorous alternative exists based on terrain-partition algorithms for path planning such as (Rowe and Lewis 1989), (Rowe and Richbourg 1990), and (Rowe and Ross 1990). These algorithms find an optimal path from a start point to a goal point by partitioning polygonally-modeled terrain (or polyhedrally-modeled airspace, for the three-dimensional program of (Rowe and Lewis 1989)) into equivalence classes of points whose path behavior to the goal is fundamentally similar. We can extend these algorithms to find paths from everywhere in the terrain to that same goal point, since they recursively decompose paths to the goal point into "wedges" or "corridors" of similarly-behaving paths, eventually decomposing terrain into a set of "well-behaved path subspaces" whose behavior can be summarized by the sequence of terrain features it encounters. Each wedge is composed of irregular subregions between features, which can be considered "behavioral regions", equivalence classes of points that cross exactly the same terrain features in the same order on their way to the goal. A path field is the mapping from points to their associated behavioral regions and hence to a sequence of terrain-feature crossings. Note that finding the exact optimal paths is then straightforward with homogeneous-cost-per-unit-distance regions, because the paths can only turn on the boundary of cost regions, and those turns must obey Snell's Law.

Thus the seemingly infinitely variable plan fields for a real-valued multi-dimensional space can be reduced to a finite number of "behavior classes" narrow enough in definition so that once a situation's class is known, computing the optimal plan for the situation is straightforward. Thus behavior classes are a kind of problem abstraction. However, the whole trick is determining the class of a situation, because such spaces need not have "nice" shapes. Thus much of our recent work, exemplified by (Alexander 1989) and (Rowe and Ross 1990), has tried to develop mathematical methods for finding behavioral boundaries, since once these are found, the region shapes follow. The methods of the last paragraph are not sufficient, because generally the wedges found will overlap, so new theorems must be developed. For instance, for Figure 1 we can prove that the boundaries not emanating from obstacle or river endpoints are hyperbolas; for "weighted regions", regions with homogeneous cost-per-unit distance, boundaries also include parabolas and a variety of curves describable only by parametric functions.

Path fields and plan fields can also address a more fundamental uncertainty, state and world-feature uncertainty, that means-ends analysis and kindred planning techniques cannot directly address. If we can create separate plan fields for possible world situations, we could combine them into a consensus plan field. In other words, we could draw an analogy to the methods of superposition of electrical fields of multiple charged objects by calculating the fields separately for each object and adding the field vectors at every point. For instance in the robot path-planning problem, suppose we are not sure where or whether a bridge crosses the river in the vicinity. We could create a separate path field for each reasonably possible siting of the bridge, then take the weighted average of the initial vector directions at every point in the terrain, based on our degree of belief in the likelihood of each possibility. This would be best when we are not close to the river, but otherwise we would probably be able to see any bridges and then create a new plan field. Such an approach is appealing, but it is easy to forget that it is heuristic and has no formal mathematical basis: for one thing, path fields are not even continuous. However, this idea does have appealing analogies to recent ideas for robot subsumption architectures like those of Rosenblatt and Payton at Hughes, for which similar "higher-level" plan ideas can be weighted against lower-level concerns like avoidance of immediate obstacles.

Reaching a consensus between the recommendations of competing plan fields is actually a problem of multiple inheritance at each point in the plan space, and many standard approaches are applicable. If one plan inherited subsumes all the others, then there is no conflict and that more general plan can be used. We could set priorities among fields that depend on location in the field, giving a consensus field consisting of pieces of the component fields stitched together. That would be appropriate if plans cannot be "diluted" without damage; for instance, if you're not sure whether an obstacle is blocking the road at a point P, you should still slow down when you approach P. If the plans differ only quantitatively, then some sort of weighted averaging can be used, where the weighting is their probabilities, as in (Rowe 1982). We may be able to guarantee even a result that does not subsume the others. If a vector field is truly an "optimal-path field", besides having the field direction represent the best path direction, we can let its vector magnitude represent the difference between the cost of going in this direction versus the cost of going in the second-best locally-optimal direction at that point. Then the vector magnitude is the relative goodness of the recommended direction at that point. If the total weight in some direction is more than the sum of weights in all other directions, it is guaranteed that that is the best direction because the other effects could never cancel it out.

A multiple-inheritance consensus between possible world situations requires that we compute complete path fields in advance. But frequently the terrain itself restricts possibilities to a finite set, like places for a bridge or ways to cross a mountain ridge. We can also exploit experiments. For instance, if we are not sure how far a forest extends to the right of us, we can pick evenly-scaled extents for the forest and derive path fields for each. If in two path fields the direction is the same, at corresponding points, it must be the same (with a few known exceptions) for any "intermediate" field in which the extent of the features is intermediate between the extents in the two original fields. So when we find two path fields whose vector directions are point-for-point nearly identical, we do not need to store any intermediate fields between them. Furthermore, even when two sample fields have significant differences, we need only study the areas in which those differences occur to obtain intermediate fields between them, which saves much time.

## Acknowledgements

This work was supported in part by the U.S. Army Combat Developments Experimentation Center under MIPR ATEC 88-86. Also in part this work was prepared in conjunction with research conducted for the Naval Air Systems Command and funded by the Naval Postgraduate School.

## References

R. Alexander, "Construction of optimal-path maps for homogeneous-cost-region path-planning problems", Ph.D. thesis, Dept. of Computer Science, U.S. Naval Postgraduate School, September 1989.

N. C. Rowe, "Inheritance of statistical properties", Proceedings of the National Conference, AAAI, Pittsburgh, PA, August 1982, 221-224.

N. C. Rowe, Roads, rivers, and rocks: optimal two-dimensional route planning around linear features for a mobile agent. To appear in *International Journal of Robotics Research*, 1990.

N. C. Rowe and D. H. Lewis, Vehicle path-planning using optics analogs for optimizing visibility and energy cost. *NASA Conference on Space Telerobotics*, Pasadena CA, January 1989.

N. C. Rowe and R. F. Richbourg, An efficient Snell's-law method for optimal-path planning across multiple two-dimensional irregular homogeneous-cost regions. To appear in *International Journal of Robotics Research*, 1990.

N. C. Rowe and R. S. Ross, Optimal grid-free path planning across arbitrarily-contoured terrain with anisotropic friction and gravity effects. Accepted to *IEEE Transactions on Robotics and Automation*, January 1990.

# The Dimensions of Knowledge Based Control Systems and the Significance of Metalevels

**Marcel Schoppers and Ted Linden**

Advanced Decision Systems, 1500 Plymouth Street, Mountain View, CA 94043

## Abstract

We argue that knowledge based control systems (KBCSs) can be viewed as making tradeoffs between four dimensions: response time, program size, processing power, and inattentiveness. As a system's rating on any one of these axes approaches zero, its rating on one of the other three axes must increase. We characterize various planning and metareasoning research thrusts as exploring different regions in the four dimensional tradeoff space. We then define metacontrols and show that they hold a central position in KBCS design, potentially allowing a system to dynamically reposition itself anywhere within the reachable volumes of the tradeoff space.

## Dimensions of Control

Knowledge based control systems (KBCSs) have numerous applications, including automated vehicles for land, air, water and space; intelligent assistants for pilots and submarine commanders; factory automation and robots in general; process control systems; and environment management systems (on space stations, for example). A general requirement of control systems is that they must select and execute a response to a stream of events and must do so in a sufficiently short time to keep the controlled system or "plant" within prescribed boundaries. The nature of the prescribed boundaries need not concern us here; the main point is that they usually imply constraints on the amount of real time a system may take to select and execute its responses.

The field of Computer Science has long known that, although there are a large number of ways of implementing any given function, faster implementations tend to need more memory space -- the so-called time/space tradeoff. To make this idea a little more precise and more directly applicable to control systems, we do the following:

- We regard a control system as a function that maps possible courses of events to possible courses of action. Courses of events might be regarded as strings of symbols, with symbols representing sets of events in the world. Courses of action might be regarded as strings whose symbols represent sets of actions.
- We take "real time" to refer to the elapsed time delay a system requires in order to map a particular course of events into an appropriate course of action that depends on those events.
- We take "processing time" to refer to the number of CPU instruction cycles a program requires in order to map a particular course of events into an appropriate course of action that depends on those events.
- We take "processing power" to refer to the number of CPU instruction cycles per second that a hardware system can deliver.
- We take "program space" to mean the amount of computer memory required to store both the control system program and whatever data it maintains.
- We take "attentiveness" to mean the extent to which the system takes all available data into full account in all its processing. The converse concept is "inattentiveness", which manifests itself in action parameters that are not computed and in input data that is ignored or underutilized.

On the time issue, two further comments are needed. First, both of the time measurements just described are implicitly subscripted by the course of events whose response time is being measured. If we added together all the (real or processing) times required to map each possible course of events to the corresponding course of action, we would have the total (real or processing) time required to compute a complete lookup table for the function being realized by the system. Second, neither concept of time pays the slightest attention to the time required to build the system. When it comes to measuring system response, execution time is all that matters; the time spent on system design, programming and compilation are irrelevant.

It is now easy to see that all real-time control systems can be mapped into some point in a four-dimensional coordinate space whose axes are **1)** real time (per response), **2)** processing power, **3)** program space, and **4)** inattentiveness.

Suppose we ignore system inattentiveness for a moment, and suppose our system is in fact a giant lookup table that maps courses of events into courses of action. Such a system could be exceedingly fast (low real time per response) but also exceedingly demanding of program space. In order to reduce program space we could replace parts of the lookup table with some amount of computation (processing time). As program size shrinks, the amount of processing time required to reconstruct the missing entries goes up. Increased demands for processing time can be dealt with in two ways: either the amount of real time required to compute a response can go up, or the system's processing power can go up (meaning either faster CPUs or more CPUs).

Lastly, we can factor in the dimension of inattentiveness. If the system can ignore some data or if it can simply leave some decisions unmade, then it can clearly get away with less computing. But the closer the inattentiveness dimension is pushed toward zero, i.e. toward complete processing, the more the system must suffer increases in either real time required, processing power required, or program space required.

Thus the four axes of real time, processing power, program space and inattentiveness are mutually constraining. For any given control system there is some volume of that four-dimensional coordinate system that the system can never reach, because reductions along one dimension inevitably require increases on the other dimensions.

Relevant work in control systems and artificial intelligence can be characterized as exploring various points on the four coordinate axes described above. "Reaction plans" [8] are programs that cover a wide variety of possible situations by being highly conditional, and they achieve very rapid execution because most of the necessary planning is done off-line, at system design time. Classical planning, on the other hand, needs to have a particular initial state from which to begin planning; in other words, the classical planning approach is to do the planning at run time. Because classical planning came first in AI, the reaction plans approach is regarded as being impossibly demanding of program space [2]. But of course, reaction plans -- and indeed the vast majority of control systems -- have merely chosen to minimize real response time at the expense of an increase in program space [8].

Among AI practitioners, an alternative approach to real-time computing is to leave unchanged the nature of the plan being constructed and to work instead on making reasoning systems adaptible under time pressure (c.f. anytime algorithms [1] and the application of blackboard systems to realtime problems, e.g. several papers in [3]). The tradeoff here is clearly between the axes of real response time and inattentiveness -- as response time comes down, inattentiveness must go up.

When real time systems builders show interest in parallel hardware, they are considering a decrease in response time at the expense of an increase in processing power (i.e. special purpose processors or more processors).

Given that the four tradeoff axes represent degrees of freedom in the design of realtime systems, there should be no sense of competition between the various research efforts. In the design of the knowledge based control systems of the future, the issue will not be which approach is best, but at what point on the four tradeoff axes the target system should be located. The goal of future control system designers might be phrased as follows:

> Subject to constraints on hardware cost and system reliability -- these implying constraints on response time, processing power and inattentiveness -- construct the smallest suitable program.

## The Notion of Metacontrol

The more a system is subject to demanding real-time constraints, the less its responses can be computed on demand, and the more its responses must be reduced to table lookup. Unfortunately, gaint lookup tables tend to be unintelligible to human cognitive processes. Hence there may well exist applications whose complexity and realtime demands are so severe that the human mind is incapable of understanding the appropriate program.

That this possibility is not merely speculative is evidenced by experience with chess playing programs. For some chess endgames it is possible to construct a complete table of legal positions and to associate each position with a move that either maximally advances or maximally delays a victory. That table amounts to an optimal and extremely fast chess player. Just such a table was constructed, by Kenneth Thompson of Bell Telephone Laboratories, for the endgame King and Queen against King and Rook. With the exception of a few special positions, this endgame is known to be a theoretical win for the Queen's side. Thompson's lookup table machine played the Rook's side and always played the move that maximally delayed the Queen's victory. At the 1977 Toronto meeting of the IFIP, Thompson invited two International Masters, Hans Berliner and Lawrence Day, to play the Queen's side against the machine. Although the Masters were at first secure in the belief that the endgame was theoretically theirs, and although every position with which they were confronted seemed to be a winning position, they were embarrassed to find that they could make no headway against the machine. They found the experience upsetting and earnestly wished to find out how the machine was making its inexplicable escape time after time. There was no explanation other than a lookup table of some three million entries. (This episode was reported to the authors by Donald Michie.)

More recently, work on the endgame King and two Bishops against King and Knight, comprising a hundred million positions, has shown that although that endgame was generally believed to lead to a draw, it is in fact a win for White from all but a few freak positions; and even when a chess Master knows this and is supplied with examples of optimal play, six months of study is not enough to make the machine's strategy intelligible to humans (the experiment is reported in [7]). We conclude that if realtime constraints can be so severe as to demand a lookup table implementation of a control system, then there exist control systems that are unintelligible to, and hence inconstructible by, unaided human cognition.

Nevertheless, such control systems can be built by other programs. As with Thompson's lookup table machine, it may be easier to specify how the control system should be built than it is to build the control system directly. Such system-building programs are instances of the AI notion of metalevel reasoning and, as reported above, can open up reaches of the four-dimensional space of possible control systems that were previously inaccessible to humans. (It is a matter deserving serious ethical consideration whether such unintelligible control systems should ever be turned loose on real applications [6].)

In AI, the "meta" relation is loosely defined as follows: "meta-X" is "reasoning about X". A moment's thought shows that this intuitive definition is quite useless as a filter of what is or is not meta. What qualifies as "reasoning"? Our main motive for trying to define the idea more clearly is that even the intuitive ideas of "meta" and of "planning" imply that "planning" is "meta" to plan execution or action. From this it follows that the "meta" relation is the primary link between AI research and control systems work. That being so, clarity in what it means to be "meta" will serve both fields.

> Activity X is a "metacontrol" for activity Y when X allocates or deallocates processing power for Y by performing operations upon a data structure that represents Y.

Several remarks are in order.

- Because of our interest in planning and control, we are interested in the use of "reasoning" to control computation and hence have left metatheory aside.
- Unlike others who have considered metalevels, we do not find the idea of representation problematic. We are content to leave to the system implementer the mode of representation to be used. We require only that the relevant aspects of Y and the data structure representing Y must always be in alignment under that mode of representation.
- Our definition makes very clear that metalevel activities are about the allocation of a limited resource: processing power.
- The requirement that X's influence should be exerted upon Y by means of a data structure serves to distinguish X's metalevel activity from conditional branching within Y.

- Metalevel 0 could consist (for example) of numerous control laws awaiting an opportunity to drive a robot's physical effectors. In that case, metalevel 1 would be selecting the particular control laws to compute at each moment.
- Planners are a special case of metacontrols: their outcome, a plan, is a data structure that allocates processing power and other resources to specified activities over an extended period of real time. The same is true of programming and programs.
- A definition that prevented an operating system from being "meta" to every program it scheduled would also rule out most agenda managers and blackboard systems, although the latter are among the foremost examples of metalevel reasoning.
- The "meta" relation may be transitive, reflexive and symmetric -- there are no restrictions on which activities may be "meta" to others.
- It follows from our definition that X is *not* a metacontrol for Y if X only determines what goals Y should attempt to achieve. X would not be allocating processing power; it would only be passing a parameter to Y.
- A more borderline case arises if X is constraining the set of activities available for Y to execute, e.g. when Y is a planner and X is deciding what set of actions Y may utilize. In this case we deem X to be a metacontrol for the activities it is ruling in or out, but X is not a metacontrol for Y.

## The Significance of Metalevels

The analysis of systems in terms of metalevels makes clear exactly how knowledge based control systems (KBCSs) will extend both control systems and AI work. KBCSs are distinguished by possessing a "controls level" and at least one metalevel. The controls level contains the system's sensor and effector capabilities, and distinguishes the system from a pure planner. The metalevels are concerned with controlling the activities of the levels below, and distinguish the system from a pure control system. The first metalevel may or may not be a planner, but it must have at least an implicit model of the effects and requirements of the activities at the controls level.

Now let us consider how metacontrols can contribute to the tradeoffs discussed earlier. Processing power is likely to be fixed at design time, leaving the system no control over its rating on the processing power axis. The system can however make dynamic adjustments in its rating on the other three axes, suggesting that it might dynamically adjust the tradeoffs between **a)** response time and program size, **b)** response time and inattentiveness, and **c)** program size and inattentiveness. We now show that metacontrols are important in all three tradeoffs.

The possible goals/setpoints, world states, and controls level activities may be so numerous that anticipating all the possible combinations is infeasible on account of program size (not to mention programming complexity). Nevertheless, the system can be redesigned to construct, at run time, only those combinations that are actually required. In making that redesign, the designer is trading off reduced program size against increased processing time, and is moving some of the system's functionality from the controls level (metalevel 0) into a planner (metalevel 1).

If a system's input data rate can vary dynamically, the system might be designed to adapt to changing loads and circumstances by dynamically deciding either to delay its responses or to ignore some information or implications. This tradeoff between response time and inattentiveness is a matter of where to allocate processing power, and is by definition a metalevel issue. A most interesting discussion of this use of metalevels is given in [4].

Situations having exceptionally urgent response deadlines can be responded to either by caching parts of the appropriate response or by ignoring other aspects of the situation. This tradeoff between program space and inattentiveness will in practice depend on how critical it is to be confident about details of the situation, how critical it is to have a precise response, how serious the damage might be, and so forth. The general issue is, knowing when it is worthwhile to cache the results of a computation. The activity of caching is "meta" to the responses being cached and may be left to the system (if that is safe).

We have now shown that each of the dynamically adjustable tradeoffs are issues of metalevel functionality. It follows that in control system design, AI technology is applicable whenever a system must vary its own rating on the tradeoff axes.

AI technology is also applicable as an antidote to the difficulty of programming complex control systems.

There are two obvious uses for AI technology even when a system's position in the tradeoff space is static: **a)** the complexity of programming everything at the controls level may be worse than the complexity of programming the same competence by distributing it across several metalevels; and **b)** the system must be able to dynamically extend its model of the domain and must be able to adapt to those extensions. The latter problem is being attacked by AI learning research.

Incidentally, it is *not* necessary to use AI technology in a control system merely because the system's functioning depends on information that can only be obtained after the system has been fielded. Control systems always depend heavily on run-time information. The real issue is the metalevel at which that information is used. AI based approaches are necessary only if the information must be used at metalevel 1 or higher, and that is arguable only if one of the reasons listed above applies.

## Conclusions

We have argued that knowledge based control systems (KBCSs) can be viewed as making tradeoffs between four dimensions: response time, program size, processing power, and inattention. As a system's rating on any one of these axes approaches zero, its rating on one of the other three axes must increase. We characterized various control, planning, and metareasoning research thrusts as exploring different regions in the four-dimensional tradeoff space, and suggested that all have a role in the technology of KBCS design. We then defined metacontrols and argued their importance, both in potentially allowing a system to dynamically reposition itself within the tradeoff space and in simplifying the construction of complex control systems.

It was pointed out to us by Mike Fehling that at least three of the axes of our tradeoff space are objectively quantifiable. We wish to encourage the use of empirical measurements in comparisons between AI systems, and cite [5] as being both relevant and exemplary.

## References

[1] DEAN, T.L. AND BODDY, M.
An analysis of time-dependent planning.
In *Proc AAAI*, pages 49-54. 1988.

[2] GINSBERG, M.
Universal planning: an (almost) universally bad idea.
*AI Magazine* 10:4:40-44, 1989.

[3] JAGANNATHAN, V., DODHIAWALA, R. AND BAUM, L. (editors).
*Blackboard Architectures and Applications.*
Academic Press, 1989.

[4] LESSER, V., PAVLIN, J. AND DURFEE, E.
Approximate processing in real-time problem solving.
*AI Magazine* :49-61, Spring 1988.

[5] MICHIE, D.
A theory of advice.
*Machine Intelligence 8.*
Ellis Horwood, Chichester, England, 1977, pages 151-168.

[6] MICHIE, D.
Computer chess and the humanization of technology.
*Nature* 299:391ff, 1982.

[7] MICHIE, D. AND BRATKO, I.
Knowledge synthesis with respect to the KBBKN chess endgame.
In *Proc Internat'l School for the Synthesis of Expert Knowledge (ISSEK) Workshop (Bled, Yugoslavia).* The Turing Institute, 36 N Hanover Street, Glasgow G1 2AD, Scotland, Aug 1986.

[8] SCHOPPERS, M.
In defense of reaction plans as caches.
*AI Magazine* 10:4:51-60, 1989.

# Robust Behavior with Limited Resources

**Reid Simmons**

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

**Abstract**

We address the problem of mobile robots that operate with multiple goals in uncertain, changing environments, but have limited sensors and computational resources. Such systems need to use their resources to best advantage, trying to achieve the tasks at hand, while maintaining reactivity. We describe how the Task Control Architecture (TCA) is being used to provide such capabilities by taking advantage of the hierarchy, concurrency, and focus of attention inherent in the robot's tasks.

We are interested in mobile robots with multiple, often conflicting, goals that operate in uncertain and changing environments. Such systems should be goal-directed, but must also be reactive to changes that signal errors or opportunities.

A prevalent approach is to build robots that continually monitor all (relevant) aspects of the environment, essentially using stimulus-response rules to decide what to do next (e.g., (Brooks 1986), (Kaelbling 1986)). To get the robot to perform a new task or react to a new contingency typically involves adding more sensors and/or processors. While this approach has the advantage of being very reactive, it does not scale well as the tasks become more complex and more numerous. For any given robot it is only a matter of time before we run out of space or power for new sensors and processors.

In contrast, our approach tries to create a robust, reactive system that can handle multiple tasks in spite of the limited sensors and processors of the robot. To succeed, our approach tries to take full advantage of the resources that the robot does have. This includes using hierarchical coarse-to-fine control strategies, using concurrency whenever feasible, and explicitly focusing attention on the robot's tasks and monitored conditions.

Our approach is also largely empirical , to date using two robot systems. The CMU Planetary Rover is a six-legged walker designed for navigation and sample acquisition in outdoor, rough terrain (Bares et al. 1989). While still under construction, we have implemented a system that integrates perception, planning, and mechanism control to "walk" a single prototype leg through an obstacle course (Krotkov, 1990).

The main testbed for developing these ideas is an indoor, mobile manipulator based on the Heathkit HERO 2000 (Lin et al. 1989b). Sensors include an overhead camera for 2D vision and sonars mounted on the robot's torso, base, and wrist. The HERO's tasks include spotting and collecting small objects from the floor of the lab, picking up printer output, delivering objects to people at workstations, and recharging when its battery gets low. The robot also tries to avoid collisions with people and other objects.

## The Task Control Architecture

Both our testbeds use the Task Control Architecture (TCA) to control the robot's actions, manage its resources, and monitor the robot and its environment (Simmons &

Mitchell 1989). TCA is a distributed system with centralized control that provides mechanisms to construct and manipulate hierarchical plans, to allocate and manage user-defined resources, to monitor selected conditions, and to handle exceptional conditions.

At the base level, TCA provides facilities for connecting processes to the central control and for sending messages between processes. All messages are routed through the central control, which decides when to handle messages, and which processes will handle them.

Various types of messages are defined on top of this base level. For example, *query* messages enable processes to obtain information, such as current sensor data, from one another.

*Goal* and *command* messages are used to create hierarchical plans, which TCA represents using a *task tree*. Whenever a goal, command, or monitor message message is issued, TCA adds a node in the task tree as a child of the node that issued the message. For example, the message handler for the "collect cup" goal issues sub-goal messages to approach the cup, line up with it, etc. These, in turn, issue other messages, eventually bottoming out in messages to execute commands and monitor the systems' progress.

Besides goal/sub-goal relationships, the task tree also encodes temporal constraints between nodes. Processes can add temporal constraints such as that one goal must be achieved before another goal can start (sequentiality), or that one goal cannot be planned out until it is ready to be achieved (deferred planning). For example, the system can add temporal constraints to indicate that while the robot must grasp the cup before going to the bin, it is free to *plan* how to approach the bin any time after it has arrived near the cup.

The task trees also contain *monitor* messages. Monitors are condition-action pairs that indicate how to react to change in a sensor. TCA provides both synchronous, polling monitors and asynchronous, demon-invoked monitors. For example, the HERO uses a polling monitor to check its battery level every so often, and uses a demon monitor to notify the system if a cup-like object is found, which is activated whenever an image has been processed.

TCA also provides mechanisms to handle exceptional conditions, such as those detected by monitors. Processes can attach exception handlers to nodes in the task tree. When an exception is raised, TCA searches up the task tree to find a handler for the exception. Exceptions are typically handled by manipulating the task tree, such as killing sub-trees, adding new nodes, or resending messages. For example, when the HERO notices a blocked path, the sub-tree representing the path plan is killed and the original "go to" message is resent, causing the system to replan the path.

Another task of TCA is to manage the robot's resources. TCA provides facilities for defining hardware and software resources and for associating message handlers with resources. TCA regulates the flow of messages to resources to prevent conflicts. For example, by defining the camera to be a resource, TCA will ensure that only one process can access the camera at any one time. On the other hand, if the robot's drive motors and its internal sensors are declared to be separate resources, information about the sensors can be obtained while the robot is moving.

## Dealing with Limited Resources

The TCA mechanisms have been used to build robot systems that can operate robustly with limited resources. In implementing the HERO and Rover systems, several simple, but effective, organizing principles have become apparent: 1) take advantage of hierarchy in plans, monitors, and exceptions, 2) make full use of the concurrency available in the tasks, and 3) maintain an explicit focus of attention for tasks and monitors. Currently, these principles are used by humans to design the system, but we hope to eventually have the robot use them to control itself.

## Hierarchy

An effective organizing principle is to take advantage of hierarchy inherent in the tasks. For example, plans can be generated much more efficiently using hierarchical planning than by considering all the details at once. Also, by using hierarchical planning the system can plan only to the level of detail warranted by its current knowledge of the environment. This capability is supported by TCA's temporal constraint mechanism. For example, the HERO defers planning how to pick up the cup until it gets near enough to determine what kind of cup it is. On the other hand, once it has approached the cup it has enough information (using its overhead camera) to plan how to get to the bin. Thus, the path plan will usually be available by the time the robot has grasped the cup.

While the principle of hierarchy is usually applied to planning, hierarchical coarse-to-fine strategies have also proven useful for monitoring and exception handling. For example, the system can use a coarse sensor to provide an indication of potential trouble or opportunity, and then actively explore using its fine sensors to discriminate the possibilities. The HERO uses this strategy in collecting cups — the overhead camera detects regions in the image that are approximately cup-shaped. These trigger monitors that insert tasks to approach the object and map it with the wrist-mounted sonar to determine if it in fact matches the robot's model of a cup.

The HERO uses a similar coarse-to-fine strategy to handle certain exceptions, enabling it to react quickly without using excessive resources. The robot uses on-board *guarded move* procedures that continually monitor the sonars and wheel encoders while the robot is moving. If the sonars detect a close object, or the encoders detect no movement, the motors are stopped to stabilize the robot and an exception is signaled (a similar mechanism is used on the Rover). TCA then uses its exception-handling mechanism to provide a reasoned response to the problem. A more sophisticated approach along these lines is that of (Miller, 89) which generates a sensor profile for the low-level control to monitor.

Another aspect of the hierarchical exception-handling mechanism is that if an exception handler decides it cannot deal with an error, TCA searches up the task tree for another handler. This approach is used in docking the HERO on its battery charger. To ensure good contact, the HERO is centered in front of the charger and backs up past where the charger is predicted to be. When the charger is contacted, the guarded move detects that the wheels are not moving, and halts the robot. This exception is passed up an exception handler associated with the "dock" goal, which checks whether the robot is in fact on the charger (by querying the HERO's charge light). If it is not on the charger, which might occur if the robot was not centered correctly, the exception is passed up to a more general handler which replans the docking action.

## Concurrency

Another way to deal with limited resources is to make full use of the available resources. In particular, we use the distributed nature of TCA to exploit the opportunities for concurrency in the domain. One source of concurrency is the interleaving of planning and execution described above. Another valuable source is the perception system. Our original system (Lin et al. 1989a) processed vision data on request; the new system (Lin et al. 1989b) continually processes image data to create 2D world maps used by the path planner and monitors. This use of asynchronous vision processing has led to a nearly twofold speedup in the cup-collection task.

To be reactive to changing conditions, the system must monitor its environment concurrently with planning and execution. To this end, processes can specify the frequency of polling for monitors. For optimal performance, these frequencies should be based on the likelihood of the monitored condition occurring, the urgency for response, and the time needed to react. For example, if at 70% of maximum charge the robot has 15 minutes of battery charge left, and the robot needs a maximum of 10 minutes to reach its charger, then the battery can be safely monitored (at the 70% level) every 5 minutes.

Some conditions require reaction times too fast for a centralized system to guarantee response, such as avoiding obstacles while moving. As described previously, our approach in such cases is to implement reflexive procedures outside TCA that stabilize the mechanism and then notifies the higher-level system (through TCA) to handle the exception.

## Focus of Attention

A third organizing principle is that a robot with limited resources must explicitly maintain a focus of attention — that is, it is unreasonable to expect the system to monitor all possible conditions or to plan for all tasks at once. While currently the robot's focus of attention is programmed in, we are developing techniques that would enable the system to make such decisions itself.

A recurring theme throughout this paper is that of selectively monitoring for change. While, admittedly, this could cause the system to miss something unexpected, the tradeoff is necessary under the presumption of limited resources. TCA supports selective monitoring in several ways. First, the use of asynchronous, demon-invoked monitors enables the system to check conditions only when certain triggering events occur. Second, the frequency of polling monitors can be specified so as to tune the monitor to the likelihood of the changes occurring.

Third, monitors in TCA are constrained to begin and end at certain times relative to other tasks. For example, the HERO system sets up a monitor that checks whether the robot is holding an object which is constrained to start after the cup has been grasped and continues until the cup is placed over the bin. In this way, resources are not overloaded checking for conditions that are not applicable to the current tasks. This contrasts with other systems, (e.g., (Brook 1986), (Kaelbling 1986)), in which all conditions are continually monitored.

TCA uses its resource mechanism to maintain the focus of attention on its currently active tasks — if multiple messages arrive for the same resource, they are prioritized. While TCA currently uses only a simple FIFO queue to schedule access, we are beginning to investigate dynamic scheduling based on a cost/benefit analysis. For example, if the battery monitor warns of a low charge while the robot is collecting a cup, the system will prioritize the tasks based on the costs of completing the collection task followed by recharging, versus recharging first and then getting the cup.

Note that resource-based focus of attention allows the system to easily handle multiple, non-conflicting tasks. For example, by declaring the robot's motors and voice synthesizer to be separate resources, the system can concurrently navigate and communicate with lab occupants. Similarly, the robot can plan one task while executing another since the planner resource is separate from the robot controller resources.

## Conclusions

While many mobile robots use a plethora of sensors and processors, we are investigating how to build robust and reactive robot systems that have limited resources for the tasks and environments they encounter.

We have developed the Task Control Architecture to manage the robot's resources. TCA provides mechanisms for defining hardware and software resources, for creating hierarchical task trees (our representation of plans), for monitoring the environment, and for handling exceptions in a context-dependent way.

Our experiments with such systems have led us to postulate several general (albeit, not particularly surprising) principles of system organization. First, the system should take advantage of hierarchy in the domain by using a variety of coarse-to-fine strategies. These include 1) hierarchical planning at various levels of detail, 2) using coarse sensors to detect potential opportunities or problems, followed by active search using fine sensors, and 3) reflexive procedures that stabilize the robot followed by a reasoned response to correct the error.

Second, the system should take advantage of those resources it does have by exploiting the concurrency inherent in its tasks. This includes 1) concurrent perception, 2) planning during execution, and 3) monitoring selected conditions at different rates. Finally, the system needs to keep an explicit focus of attention.

These principles have been tested using TCA to control the HERO robot in its tasks of collecting cups, retrieving printer output, and recharging, and to "walk" a single leg of the CMU Planetary Rover through rough terrain. We intend to further test and refine these ideas by operating the HERO in an occupied laboratory for extended periods of time.

## Acknowledgements

Many people have contributed to the HERO and Planetary Rover projects. In particular, Christopher Fedor implemented much of the TCA, Long-Ji Lin designed the exception-handling mechanism and implemented much of the HERO system, and Lonnie Chrisman and Goang-Tay Hsu implemented behaviors to retrieve printer output and deliver objects to workstations. The research is funded by under NASA contract NAGW-1175.

## References

Bares, J, et al., Ambler: An autonomous rover for planetary exploration, in: *IEEE Computer* vol 22, no 6 (1989).

Brooks, R, A robust layered control system for a mobile robot, in: *IEEE Journal of Robots and Automation*, vol RA-2, no 1 (1986).

Kaelbling, L, An architecture for intelligent reactive systems, in: *Proceedings of the Workshop on Planning and Reasoning about Action* (1986).

Krotkov, E, Roston, G, Simmons, R, Integrated system for single leg walking, in preparation.

Lin, L J, et al., A case study in autonomous robot behavior, CMU-RI-89-1, Robotics Institute, Carnegie Mellon University (1989).

Lin, L J, Simmons, R, Fedor, C, Experience with a task control architecture for mobile robots, CMU-RI-89-29, Robotics Institute, Carnegie Mellon University (1989).

Miller, D, Execution monitoring for a mobile robot system, in: *SPIE Conference on Intelligent Control* (1989).

Simmons, R, Mitchell, T, A task control architecture for mobile robots, in: *Stanford Spring Symposium* (1989).

# Subjective Ontologies

Devika Subramanian[*]
Computer Science Department
Cornell University
Ithaca, New York 14850

John Woodfill[†]
Computer Science Department
Stanford University
Stanford, California 94305

## Abstract

In this paper, we study the tradeoffs made in expressive power and computational efficiency by indexical-functionals [?], a new class of representations now popular in the world of reactive planning [?]. We present a knowledge level analysis [?] of some indexical terms that clarifies their logical content and helps us identify the source of their computational power. In particular, we demonstrate that indexical terms can be formalized in a restriction of the situation calculus. Indexical theories gain descriptional as well as computational efficiency by using terms whose referents are determined in context by the perceptual machinery of an agent, and by selecting those and only those terms that are essential for determining a course of action for the agent. We show how, in some cases, indexical theories can be synthesized from a situation calculus description of a planning problem using knowledge of the goals of the agent. The general problem of synthesizing appropriate indexicals remains open. Our analysis of indexical terms provides us not only with a way of understanding what their ontological underpinnings are, but also helps us analyze the conditions under which they are useful.

## Introduction

The intractability of classical planning and the need to actively monitor plans in a complex, dynamic world has led to the development of reactive planners that build plans at execution time based solely on the situation existing then [?]. Several designs for reactive planners have been proposed in the recent literature: the theory of situated automata and their combinational circuit compilation by Rosenschein and Kaelbling [?], the theory of indexical-functional aspects in the reactive planner Pengi proposed by Chapman and Agre, universal plans by Schoppers [?], and Rod Brooks' [?] subsumption architecture. The aim of our analysis of these approaches is to articulate the assumptions about the world and the planning process that are made in order to contain the complexity of planning in these frameworks. The results we seek are a declarative specification of the worlds in which reactive planning works and an analysis of its computational and descriptional efficacy.

The impetus to perform this analysis comes from two sources. One, the existence of the fundamental tradeoff between expressiveness and efficiency in knowledge representation [?] indicates that the computational advantage in indexical-functional representations ought to come from expressiveness limitations. If we can identify them, we can determine the worlds for which they are an epistemologically adequate formalism. Two, we would like to design methods for compiling out indexical-functional aspects from more expressive situation calculus descriptions of the world. Presently the designers of Pengi synthesize these aspects by hand. Should the rules of the world change, they will have to manually reconstruct these indexical-functionals and the situation-action rules that use them. Specifying modifications at the level of individual situation-action rules (or worse circuits) is clumsy and for moderately complex worlds almost impossible. An analysis of this representation in terms of situation calculus allows us to build a compiler for indexical-functional aspects and thus to specify modifications at a reasonable level of description.

The designers of Pengi state that "registering and acting on indexical-function aspects is an alternative to representing and reasoning about complex domains, and avoids combinatorial explosions". We demonstrate that indexical-functionals can be constructed out of the objective ontology of the world assumed by a situation calculus representation using the processes of indexicalization and propositionalization. Theories that contain indexical-functionals use a simpler model of time than the situation calculus, they also solve a simpler formulation of the standard planning problem.

The problems with the classical planning model are well known. First, it assumes that the world had been abstracted into an objective description consisting of a

[*]This author is supported by NSF IRI-8902721.
[†]This author was supported in part by ONR Contract N00014-81-K-0004.

set of well-formed formulas in the situation calculus. The effects of actions in the world are then formulated as sentences in first-order logic and the planning problem is: what sequence of actions achieves the goal description? Chapman[?] showed that the problem as formulated above is NP complete even in the propositional case. This standpoint on planning leaves open the questions of how exactly the referents of symbols in these theories map to the world (the reference problem), and introduces the additional problem of execution monitoring where the planner makes sure that the world is indeed in the state specified by the descriptions.

Reactive planners make a more careful analysis of the requirements of planning. Situation calculus allows one to make too many distinctions in the world. Planners that need to work in changing environments cannot be as wasteful – they need to simplify their models of the world and of their decision-making. We demonstrate that reactive planners access the world via terms in a agent-centered ontology (e.g. the block-to-my-right) as opposed to terms in an objective ontology (block A). This allows one to rewrite theories of the world in propositional form. Designers of reactive planners develop elaborate perceptual components for delivering the referents of these subjective or indexical terms. This is a partial solution to the reference problem. The next simplification they perform is to formulate the planning problem as: what action to take *next*. This simplification is in accord with the nature of the world – it is pointless to calculate action two or more time steps from *now*, because the world may cease to be in a state that allows for the execution of these action sequences.

This informal argument is made precise in this paper and extended in [?] to account for indexicals other than the time indexical *Now*. Using the indexical situation calculus we demonstrate that

1. What is common among the various approaches to reactive planning is their use of propositional theories to link perception to action. Naive propositionalization of full first-order theories of situations will not do – instead, indexicals are used to provide limited quantification.
2. The visual sensors of an agent deliver the referent of an indexical at any given time. The cost of unification to determine an applicable instance of a universal rule in a theory is replaced by the cost of determining the referents via perception. When the rules of the world change slowly relative to the actual state of the world at any given time, the approach of reasoning with indexical theories is a computational win.
3. The theories of action that can be indexicalized are those which permit computation of the *next* action based on information available *now*. It is possible to extend this analysis to agents with limited state information.
4. Learning indexicals is a hard induction problem, and this is clearly revealed in our attempts to indexicalize theories with objective ontologies. The hardest part of designing reactive agents is to pick appropriate distinctions (indexicals) in the world that determine the agent's next action.

Reactive planners work because their designers carefully and explicitly separate the changing aspects of the world from the stable ones, at design time. The rules of the world are compiled into propositional theories with indexicals which allow them to access the dynamic aspects of the world via sensors.

We begin by introducing the time indexical *Now* into situation calculus. An agent that has its present theory of the world expressed in terms of the logical constant `Now` alone, is computationally simpler than one that requires expression of statements about time points in terms of some fixed time point other than *Now*. We introduce the axioms needed for meaningful introduction of *Now* in the situation calculus. In the next section, we use these axioms to reformulate a standard first order predicate logic sentence that assumes an objective ontology into an indexical one tuned for one-step planning. The introduction of *Now*, sets the stage for introducing further indexicals pertaining to space (*Here* and *There*), and to objects (*The Block-that-is-behind-me*). However, introducing additional indexical terms, necessitates making assumptions about the process of determining their referents. We then discuss the expressive limitations of indexicals and present an analysis of the time and space costs associated with using them for execution monitoring and one-step planning. We conclude with a summary of the main points and directions for future research.

## A Conceptualization with a View

To use logic one must choose a point of view. Traditional situation calculus starts from a sort of god's eye or objective point of view. This objective stance allows one to frame eternal sentences, e.g. "Block A is on block B at 3:13, May 3rd, 1989". "block A" and "3:13, May 3rd, 1989" are considered as rigid designators that denote the same things for all time. Eternal sentences have the nice property that any conjunction of eternal sentences is also an eternal sentence. If an agent acquires true beliefs in the form of eternal sentences, it can always conjoin these new beliefs to its old true beliefs without fear of contradiction.

It is also possible to develop a situation calculus from an agent's subjective point of view. Consider a subjective ontology. What things and relations are central to an agent's point of view? First, the current time seems interesting, *Now* must be a distinguished thing for an agent. The place, *Here*, must be something special. The agent could have a theory of here-and-now couched in symbols denoting these various elements of the ontology of here-and-now.

As a first example of an indexical theory, let us consider a rather narrow minded agent that makes one distinction, and hence deals with one predicate, *Rich*. *Rich* is true when the agent is rich, and false when the agent is not. Suppose further that the agent owns AI stock that was worth millions yesterday. Yesterday, the agent could have a theory containing the one sentence `Rich`. Today, the stock market fell and the stock is worth nothing; the agent can add $\neg$`Rich` to its supply of facts. The agent seems committed to the sentence: `Rich` $\wedge \neg$`Rich`.

*Rich* must be made situation dependent. From an agent's point of view, the current moment (not $S_0$, the first situation) is the most interesting moment, so we introduce a term `Now` which now denotes the current situation. Our agent can maintain $\neg$`Rich(Now)`. The agent's state of richness yesterday can be stated today if we introduce a function *Before* which maps one state to its predecessor: `Rich(Before(Now))` Notice that `Rich-Now` is a quite useful propositional symbol where `Rich-S0` is not. The question of whether I am rich now is always fairly relevant, while the question of being rich at some fixed time loses significance. In what follows, we will describe how we can convert a theory that contains a situation-dependent predicate like *Rich* expressed in the full situation calculus that allows access to arbitrary situations, into an indexical one that allows reference only to the current situation and its predecessors via the *Before* function.

Two obstacles prevent the use of *Now* and *Before* and their corresponding symbols `Now` and `Before` in standard situation calculus: first, *Before* is not definable in standard situation calculus as there is no unique predecessor of a situation; second, it seems that an agent's theory becomes highly non-monotonic across time. `Rich(Now)` was in the theory yesterday, and it is not today. The first obstacle is overcome by adding appropriate axioms for situations and *Before*. The second obstacle is stepped around by accepting the idea of having a different, current, theory for each new situation in time, and considering the issue of transforming one theory of here-and-now at $t_1$ to another theory of here-and-now at $t_2$.

## The Reformulated Situation Calculus

The need to provide more axioms for the situation calculus arises because *Before* is not explicitly definable in the standard situation calculus. To see that *Before* is not definable consider a world containing a compass $C$ and four directions in which the compass' needle can point: $N$, $E$, $S$ and $W$. The two actions in this world, *Clockwise*, and *Anti-Clockwise*, turn the compass. One model of this world has four situations, one for each direction the needle of the compass can point to.

$$S1 = C(N), S2 = C(E), S3 = C(S), S4 = C(W)$$

If *Before* were definable, $Before(S2)$ would have to be either $S1$ or $S3$. But $Do(Clockwise, S2) = S1$ and $Do(Anti\text{-}Clockwise, S3) = S1$. Clearly there can be no function defined on these four situations which picks out the previous situation.

The following axioms restrict the models of situation calculus to those in which there is a unique previous situation. The variables `a1` and `a2` range over actions, and `s1` and `s2` range over situations.

Intuitively, we would like to define a function *Before* which maps a situation into its predecessor, just as the function "the day before" maps today into yesterday. This intuition is captured by the following definition for a relation:

$(A)$. $\{\langle x, y\rangle \mid \exists a\, Do(a, x) = y\}$

However for many models of situation calculus, this formula is not functional. The definition becomes functional if *Do* satisfies the following sentence.

$(B)$. $\forall y\, \exists a\, x\, Do(a, x) = y$
$\Rightarrow (\forall z\, Do(a, z) = y \Rightarrow x = z)$

After restricting *Do* in this way, *Before* is still not defined by $(A)$ for situations which are not in the range of *Do*. We choose to introduce an axiom relating *Before* and *Do* which entails $(B)$ and forces *Before* to agree with $(A)$:

$(ISC1)$. $\forall a\, s\, Before(Do(a, s)) = s$

As in our everyday world, given this axiom, each situation has a unique predecessor. However, there may be several distinct actions which take one situation to the same successor. Although not necessary for the construction, in accord with common intuition, we provide an axiom which forces situations with distinct histories to be distinct:

$(ISC2)$. $\forall a1\ a2\ s\ a1 \neq a2$
$\Rightarrow Do(a1, s) \neq Do(a2, s)$ *optional*

Later when we discuss transforming one theory of here-and-now to the next theory of here-and-now, we will require that every situation be *before* some situation, (that *Before* be onto) for a soundness condition to hold:

$(ISC3)$. $\forall s1\, \exists s2\, Before(s2) = s1$

We introduce $(ISC4)$ in order to pick out an initial situation for a set of standard points, just as 0 picks out an initial element in the standard points of Peano arithmetic.

$(ISC4)$. $\neg\exists a\, s\, Do(a, s) = S0$ *optional*

Given the notion that a situation has a predecessor, the identity of the action resulting in a situation becomes an object of interest[1]. $(ISC5)$ is a definition of an additional relation *Action-that-generated* in terms of *Do*.

$(ISC5)$. $\forall a\, s1\, Action\text{-}that\text{-}generated(s1, a)$
$\Longleftrightarrow \exists s2\, Do(a, s2) = s1$ *definition*

`Action-that-generated(S,A)` is true if performing $A$ in some situation would result in $S$.

A useful consequence of these axioms and definitions is:

[1]If $(ISC2)$ is omitted, the *identities* of the actions which could have resulted in a situation are objects of interest.

$(ISC6)$. $\forall$a s $\texttt{Action-that-generated(s,a)} \iff$
$\texttt{Do(a,Before(s))} = \texttt{s}$ $[ISC1, ISC5]$

## Transforming Theories

At each situation an agent is best served by having the best theory for that situation. In some environments the best theory for a situation is a theory couched in the subjective terms denoting the objects present to the agent in that situation. Each new situation will have a different theory. The theory of one situation is not, in general, a subset of the theory of the succeeding situation. We need a method for generating the current theory in terms of the current subjective ontology that describes the preceding situation. For now, only the term Now changes its reference from situation to situation We define a translation which, given a theory of the previous situation, produces a theory that is true in the current situation. The translation maps the term Now in the first theory to a term which designates the same thing in the second theory. Since Before(Now) now designates the situation which Now designated in the previous situation, the translation $\pi$ maps Now to Before(Now), and all other parameters to themselves. To generate the theory of a new situation from the theory of the previous situation one applies the syntactic translation corresponding to $\pi$, replacing all occurrences of Now by Before(Now). This transformation has the desirable property that it preserves the consequences of a theory modulo the shift in designation for Now.

**Lemma 1** *Given a set of sentences $\mathcal{T}$, and letting $\psi$ denote the sentence* $\forall$s1 $\exists$s2 s1 = Before(s2) *(axiom ($ISC2$) from above), and $\pi$ denote* [Now $\rightarrow$ Before(Now)], *(the replacement of* Before(Now) *for* Now*), for any sentence $\phi$: $\mathcal{T} \cup \{\psi\} \models \phi$ if and only if $\pi(\mathcal{T}) \cup \{\psi\} \models \pi(\phi)$*

## Reformulations

Now that we have the apparatus for talking about *Now* and *Before*, we consider formulations which are particularly well suited for one-step planning. We start with a blocks world axiom from [?]:

$$\forall \texttt{s x y} \quad \texttt{T(On(x,y),s)} \wedge \texttt{T(Clear(x),s)} \Rightarrow \texttt{T(Clear(y),Do(U(x,y),s))} \qquad (1)$$

For one-step planning, we also need some notion of the relation between goals and what an agent ought to do now. We introduce an axiom of rationality.

$(ISC7)$. $\forall$a p s $\texttt{T(p,Do(a,Now))} \wedge \texttt{Goal(p,Now)}$
$\wedge \neg \texttt{T(p,Now)} \Rightarrow \texttt{Must(Now)} = \texttt{a}$

Which just says that if doing $a$ would result in $p$, if one has the goal $p$, and if $p$ doesn't hold now then one must do $a$.

In one step planning the *Goal* and properties of *Now* are the relevant aspects of the situation, and so the axiom should be reformulated in those terms.

Resolving (ISC7) and (1), so that we are talking about goals,

$$\forall \texttt{x y} \quad \texttt{T(On(x,y),Now)} \wedge \texttt{T(Clear(x),Now)} \wedge \neg\texttt{T(Clear(y),Now)} \wedge \texttt{Goal(Clear(y),Now)} \Rightarrow \texttt{Must(Now)} = \texttt{U(x,y)} \qquad (2)$$

Introducing the time indexical Now has propositionalized the situation component of the planning axiom. We can carry this one step further, by instantiating the variables x and y with particular blocks.

$$\texttt{T(On(A,B),Now)} \wedge \texttt{T(Clear(A),Now)} \wedge \neg\texttt{T(Clear(B),Now)} \wedge \texttt{Goal(Clear(B),Now)} \Rightarrow \texttt{Must(Now)} = \texttt{U(A,B)} \qquad (3)$$

And finally introducing propositional symbols to make the sentence propositional,

$$\texttt{On-A-B} \wedge \texttt{Clear-A} \wedge \neg\texttt{Clear-B} \wedge \texttt{Goal-Clear-B} \Rightarrow \texttt{Must-U-A-B} \qquad (4)$$

Formula (4) is good for one-step planning because determining the truth value of a set of literals about *Now* determines what the agent must do.

Here is another example taken from the Pengo [?].
$\forall$apts T(Completely-Adjacent(a,p),s) $\wedge$
T(Heading(a) = Axis-Between(a,p),s)
$\wedge$T(Heading(a) = Axis-Between(p,t),s)
$\Rightarrow$ Must(s) = Throw-Projectile-At-Target(a,p,t)
This states that if the penguin, a projectile and a target are lined up in that order, the penguin must throw the projectile at the target.
$\forall$aptT(Completely-Adjacent(a,p),Now) $\wedge$
T(Heading(a) = Axis-Between(a,p),Now)
$\wedge$T(Heading(a) = Axis-Between(p,t),Now)
$\Rightarrow$ Must(Now) = Throw-Projectile-At-Target(a,p,t)
We replace the situation variable *s* by the constant *Now*, so that the above statement is specialized to the current situation.
T(Completely-Adjacent(A,P),Now) $\wedge$
T(Heading(A) = Axis-Between(A,P),Now)
$\wedge$T(Heading(A) = Axis-Between(P,T),Now)
$\Rightarrow$ Must(Now) = Throw-Projectile-At-Target(A,P,T)
Again, we replace universal variables $a$ and $p$ that stood for arbitrary penguins and projectiles by constants A and P that are indexicals: A stands for Pengo and P stands for the projectile adjacent to Pengo now. The referent of P is delivered by Pengo's visual apparatus.
T-Completely-Adjacent-A-P-Now $\wedge$
T-Heading-A-=-Axis-Between-A-P-Now
$\wedge$T-Heading-A-=-Axis-Between-P-T-Now
$\Rightarrow$ Must-Now-=-Throw-Projectile-At-Target-A-P-T
In both these theories, the right indexicals could be generated directly by replacing situation variables and object variables by the constants Now and A and P. As long as the visual sensors deliver the right referents for A and

P, appropriate actions will be generated by our propositional theory. In general, the choice of indexicals is to be determined by what sensors an agent has and its present goals. The most creative aspect of reactive planner design is the choice of which indexical-functional aspects of the situation determine an agent's action now.

## Analysis

The class of theories in the situation calculus that we can transform to the indexical form is restricted by the requirement that the theory be consistent with $ISC1$-$ISC4$. This restriction rules out theories that fix the number of situations or that make two situations with distinct histories equivalent.

**Claim 1** *A theory $\mathcal{T}$ can be indexicalized if $\mathcal{T} \cup \{ISC1\text{-}ISC4\}$ is consistent.*

In a previous section we used the axioms of the indexical situation calculus to generate propositional theories that are suited for plan monitoring from a theory in the situation calculus. The previous theorem establishes the class of theories for which this conversion is possible. We now show that the compilation methods we use are sound.

Let $I$ be a transformation performed on a theory $\mathcal{T}$ in the situation calculus that obeys the limitation in Claim 1. $I$ is sound if

*Soundness.* $\mathcal{T} \cup \texttt{ISC1-ISC6} \models \texttt{I}(\mathcal{T})$

.

We further require that the theory $I(\mathcal{T})$ be propositional, because we wish to gain the computational advantages that reasoning within the propositional fragment of logic gives us.

**Claim 2** *The transformations in Sections 3 are sound.*

**Proof:** Follows from the fact that the transformation $I$ is deduction. □

We examine the computational consequences of indexicalization and propositionalization. First we consider space requirements. To propositionalize a theory $\mathcal{T}$ in clausal form in a typed logic, one must augment the language until there are object constants for each object in each type. The resulting theory $\mathcal{T}'$ is the set of all ground instances of all clauses in the original theory. The cardinality of this set of clauses is expressed by the following formula.

$$|\mathcal{T}'| = \sum_{c \in \mathcal{T}} \prod_{v \in Variables(c)} |type(v)|$$

Considering one type, $\mathcal{D}$, a clause replicates exponentially in the number of variables occurring in it.

$$|\mathcal{T}'| = \sum_{c \in \mathcal{T}} |\mathcal{D}|^{|Variables(c)|}$$

Even a theory with one clause containing one literal with one situation variable in propositional form would contain infinitely many clauses.

Here is where the power of the indexical approach lies. If the agent can do all its reasoning with a few situation terms such as `Now` and `Before(Now)`, and a few world terms `This-Block` and `That-Block`, this explosion may not be significant. Essentially we reduce the size of the theory to

$$|\mathcal{T}'| = \sum_{c \in \mathcal{T}} \prod_{v \in Variables(c)} |type(v)|$$

where type($v$) = 1 (`Now`) when $v$ is a situation variable and type($v$) = $n$ (the number of propositional object referents, like `the-bee-that-is-chasing-me`) when $v$ is an object variable.

**Claim 3** *Indexicalization is a win in terms of space only if the number of situation referents and object referents (indexical-functional aspects) is much smaller than the total number of situations and objects described in the theory $\mathcal{T}$ in the situation calculus.*

Suppose we have a theory of cars in the United States expressed in situation calculus. The straightforward propositionalization of this theory would result in another theory whose size is proportional to 200 million (number of cars in the United States). This theory allows us to name every car. If however, we restricted our attention to cars of interest to the agent: *The-car-behind-me*, *The-car-passing-me* etc.., we would have a propositional theory whose size is proportional to the number of such referents. The power of using these indexical terms is that it gives us implicit quantification: the actual referent of the `The-car-behind-me` is determined by the perceptual machinery of the agent. These indexical terms allow access to that part of the immense propositional theory that is actually needed by the agent.

A really significant performance improvement can be achieved by transforming the propositional theory into a set of equations. This can be done if the propositional theory can be "directionalized". We first subdivide the literals in the theory into the perceptions or the givens (examples are `On-A-B-Before-Now` and `On-A-B-Now`) and the actions or the conclusions (e.g. `Clear-B-Now` and `Must-Unstack-A-B`). To facilitate the directionalization of the reasoning chains in this theory from perceptions to actions, we will assume that the conclusion literals occur in unnegated form.[2] Now, for each conclusion, we collect all minimal conjunctions of the givens which imply it, disjoin the set of conjunctions, and construct a definition of the form *Conclusion* $\equiv$ *Disjunction of Conjunctions.*

**Claim 4** *The worst case complexity of concluding the truth of the action literals in the directionalized theory constructed above is linear in their number.*

[2]This is to avoid problems that arise from negation as failure.

**Proof:** Since the truth assignments to the givens can be determined in constant time by the perceptual equipment, the complexity of concluding each action literal in the constructed theory is also a constant. If there are $n$ action literals, we need $O(n)$ time to determine their truths.

The definitions for the action literals can be straightforwardly compiled into combinational circuits (AND gates for conjunctions and OR gates for disjunctions), then the time complexity for concluding *all* the action literals goes to $O(1)$.

**Claim 5** *By compiling the definitions of the action literals into combinational circuits, we can determine their truths in constant time.*

Note that the two results above depend on the fact that perception can be done in constant time. This is actually an assumption that Pengi makes.

Another interesting property of indexicalized theories is that they allow simple descriptions of current situations. The description of the current situation is `Now` in the propositional indexical theory and `Do(an (Do(.... (Do(a1, s0)))))` in the original theory expressed in situation calculus (`s0` is the initial situation). Situations in the past can be accessed by the `Before` function in the indexical theory, so that situations far back in the past are rather clumsy to express; their length is linear in their distance from `Now`. In the first-order theory expressed in situation calculus, the length of a term describing a situation is linear in its distance from the initial situation `s0`.

## Conclusions

This paper provides a principled account of several forms of indexicality. The introduction of *Now* is a necessary condition for introducing other indexical terms non-rigidly referring to objects. We present a method for generating indexical theories from a domain description in situation calculus. Determining the truth of literals involving indexicals requires that perception be able to determine their referents. Indexical theories can be made into propositional ones with good computational properties. This benefit comes at the cost of losing the ability to rigidly refer to arbitrary objects and situations.

We have shown that indexical-functionals can be explained in terms of a well understood formalism: the situation calculus. In so doing we have exposed some of the expressiveness and efficiency trade-offs resulting from using them.

## Acknowledgements

We would like to thank Andy Baker, Michael Genesereth, Jane Hsu, Nils Nilsson, and Stan Rosenschein for helpful discussions on an earlier version of this paper which appeared in the proceedings of the First International Conference on Representation an Reasoning, May 1989.

## References

[Agre and Chapman, 1987] Philip E. Agre and David Chapman. Pengi: An implementation of a theory of activity. In *AAAI*, 1987.

[Brooks, 1987] Rod Brooks. Intelligence without representation. In *Workshop on the Foundations of AI*, 1987.

[Chapman, 1987] David Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32(3), 1987.

[Firby, 1987] R. James Firby. An investigation into reactive planning in complex domains. In *AAAI*, 1987.

[Genesereth and Nilsson, 1987] Michael R. Genesereth and Nils J. Nilsson. *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann, 1987.

[Levesque and Brachman, 1985] Hector J. Levesque and Ronald J. Brachman. A fundamental tradeoff in knowledge representation and reasoning (revised version). In *Readings In Knowledge Representation.* Morgan Kaufmann, 1985.

[Newell, 1982] Alan Newell. The knowledge level. *AIJ*, 18(1), 1982.

[Rosenschein and Kaelbling, 1986] S. J. Rosenschein and L. P. Kaelbling. The synthesis of machines with provably epistemic properties. In *Proc of the Conf on Theoretical Aspects of Reasoning about Knowledge*, 1986.

[Schoppers, 1989] M. Schoppers. *Representation and Automatic Synthesis of Reaction Plans.* PhD thesis, University of Illinois at Urbana-Champaign, 1989.

[Subramanian and Woodfill, 1990] Devika Subramanian and John Woodfill. Making situation calculus indexical. Technical report, Cornell University, 1990. In Preparation.

# First Results with DYNA, an Integrated Architecture for Learning, Planning and Reacting

Richard S. Sutton

GTE Laboratories Incorporated
Waltham, MA 02254
Rich@gte.com

How should a robot decide what to do? The traditional answer in AI has been that it should deduce its best action in light of its current goals and world model, i.e., that it should *plan*. However, it is now widely recognized that planning's usefulness is limited by its computational complexity and by its dependence on a complete and accurate world model. An alternative approach is to do the planning in advance and compile its result into a set of rapid *reactions*, or situation-action rules, which are then used for real-time decision making. Yet a third approach is to *learn* a good set of reactions by trial and error; this has the advantage of eliminating the dependence on a world model. In this paper I briefly introduce *Dyna*, a simple architecture integrating and permitting tradeoffs among these three approaches. Results are presented for a simple Dyna system that learns from trial and error while it learns a world model and uses the model to plan reactions that result in optimal action sequences.

Dyna is based on the old idea that planning is like trial-and-error learning from hypothetical experience (Craik, 1943, Dennett, 1978). The theory of Dyna is based on the classical optimization technique of *dynamic programming* (Bellman, 1957; Ross, 1983) and on the relationship of dynamic programming to reinforcement learning (Watkins, 1989; Barto, Sutton & Watkins, 1989), to temporal-difference learning (Sutton, 1988), and to AI methods for planning and search. Werbos (1987) has previously argued for the general idea of building AI systems that approximate dynamic programming, and Whitehead (1989) and others (Sutton & Barto, 1981; Sutton & Pinette, 1985; see also Rumelhart et al., 1986) have presented results for the specific idea of augmenting a reinforcement learning system with a world model used for planning.

The Dyna architecture consists of four primary components, interacting as shown in Figure 1. The *policy* is simply the function formed by the current set of reactions; it receives as input a description of the current state of the world and produces as output an action to be sent to the world. The *world* represents the task to be solved; prototypically it is the robot's external environment. The world receives actions from the policy and produces a next state output and a reward output. The overall task is defined as maximizing the long-term average reward per time step (cf. Russell, 1989). The Dyna architecture also includes an explicit *world model*. The world model is intended to mimic the one-step input-output behavior of the real world. Finally, the Dyna architecture includes an *evaluation function* that rapidly maps states to values, much as the policy rapidly maps states to actions. The evaluation function, the policy, and the world model are each updated by separate learning processes.

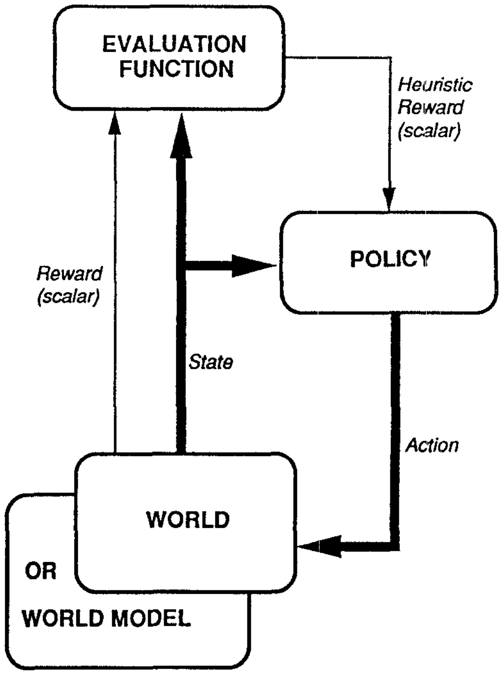


**Figure 1. Overview of Dyna.**

For a fixed policy, Dyna is a simple reactive system. However, the policy is continually adjusted by an integrated planning/learning process. The policy is, in a sense, a *plan*, but one that is completely conditioned by current input. The planning process is incremental and can be interrupted and resumed at any time. It consists of a series of shallow searches, each typically of one ply, and yet ultimately produces the same result as an arbitrarily deep conventional search. I call this *relaxation planning*. Dynamic programming is a special case of this.

Relaxation planning is based on continuously adjusting the evaluation function in such a way that credit is propagated to the appropriate steps within action sequences. Generally speaking, the evaluation of a state $x$ should be equal to the best of the states $y$ that can be reached from it in one action, taking into consideration the reward (or cost) $r$ for that one transition, i.e.:

$$Eval(x) \ \text{``}=\text{''} \ \max_{a \in Actions} E\{r + Eval(y) \mid x, a\}, \quad (1)$$

where $E\{\cdot \mid \cdot\}$ denotes a conditional expected value and the equal sign is quoted to indicate that this is a condition that we would like to hold, not one that necessarily does hold. If we have a complete model of the world, then the right-hand side can be computed by looking ahead one action. Thus, we can generate any number of training examples for the process that learns the evaluation function: for any $x$, the right-hand side of (1) is the desired output. If the learning process converges such that (1) holds in all states, then the optimal policy is given by choosing the action in each state $x$ that achieves the maximum on the right-hand side. There is an extensive theoretical basis from dynamic programming for algorithms of this type for the special case in which the evaluation function is tabular, with enumerable states and actions. For example, this theory guarantees convergence to a unique evaluation function satisfying (1) and that the corresponding policy is optimal (e.g., see Ross, 1983).

The evaluation function and policy need not be tables, but can be more compact function approximators such as decision trees, *k-d* trees, connectionist networks, or symbolic rules. Although the existing theory does not directly apply to the case in which these machine learning algorithms are used, it does provide a theoretical foundation for exploring their use. Finally, this kind of planning also extends conventional planning in that it is applicable to stochastic and uncertain worlds and to non-boolean goals.

The above discussion gives the general idea of relaxation planning, but not the exact form used in Dyna.

```
0. Decide if this is a real experience or a hypothetical
   one.
1. Pick a state $x$. If this is a real experience, use the
   current state.
2. Form prior evaluation of $x$: $e \leftarrow Eval(x)$
3. Choose an action: $a \leftarrow Policy(x)$
4. Do action $a$; obtain next state $y$ and reward $r$ from
   world or world model.
5. If this is a real experience, update world model from
   $x$, $a$, $y$ and $r$.
6. Form posterior evaluation of $x$: $e' \leftarrow r + \gamma Eval(y)$
7. Update evaluation function so that $Eval(x)$ is
   more like $e'$ rather than $e$; this typically involves
   temporal-difference learning.
8. Update policy—strengthen or weaken the tendency
   to perform action $a$ in state $x$ according to $e' - e$.
9. Go to Step 0.
```

**Figure 2. Inner loop of a Dyna algorithm.** These steps are repeatedly continually, sometimes with real experiences, sometimes with hypothetical ones.

Dyna is based on a closely related method known as *policy iteration* (Howard, 1960), in which the evaluation function and policy are simultaneously approximated. In addition, Dyna is a *Monte Carlo* or *stochastic approximation* variant of policy iteration, in which the world model need only be sampled, not examined directly. Since the real world can also be sampled, by actually taking actions and observing the result, the world can be used in place of the world model in this method. In this case, the result is not relaxation planning, but a trial-and-error learning process much like reinforcement learning (see Barto, Sutton & Watkins, 1989). In Dyna, both of these are done at once. The same algorithm is applied both to real experience (resulting in learning) and to hypothetical experience generated by the world model (resulting in relaxation planning). The results in both cases are accumulated in the policy and the evaluation function. There is insufficient room here to fully justify the algorithm, but it is quite simple and is given in outline form in Figure 2.

As a simple illustration of the Dyna architecture, consider the navigation task shown in the upper right of Figure 3. The space is a 6 by 9 grid of possible locations or states, one of which is marked as the starting state, "S", and one of which is marked as the goal state, "G". The shaded states act as barriers and cannot

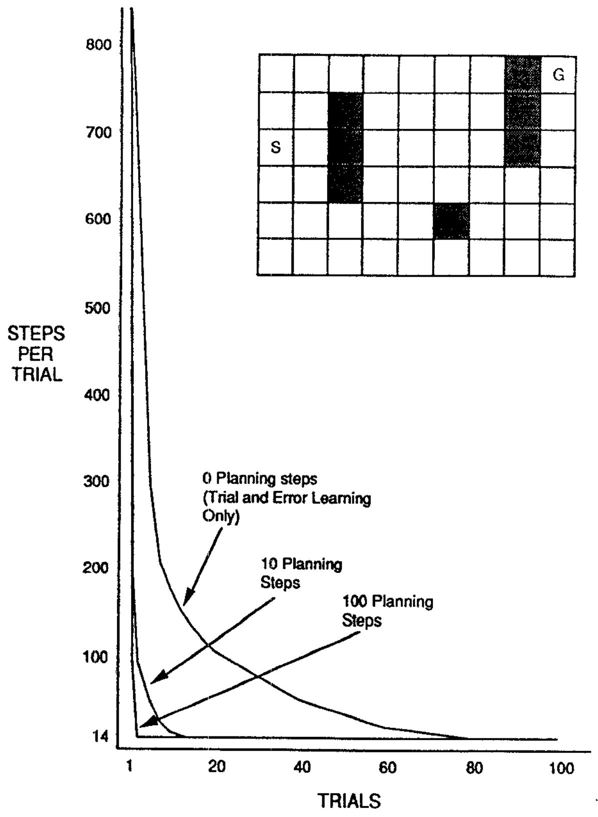


**Figure 3. Learning curves for Dyna systems on a simple navigation task.** A trial is one trip from the start state "S" to the goal state "G". The shortest possible trial is 14 steps. The more hypothetical experiences ("planning steps") using the world model, the faster an optimal path was found.

be entered. All the other states are distinct and completely distinguishable. From each there are four possible actions: UP, DOWN, RIGHT, and LEFT, which change the state accordingly, except where such a movement would take the system into a barrier or outside the space, in which case the location is not changed. Reward is zero for all transitions except for those into the goal state, for which it is +1. Upon entering the goal state, the system is instantly transported back to the start state to begin the next trial. None of this structure and dynamics is known to the Dyna system a priori.

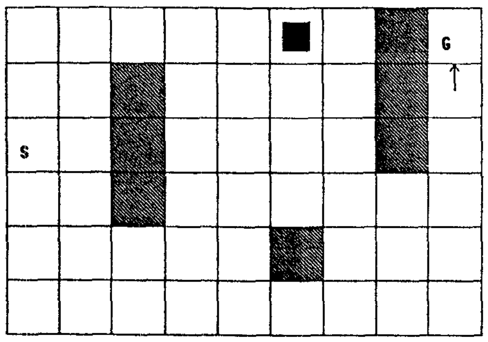
In this demonstration, the world was assumed to be deterministic, that is, to be a finite-state automaton, and the world model was implemented simply as next-state and reward tables that were filled in whenever a new state-action pair was experienced (Step 5 of Figure 2). The evaluation function was also implemented as a table and was updated (Step 7) according to the simplest temporal-difference learning method: $Eval(x) \leftarrow Eval(x) + \beta(e' - e)$. The policy was implemented as a table with an entry $w_{xa}$ for every pair of state $x$ and action $a$. Actions were selected (Step 3) stochastically according to a Boltzmann distribution: $P\{a \mid x\} = e^{w_{xa}} / \sum_j e^{w_{xj}}$. The policy was updated (Step 8) according to: $w_{xa} \leftarrow w_{xa} + \alpha(e' - e)$. For hypothetical experiences, states were selected (Step 1) at random uniformly over all states previously encountered. The initial values of the evaluation function $Eval(x)$ and the policy table entries $w_{xa}$ were all zero; the initial policy was thus a random walk. The world model was initially empty; if a state and action were selected for a hypothetical experience that had never been experienced in reality, then the following steps (Steps 4–8) were simply omitted.

In this instance of the Dyna architecture, the inner loop (Figure 2) was applied alternately to the real world and to the world model. For each experience with the real world, $k$ hypothetical experiences were generated with the model (Step 0). Figure 3 shows learning curves for $k = 0$, $k = 10$, and $k = 100$, each an average over 100 runs. The $k = 0$ case involves no planning; this is a pure trial-and-error learning system entirely analogous to those used in reinforcement learning systems (Barto, Sutton & Anderson, 1983; Sutton, 1984; Anderson, 1987). Although the length of path taken from start to goal falls dramatically for this case, it falls much *more* rapidly for the cases including hypothetical experiences (planning), showing the benefit of using a learned world model. For $k = 100$, the optimal path was generally found and followed by the fourth trip from start to goal; this is very rapid learning. The parameter values used were $\beta = 0.1$, $\gamma = 0.9$, and $\alpha = 1000$ ($k = 0$) or $\alpha = 10$ ($k = 10$ and $k = 100$). The $\alpha$ values were chosen roughly to give the best performance for each $k$ value.

Figure 4 shows why a Dyna system that includes planning solves this problem so much faster than one that does not. Shown are the policies found by the $k = 0$ and $k = 100$ Dyna systems half-way through the second trial. Without planning ($k = 0$), each trial adds only one additional step to the policy, and so only one step (the last) has been learned so far. With planning, the first trial also learned only the last step, but here during the second trial an extensive policy has been developed that by the trial's end will reach almost back to the start state. By the end of the third or fourth trial a complete optimal policy will have been found and perfect performance attained.

This simple illustration is clearly limited in many ways. The state and action spaces are small and denumerable, permitting tables to be used for all learning

WITHOUT PLANNING ($k = 0$)



WITH PLANNING ($k = 100$)

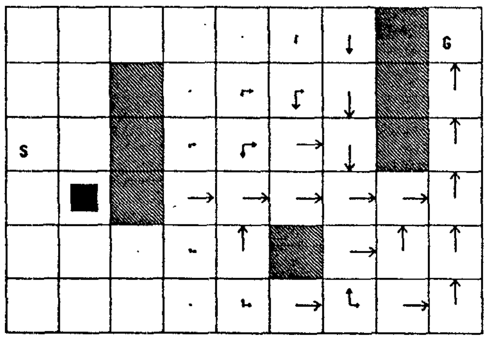


**Figure 4. Policies found by planning and non-planning Dyna systems by the middle of the second trial.** The black square indicates the current location of the Dyna system, and the arrows indicate action probabilities (excess over the smallest) for each direction of movement.

processes, and making it feasible for the entire state space to be explicitly explored. For large state spaces it is not practical to use tables or to visit all states; instead one must represent a limited amount of experience compactly and generalize from it. The Dyna architecture is fully compatible with the use of a wide range of learning methods for doing this. In this example, it was also assumed that the Dyna system has explicit knowledge of the world's state. In general, states can not be known directly, but must be estimated from the pattern of past interaction with the world (Rivest & Schapire, 1985; Mozer & Bachrach, 1989). The Dyna architecture can use state estimates constructed in any way, but will of course be limited by their quality and resolution. A promising area for future work is the combination of Dyna architectures with egocentric or "indexical-functional" state representations (Agre & Chapman, 1987; Whitehead, 1989).

Yet another limitation of the example Dyna system presented here is the trivial form of search control used. Search control in Dyna boils down to the decision of whether to consider hypothetical or real experiences, and of picking the order in which to consider hypothetical experiences. The task considered here is so small that search control is unimportant, and was thus done trivially, but a wide variety of more sophisticated methods could be used. Particularly interesting is the possibility of using the Dyna architecture at a higher level to make these decisions.

Finally, the example presented here is limited in that reward is only non-zero upon termination of a path from start to goal. This makes the problem more like the kind of search problem typically studied in AI, but does not show the full generality of the framework, in which rewards may be received on any step and there need not even exist start or termination states. In the general case, the Dyna algorithm given here attempts to maximize the cumulative reward received per time step.

Despite these limitations, the results presented here are significant. They show that the use of an internal model can dramatically speed trial-and-error learning processes even on simple problems. Moreover, they show how the functionality of planning can be obtained in a completely incremental manner, and how a planning process can be freely intermixed with reaction and learning processes. I conclude that it is not necessary to choose between planning systems, reactive systems and learning systems. These three can be integrated not just into one system, but into a single algorithm, where each appears as a different facet or slightly different use of that algorithm.

**Acknowledgements.** The author gratefully acknowledges the extensive contributions to the ideas presented of his colleagues Andrew Barto and Chris Watkins. I also wish to also thank the following people for ideas and discussions: Steve Whitehead, Paul Werbos, Luis Almeida, Ron Williams, Glenn Iba, Leslie Kaelbling, John Vittal, Charles Anderson, Bernard Silver, Oliver Selfridge, Judy Franklin, Tom Dean and Chris Matheus.

## References

Agre, P. E., & Chapman, D. (1987) Pengi: An implementation of a theory of activity. *Proceedings of AAAI-87*, 268–272.

Anderson, C. W. (1987) Strategy learning with multilayer connectionist representations. *Proceedings of the Fourth International Workshop on Machine Learning*, 103–114. Irvine, CA: Morgan Kaufmann.

Barto, A. G., Sutton R. S., & Anderson, C. W. (1983) Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, *13*, 834–846.

Barto, A. G., Sutton, R. S., & Watkins, C. J. C. H. (1989) Learning and sequential decision making. COINS Technical Report 89-95, Dept. of Computer and Information Science, University of Massachusetts, Amherst, MA 01003.

Bellman, R. E. (1957) *Dynamic Programming.* Princeton University Press, Princeton, NJ.

Craik, K. J. W. (1943) *The Nature of Explanation.* Cambridge University Press, Cambridge, UK.

Dennett, D. C. (1978) Why the law of effect will not go away. In *Brainstorms*, by D. C. Dennett, 71–89, Bradford Books, Montgomery, Vermont.

Howard, R. A. (1960) *Dynamic Programming and Markov Processes.* Wiley, New York.

Mozer, M. C., & Bachrach, J. (1989) Discovering the structure of a reactive environment by exploration. Technical Report CU-CS-451-89, Dept. of Computer Science, University of Colorado at Boulder 80309.

Rivest, R. L., & Schapire, R. E. (1987) A new approach to unsupervised learning in deterministic environments. *Proceedings of the Fourth International Workshop on Machine Learning*, 364–375. Irvine, CA: Morgan Kaufmann.

Ross, S. (1983) *Introduction to Stochastic Dynamic Programming.* Academic Press, New York.

Rumelhart, D. E., Smolensky, P., McClelland, J. L., & Hinton, G. E. (1986) Schemata and sequential thought processes in PDP models. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume II*, by J. L. McClelland, D. E. Rumelhart, and the PDP research group, 7–57.

Russell, S. J. (1989) Execution architectures and compilation. *Proceedings IJCAI-89*, 15–20.

Sutton, R. S. (1984) *Temporal credit assignment in reinforcement learning.* Doctoral dissertation, Department of Computer and Information Science, University of Massachusetts, Amherst.

Sutton, R.S. (1988) Learning to predict by the methods of temporal differences. *Machine Learning 3*: 9–44.

Sutton, R.S., Barto, A.G. (1981) An adaptive network that constructs and uses an internal model of its environment. *Cognition and Brain Theory Quarterly 4*: 217–246.

Sutton, R.S., Pinette, B. (1985) The learning of world models by connectionist networks. *Proceedings of the Seventh Annual Conf. of the Cognitive Science Society*, 54–64.

Watkins, C. J. C. H. (1989) *Learning with Delayed Rewards.* PhD thesis, Cambridge University.

Werbos, P. J. (1987) Building and understanding adaptive systems: A statistical/numerical approach to factory automation and brain research. *IEEE Transactions on Systems, Man, and Cybernetics*, Jan-Feb.

Whitehead, S. D. (1989) Scaling reinforcement learning systems. Technical Report 305, Dept. of Computer Science, University of Rochester, Rochester, NY 14627.

# Abstraction Planning in Real-Time

**Richard Washington** and **Barbara Hayes-Roth**
Knowledge Systems Laboratory
Computer Science Department
Stanford University
Stanford, CA 94305

## Introduction

The basic aim of real-time planning is, given a goal and time constraints, to decide on and execute a series of actions towards that goal under the time constraints. In many realistic applications, planning must also be able to adapt to changes in the world, in the goals, and in the time constraints. The planner should adapt its plan to the changing situation.

Changes in the world are either unexpected effects of the planner's actions or events caused by some external force. These changes may present opportunities for new planning. They may also require the existing plan to be modified or discarded. Complete precompiled plans [Schoppers, 1987] adapt to these changes by storing all possible world states and the appropriate actions for each one, but such plans are impractical in practice [Ginsberg, 1989]. Any other approach needs to be able to replan. This replanning needs to happen under the existing time constraints to maintain the overall real-time performance of the planner.

The goals of the system may change during planning. These changes may be small enough that an existing partial plan could be modified to achieve the new goals, or large enough to require a completely new plan. To maintain real-time planning performance, the planner must be able to adapt to new goals within the time constraints. Approaches that require building a complete plan to achieve goals, such as existing compilation work, require too much time to adapt to new goals.

The time constraints also may vary during planning. Unexpected world changes, new goals, or reduced computational resources may affect the available time. To prove effective under varying time constraints, the planner's performance must improve given more time and degrade gradually given less time. This performance change should be smooth, not catastrophic.

The approach we propose is to use levels of abstraction, with partial plans at each level of abstraction. The steps in the partial plans are sensitive to the situation in which they may be used. When the situation changes, plan steps are added, deleted, or modified as necessary to keep all plan steps consistent with the expected state of the world. At any point, the plan may be expanded further within one level of abstraction, or it may be refined to a lower level of abstraction. The planner chooses a strategy that will provide an executable (partial) plan under the time constraints. However, if time were to run out unexpectedly, the planner could generate an executable action appropriate for the partially-constructed plan. This approach provides the performance and adaptivity that are necessary for effective real-time planning.

This approach to planning using levels of abstraction and partial plans is being implemented as a component of the Guardian system for intensive-care monitoring [Hayes-Roth *et al.*, 1989]. Currently we are using planning for patient therapy. We intend to apply the approach also to control of the system's reasoning.

## Architectural Foundations

In this section we will describe the basic architectural assumptions that underlie our approach.

These have been dictated by the application domain and implementation system, but they are reasonable assumptions for a wide range of tasks.

The basic representation of an event is an interval of time. Events are separated into the classes of *occurred*, *expected*, and *intended*. *Occurred* events are observations and executed actions. *Expected* events are predictions made by the reasoning system, to be confirmed or contradicted by occurred events. *Intended* events serve as goals to guide the reasoning system. Together the intervals form a timeline representing the state of the system's knowledge.

Observations enter the system via a set of sensors. The sensors monitor the world continuously, reporting events judged significant by criteria provided by the reasoning system [Washington and Hayes-Roth, 1989]. The reporting of observations is asynchronous with respect to the reasoning cycle of the system.

Actions to be executed are placed on the timeline along with their expected effects. Once the planner has constructed a plan and put it on the timeline, that plan will be executed unless the planner retracts it. The plan execution is independent of planning: when the time arrives to execute a planned action, it will be executed by the system even when the planner is still extending the plan.

Plans are constructed using states and operators, but the states and operators differ from their traditional meaning. A state contains the system's knowledge and beliefs about the world, including the past, present, and future. The current state contains intervals representing the current observations, expectations, and intentions of the system. Note that these intervals represent the system's beliefs at a particular time, not the instantaneous state of the world at that time. An operator moves the system from one set of beliefs to a new set of beliefs. In a given state, an operator has the effect of suggesting a "future" state that contains the knowledge and beliefs of the given state augmented with the operator's expected effects. The operator's effects appear as expected intervals in the new state (see Figure 1). So a future state in a plan is the result of a sequence of operators, and it represents the beliefs the system would have if it were to execute that operator sequence. The goals of the planner are represented as intentions on the timeline.

Planning competes with other reasoning tasks in the overall system. The system as a whole may apply various reasoning methods to the data it receives. Planning will receive varying resources depending on the importance of the other reasoning. As the situation changes, so may the resources available to planning. In addition, the other reasoning may provide the planner with information that may affect the developing plan.

## Planning as Search

At each level of abstraction, planning is treated as a search problem. The search proceeds "left to right," from the start state to the goals. The motivation for this search order is to maintain at all times a partial plan that is the best given the knowledge and beliefs in the current state. Then if time were to run out, the existing plan would begin execution as the planner worked to extend the plan.

Constructing a plan is performed by adding operators to the existing states and then adding the operators' resultant states. To add an operator to an existing state, the planner first matches the available operators against the state. Operators that match are instantiated for the state. An operator will have as its expected result a new state with the operator's effects appearing as expected intervals.

Since this is a basic search technique, the problems of search control arise. To handle that, an evaluation is performed on each state generated in the plan to determine its progress towards achieving the goals. Using these evaluations, a best-first search strategy such as Real-Time A* [Korf, 1987] will provide heuristic search control to limit the search space (see Figure 2). The levels of abstraction discussed below also help limit the search.

The best plan from a state, as defined by the evaluation, appears as expected actions in the state. The system will execute an action in the best plan from the current state when the appropriate time arrives. When an operator's action is

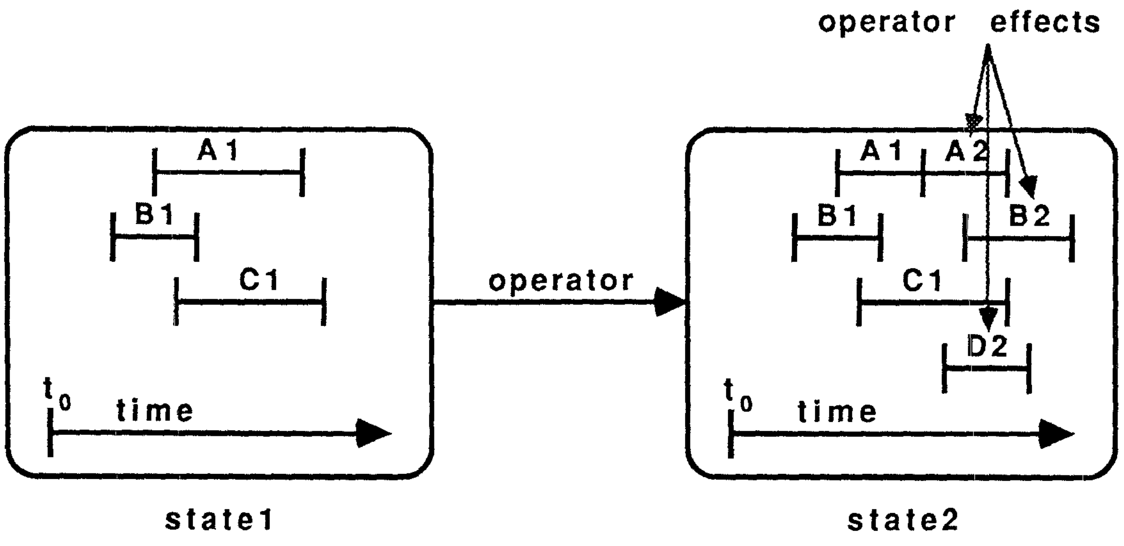


Figure 1: Application of an operator to a state. The new state contains the knowledge and beliefs of the old state, with the operator's expected effects added.

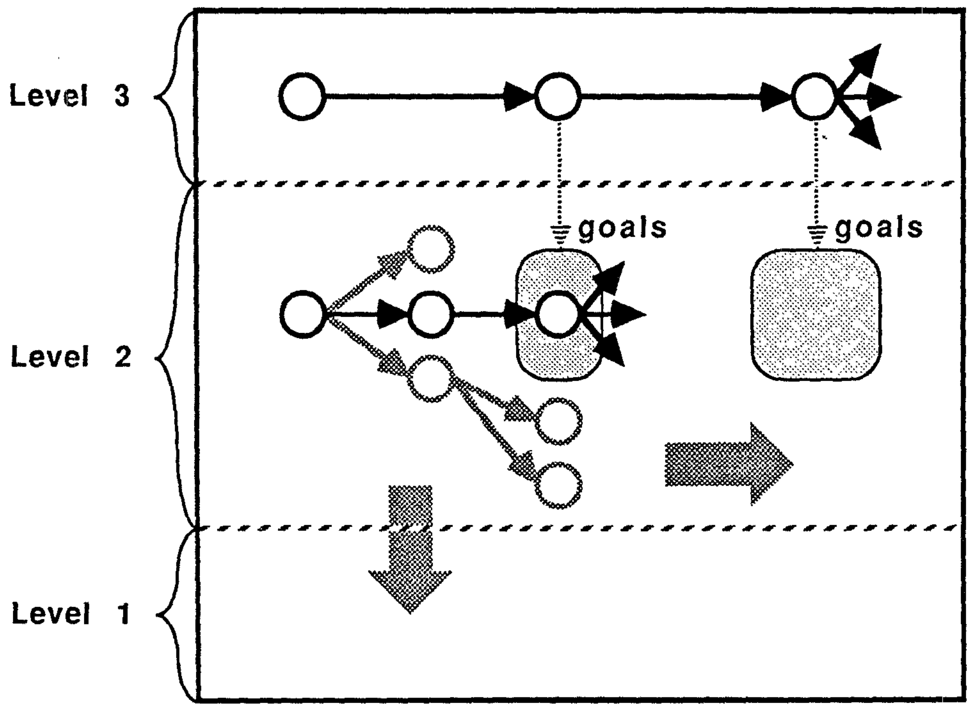


Figure 2: Planning at multiple levels of abstraction. Planning at each level is a best-first search from the current state to the goals. Abstract states are refined to lower-level goals. The plan may be expanded either by searching further within a level or refining an existing plan to a lower level.

executed, the current state will now contain the expected effects of the operator. This in essence moves the current state one step along the plan. Note that executing an action may not actually result in the action's expected effects, so the planner merely adds expectations that must be confirmed by the observations.

The planner adapts to a changing situation because each operator is sensitive to the conditions that make it appropriate. Whenever a situation changes, new operators that rely on the new information may be invoked, and existing operators that relied on the old information may be modified or discarded. For example, an operator in a planned sequence may depend on effects expected as a result of earlier operators. If these effects are contradicted by observations, then the operator can no longer be executed. Thus the structure of the plan will change as needed to remain consistent with current information.

## Planning with Levels of Abstraction

Abstraction is used to guide both planning and replanning. Abstract plans provide goals for constructing lower-level plans. When the situation changes so that lower-level plans fail, the higher-level plans will still offer a general strategy that can be refined into a low-level plan.

A partial plan at one level of abstraction is a sequence of states and operators. Each of these states contains expectations about the effects of the operator leading to the state. At an abstract level, this operator may not be directly executable, but instead it may represent a sequence of lower-level operators for achieving the effects. So at the next lower level, the expected effects are converted to goals to be achieved by a sequence of low-level operators. Since the expected effects are represented as expected intervals in the states, the transformation is thus from states at the abstract level to goals at the lower level (see Figure 2). These goals serve to guide the search in the lower level, augmenting the local evaluation function with more global direction.

This global perspective aids in replanning. Replanning is performed as a form of planning, so the influence of abstraction is the same. In replanning, the more abstract plans lead the lower-level replanning work along a path consistent with the planner's goals. Since the search at each level is adaptive, the highest level plan that needs modification will start to be reconstructed, and that will affect the replanning at the lower levels.

One potential problem in planning at levels of abstraction is reaching executable actions under time stress. Although the plan at each level need not be complete before starting work on the next lower level, the planner may not have refined the plan to an executable level when execution becomes necessary. In this case the planner must quickly find an executable action compatible with its partial, abstract plan. This can be achieved by expanding only the first plan step at successively lower levels of abstraction. Each such plan step would be the best with respect to the goals provided by the more abstract plans, so the executable action would remain consistent with the higher-level goals.

## Strategies for Partial Planning

As the planner searches more deeply at each level of abstraction and refines the plan to lower levels of abstraction, the executable actions are more likely to be useful for the higher-level goals. The decision whether to search to greater depth or to refine to lower levels depends on the characteristics of the domain.

If the world is unpredictable, a long plan is unlikely to execute completely without any unexpected effects. In that case, the time spent building a long plan may be better spent refining a short plan to lower levels. If the time constraints vary widely, then it may be useful to maintain a nearly executable plan at all times. Then if time were to run out unexpectedly, it would take only a few steps to reach an executable action. If the evaluation function is suboptimal, then the best plan at a given level may change as the search progresses, causing changes to the goals of lower-level planning. In that case, a more complete plan at a given level would provide more stability in the plan and less wasted work at lower levels.

The planner should tailor the search and refinement to match the characteristics of the domain, since there is no general strategy that is optimal for all domains. In particular, the planner must provide a flexible way to specify and implement strategies for controlling the plan expansion.

Since execution occurs concurrently with planning, the results of execution could provide additional information that would reduce the search space of the remaining plan. Alternatively, execution of an uncertain plan could prove harmful if the actions are irreversible or costly. Given two partial plans that execute at different times, the planner would need to use information about the costs and benefits of the planned actions to prefer one over the other. Thus the planner should have a way to specify and implement strategies for preferring planning or execution.

## Application Domain

The planning work described here is being applied to the domain of monitoring patients in a surgical intensive-care unit (SICU). In the SICU, a variety of machines measure patient parameters and provide observations continually. Others offer observations upon request. Actions and effects in the SICU take place over time. Deadlines arise to solve problems with the patient, and these deadlines may change with the patient condition. Unexpected events may occur, altering the understanding of the patient's state.

Planning is implemented as a component of the Guardian system for SICU monitoring [Hayes-Roth *et al.*, 1989]. Planning is currently being applied to the task of constructing plans for patient therapy over time. Other components of Guardian perform tasks such as associative diagnosis, model-based diagnosis, and input data management. Execution of the various components may interleave, and the components may compete for computational resources. To manage the overall system behavior, planning will also be applied to the global control problem.

## Conclusion

Real-time planning and replanning can be achieved by using multiple levels of abstraction, with partial planning at each level. The plan at each level is sensitive to changes in the situation, modifying the plan to remain consistent with the evolving world and problem-solving states. The levels of abstraction provide goal-directed guidance for planning and replanning, while retaining real-time performance and adaptivity.

Further work is necessary on strategies for controlling the planning process. Although universal strategies appear unlikely, strategies applicable to a class of problems appear more reasonable. Such strategies should be identified and specified.

## References

[Ginsberg, 1989] M. L. Ginsberg. Universal planning: An (almost) universally bad idea. *AI Magazine*, 10(4):40–44, 1989.

[Korf, 1987] R. E. Korf. Real-time heuristic search: First results. In *Proceedings of AAAI-87*, pages 133–138, 1987.

[Hayes-Roth *et al.*, 1989] B. Hayes-Roth, R. Washington, R. Hewett, M. Hewett, and A. Seiver. Intelligent monitoring and control. In *Proceedings of the 1989 International Joint Conference on Artificial Intelligence*, 1989.

[Schoppers, 1987] M. J. Schoppers. Universal plans for reactive robots in unpredictable environments. In *Proceedings of IJCAI*, pages 1039–1046. IJCAI, 1987.

[Washington and Hayes-Roth, 1989] R. Washington and B. Hayes-Roth. Input data management for real-time AI systems. In *Proceedings of the 1989 International Joint Conference on Artificial Intelligence*, 1989.

# The STRIPS Assumption for Planning Under Uncertainty

Michael P. Wellman

AI Technology Office, Wright R&D Center
Wright-Patterson AFB, OH 45433
*wellman@aagate.avlab.wpafb.af.mil*

## Abstract

The STRIPS assumption bounds the information relevant to determining the effects of actions. It is fundamentally a statement about beliefs, and does not in fact assume anything about the world itself. This treatment separates the STRIPS assumption from other necessary features of a planning architecture, such as its model of persistence and its inferential policies. We can characterize the STRIPS assumption in terms of probabilistic independence, thereby facilitating analysis of representations for planning under uncertainty.

## 1 The Frame Problem and the STRIPS Assumption

The classic dilemma in representing and reasoning about the effects of actions is the *frame problem*, originally identified by McCarthy and Hayes [7]. The frame problem has come to stand for a variety of computational and notational complexities arising from the apparent necessity of considering the possible change in status of every proposition for each action. Characterizations of the problem vary widely [1, 11], proposed solutions even more so, but a kernel of consensus does seem to exist. AI researchers agree that part of the problem, at least, has to do with specifying the effects of actions without explicitly describing all ramifications and qualifications. In particular, we want to avoid a requirement for explicit *frame axioms* specifying the propositions *not* affected by each action.

Actual planners eschew frame axioms and restrict attention to propositions explicitly mentioned in action specifications. Waldinger has named this policy the "STRIPS assumption" [12], after its first application [3]. McDermott [8] asserts that no program since STRIPS has been practically bothered by the frame problem, which is true if we define it as the need for frame axioms in the deductive planning approach. But as McDermott also points out, the frame problem *does* frustrate attempts at logical analyses of these programs, which (many believe) hinders our efforts to build planners capable of reasoning about change in complex environments.

Understanding the nature of the STRIPS assumption and the extent to which it circumvents the frame problem is a first step to understanding the larger issues in reasoning about actions. Previous discussions of the STRIPS assumption tended to encompass all of these issues, failing to distinguish the various roles played by this particular notational convention. While the broader views provide fuller accounts of the planners they address, their analyses are not transferable to planning frameworks that take significantly different approaches to representing and reasoning about change.

For example, Lifschitz [6] focuses on conditions under which STRIPS's add/delete mechanism will be guaranteed to produce only valid plans. The analysis concludes essentially that STRIPS systems are sound as long as

1. the use of non-atomic sentences in operator descriptions and world models is restricted (in a precise manner described by Lifschitz), and

2. a kind of strong persistence holds, where no changes occur except as specified in add and delete lists.

These conditions apply to planning frameworks that adopt the same strong persistence model and forbid inference about the further consequences of specified effects. Many have been unwilling to accept these restrictions, and have worked on methods and semantic accounts of systems that go beyond them.

Research on the task of determining the implications of specified effects of actions (called the *ramification problem*), its counterpart for preconditions (the *qualification* problem), and development of models of persistence are important areas of investigation for AI planning. The point of this paper is that there is a separable aspect of the STRIPS assumption that is orthogonal to these issues, and therefore applicable across a variety of planning frameworks. Generally stated, the STRIPS assumption *per se* dictates that the effect of an action on the world model be completely determined by the direct effects explicit in its specification. By saying only that it is "completely determined," we permit the nature of the implicit effects to vary among planning systems.

I examine this interpretation below from the perspective of planning under uncertainty. Uncertainty provides further motivation for this view of the STRIPS assumption, and concepts from uncertain reasoning help to characterize it more precisely for application to existing planning frameworks.

## 2 Planning under Uncertainty

An agent plans *under uncertainty* whenever it cannot flawlessly predict the state of the environment resulting from its actions. By this definition, uncertainty is a characteristic of the agent's knowledge rather than an inherent property of the environment. Given that we are never likely to achieve perfect prediction in realistic environments, all planning is actually performed under uncertainty; planning under *certainty* is an unrealizable—yet often useful—idealization.

The frame problem arises in planning under uncertainty just as it does in the idealized framework. Planners need something like the STRIPS assumption to justify leaving non-effects of an action implicit in their omission from the action's specification. However, semantic accounts of the STRIPS assumption in classical planning (e.g., STRIPS itself [6]) do not easily map over to the uncertain case. The conventional informal interpretation—that planners assume relations in the world model are unchanged unless explicitly specified otherwise—does not apply to planners that admit to incomplete knowledge about the effects of their actions.

Perhaps we could modify the interpretation to assume that changes of unspecified relations are unlikely rather than impossible. The problem of this approach is identifying a particular, well-motivated, likelihood assumption that is sufficiently general for domain-independent planning. As demonstrated by work along these lines [2, 4], defining such a convention is tantamount to adopting a model of persistence and probabilistic inference. Moreover, these persistence models tend to be more varied and complicated than those proposed for planning under certainty. These differences provide further motivation for a characterization of the STRIPS assumption independent of a particular model of persistence.

The essential property of the STRIPS assumption that justifies implicit treatment of non-effects is the presumption that the information specified explicitly is sufficient to describe the agent's change in belief. In other words, once the direct effects are known, knowledge of the action itself is superfluous for purposes of prediction. Thus, the STRIPS assumption is fundamentally a statement that the agent's beliefs about changes in propositions not mentioned in an action's specification are independent of the action, given those effects explicitly specified. For planning under uncertainty, we can characterize beliefs in terms of probability distributions and use the concept of probabilistic independence to formalize this interpretation of the STRIPS assumption.

## 3 Probabilistic Independence

In a state of uncertainty, an agent's beliefs are representable by a probability distribution over possible situations (which is not to say that the agent's beliefs need be encoded as such). We take situations to be assignments on a universe of variables describing the world, including such things as what actions are performed and their consequences. Beliefs are then probability distributions over this space. Note that this framework avoids imposing a temporal ontology, which would provide essential structure for the planning problem but would detract from the generality of our analysis of the issue at hand.

To capture the meaning of the STRIPS assumption proposed above, we need a way to express the sufficiency of explicitly specified effects to describe the full impact of an action on the agent's beliefs about the world. For this purpose, the natural concept in probability theory is *conditional independence.* We say that random variables $x$ and $y$ are conditionally independent given $z$ iff

$$\Pr(x|y,z) = \Pr(x|z) \qquad (1)$$

for any possible values of the variables. In other words, once the value $z$ is known, finding out the value of $y$ has no effect on the agent's belief about $x$. In this case, $y$ is superfluous information.

The STRIPS assumption is paraphrased by a schema for equation (1). Performance of an action is represented by $y$, $z$ stands for the explicit effects of $y$ plus the "background," and $x$ represents "everything else." The independence assertion is that for a given background, knowing the explicit effects of $y$ provides all the information useful for predicting its implicit effects, that is, everything else. Given its explicit effects, knowledge about the action's performance is redundant.

For a satisfactory interpretation, we need a more complete account of concepts like "background" and "everything else." To understand their role in planning systems, we investigate a class of representations for actions and events based on graphical models. Graphs provide a formal language for expressing (via adjacency) the locality of explicit effects in planning representations.

## 4 Dependency Graphs

A *probabilistic network* (also called *Bayesian* or *belief network* [9] or *influence diagram*) is a directed acyclic graph (DAG) with nodes for the random variables connected by links indicating probabilistic relations. Associated with each node is a probability distribution for its variable given the possible values for its predecessors in the graph. Thus, a link from $x$ to $y$ indicates that $y$ might depend probabilistically on $x$. Conversely, the absence of links restricts the dependencies that can be encoded in the network. The graphical condition for conditional independence in probabilistic networks is called *d-separation* [9, 10]. Two nodes $x$ and $y$ are d-separated by a set of nodes $Z$ in a DAG iff for every *undirected* path between them either:

1. there is a node $z \in Z$ on the path with at least one of the incident edges leading out of $z$, or
2. there is a node $z'$ on the path with both incident edges leading in, and neither $z'$ nor any of its successors are in $Z$.

A dependency graph for which all d-separations are valid conditional independencies is called an *I-map*. Although any joint distribution can be represented graphically by some probabilistic network (which are all I-maps), the most efficient representations are those without superfluous links, called *minimal* I-maps.

We can characterize the independence condition underlying the STRIPS assumption in terms of these dependency graph concepts. Consider a probabilistic network with variables for all actions and events relevant to the planner. Every action node has an outgoing link exactly to those events explicitly represented as direct effects. Events may have arbitrary connections among themselves, as dictated by some world model (outside the scope of discussion here). Action nodes have no incoming links, reflecting our presumption that the planner has control over which actions are to be performed.

The STRIPS assumption is that the graph so constructed is an I-map. Let $S_a$ be the set of event variables that action variable $a$ directly affects, $a$'s immediate successors in the dependency graph. By virtue of $I$-mapness, $a$ is conditionally independent of any $e \notin S_a$ given $e$'s predecessors (see, for example, [14, Lemma 4.1]). Each predecessor $d$ of $e$, in turn, is either a direct effect of $a$ or is conditionally independent given its own predecessors. Ultimately, the effect of $a$ on $e$ is completely determined by $a$'s direct effects and $e$'s relation to them. Note that we still need to describe the interaction, if any, between $a$ and $e$ in their joint effects.

The probabilistic STRIPS assumption does *not* require that $a$ be conditionally independent of $e$ given the direct effects $S_a$, or even by any subset of $S_a$. In Figure 1, for example, $a$ and $e$ are d-separated by $\{s, b\}$ but by no other variable set. The variable $b$ is necessary for conditional independence of $a$ and $e$ even though $b$ itself is unconditionally independent of $a$.

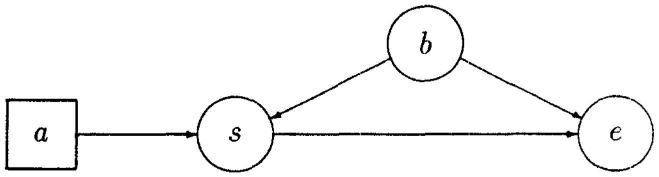


Figure 1: Action $a$ is conditionally independent of $e$ given $Z_a = \{s, b\}$ but not given any subset of its direct effects $S_a = \{s\}$.

If we enlarge the conditioning set to include predecessors of $a$'s direct effects, however, we get another valid independence condition. Let $Z_a = S_a \cup B_a$, where $B_a$ (the "background") is the set of variables that affect $a$'s direct effects:

$$B_a = \bigcup_{s \in S_a} predecessors(s) - \{a\}.$$

The d-separation condition implies that $a$ is conditionally independent of $e$ given $Z_a$. In the graph of Figure 1, for example, the background $B_a = \{b\}$, and $Z_a = \{s, b\}$.

The dependency graph model permits us to formalize the STRIPS assumption in terms of probabilistic conditional independence. In particular, there must exist an I-map of variables in the world model where any variable $e$ not specified as an effect of action $a$ is not directly connected to $a$. Under this condition there may be a probabilistic dependency between $a$ and $e$ in some situations, but this can always be described in terms of $a$'s and $e$'s relations to $S_a$.

We can apply the graph construction to the informal statement of the independence condition given in the previous section. Filling in the terms, our statement is that the complete effects of an action $a$ are fully specified by the direct effects, $S_a$, and the background, $B_a$. Everything else, $e$, is implicit in these variables. That is, $e$ is conditionally independent of $a$ given $S_a$ and $B_a$.

## 5 Applications

The conditional independence interpretation is valuable tool for studying specific planning systems and validating their use of the STRIPS assumption. A practical prerequisite for applying these results is identifying the relevant background context, $B_a$, for the various planners. Note that while planners adopt different policies regarding *how* the implicit effects are derived from the explicit effects and background, validity of the STRIPS assumption does not depend on these policies.

In the following sections I illustrate the application of the independence concepts by analyzing aspects of three planning systems. The planners examined differ in their probabilistic or deterministic representations for the effects of actions, as well as the type of temporal structure imposed on the planning environment.

## 5.1 SUDO-Planner

SUDO-PLANNER [13] uses ***qualitative probabilistic networks*** (QPNs) [14], abstractions of the models described above, for representing and reasoning about the effects of actions. When introducing actions and events of interest, the planner modifies the structure of the existing network to preserve the model's validity. One class of constructs appearing in SUDO-PLANNER's knowledge base, called ***Markov influences***, specify the effect of an action on an event variable and its dependence on the previous value of that variable.

For example, consider a QPN for a medical therapy problem that includes a variable for the extent of a patient's coronary artery disease (CAD). One action considered by the planner is a coronary artery bypass graft (CABG): bypass surgery to alleviate the coronary disease. The effect of CABG is to decrease CAD (in a precise probabilistic sense [14]). Furthermore, the Markov influence specifies that the decrease is greater for patients with more severe CAD initially. This relationship refers to the variable CAD at two distinct points in time—before and after CABG—and thus cannot be captured by simply adding CABG to the network. Instead, SUDO-PLANNER modifies the QPN by splitting CAD into two variables, $CAD_1$ and $CAD_2$. Figure 2 diagrams the result of this *mitosis* process. CABG negatively influences $CAD_2$, which is otherwise positively related to its value before surgery, $CAD_1$. The boxed minus sign indicates the synergistic interaction of CABG with $CAD_1$. Predecessors of the original CAD variable are connected to $CAD_1$, while its successors before processing the Markov influence are transferred to $CAD_2$.

This process has direct implications for conditional independence (which indeed was the reason for calling them *Markov* influences). Specifically, influencers of the original variable CAD are independent of $CAD_2$ given $CAD_1$, and CAD's original influences cannot depend on $CAD_1$ given $CAD_2$. (The reason is that any path be-

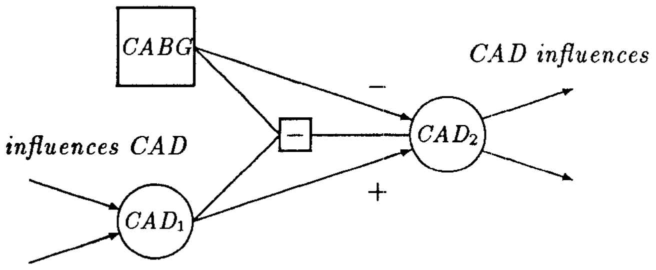


Figure 2: The Markov influence of CABG on CAD.

tween influences and influencers that circumvents the $CAD_i$ variables must include at least one node with both incident edges leading in.) These conditions in turn imply that any variable in the network is independent of CABG given $CAD_1$ and $CAD_2$.

More generally, suppose the action $a$ is defined exclusively by Markov influences on a set of event variables $E$. The STRIPS assumption dictates that the effects of $a$ be completely captured by these influences. The corresponding independence condition is that any other event be conditionally independent of $a$, given $Z_a = S_a \cup B_a$, where the direct effects $S_a = E_2$, the second halves of the split event variables, and the background $B_a = E_1$, the first halves produced by SUDO's variable mitosis process.

## 5.2 Markov Influence Diagrams

Kanazawa and Dean [5] propose a framework for planning under uncertainty based on "causal models," influence diagrams with the Markov property and some other features inessential for our purposes. In a Markov influence diagram, there is a node corresponding to every proposition of interest at every distinguished instant of time. The Markov property is enforced by permitting nodes at time $t$ to depend only on nodes from time $t-1$. If this convention applies to actions as well, then any event at time $t$ is d-separated by actions from time $t' < t$ by the action's direct effects (all at time $t'+1$), plus the events of time $t'$.

With respect to our statement of the STRIPS assumption, the background is (conservatively) the state of the world at the time of the action. The direct effects are the events from the next time point with links from the action node.

## 5.3 STRIPS

Instantiated propositionally for a finite world, we find that a STRIPS model is actually a degenerate kind of Markov influence diagram. The Markov property follows from the linearity of the situation calculus framework [7]. The propositions at a situation $s$ are deterministic functions of those of the previous situation. The persistence model of STRIPS is that for a given proposition this function is the identity in the absence of an action performed at $s$ affecting that proposition. Thus, the background required for any proposition is only its value in the previous situation.

For deterministic variables, there is a stronger graphical criterion for conditional independence, called *D-separation* (note uppercase) [10]. Although the independence condition for STRIPS's simple graph structure is trivial, the more powerful criterion might be useful for analyzing STRIPS-like systems that permit logical and perhaps probabilistic relations among propositions.

It is not surprising that the analyses of these systems all appeal to some sort of Markov property. Any temporal structure on a pattern of conditional independence can be properly termed Markovian.

## 6 Summary

The interpretation presented here provides a new perspective on the STRIPS assumption, constraining the semantics of a planner's knowledge base of actions and events. Essentially, it mandates that the implicit consequences of an action be completely specified by its direct effects. Although described and motivated in terms of probabilistic conditional independence, the interpretation has implications for planning systems regardless of whether they employ probabilistic representations.

The main advantage of this approach is that it distinguishes the concept of belief dependency from the model of persistence of events in the world. It does not obviate the need for such a persistence theory, though it renders the issue orthogonal to the STRIPS assumption *per se*.

The most important limitation of the analysis is that dependency graphs are an inherently propositional representation. Application to planning systems with quantified constructs (any nontrivial action and event representation) requires some instantiation mechanism. A potential solution approach is to apply the axioms of conditional independence directly; the axiomatic theory may be stronger than the graphical even for the propositional case [10].

## References

[1] Frank M. Brown, editor. *The Frame Problem in Artificial Intelligence: Proceedings of the 1987 Workshop*. Morgan Kaufmann, 1987.

[2] Thomas Dean and Keiji Kanazawa. Probabilistic temporal reasoning. In *Proceedings of the National Conference on Artificial Intelligence*, pages 524–528, 1988.

[3] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[4] Steven John Hanks. Projecting plans for uncertain worlds. Technical Report YALEU/CSD/RR 756, Yale University, January 1990.

[5] Keiji Kanazawa and Thomas Dean. A model for projection and action. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 985–990, 1989.

[6] Vladimir Lifschitz. On the semantics of STRIPS. In Michael P. Georgeff and Amy L. Lansky, editors, *Reasoning about Actions and Plans: Proceedings*, pages 1–9. Morgan Kaufmann, 1986.

[7] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969.

[8] Drew McDermott. AI, logic, and the frame problem. In Brown [1], pages 105–118.

[9] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[10] Judea Pearl, Dan Geiger, and Thomas Verma. Conditional independence and its representations. *Kybernetika*, 25:33–44, 1989.

[11] Zenon W. Pylyshyn, editor. *The Robot's Dilemma: The Frame Problem in Artificial Intelligence*. Ablex, 1987.

[12] Richard Waldinger. Achieving several goals simultaneously. In E. Elcock and D. Michie, editors, *Machine Intelligence 8*, pages 94–136. Edinburgh University Press, 1977.

[13] Michael P. Wellman. *Formulation of Tradeoffs in Planning Under Uncertainty*. Pitman and Morgan Kaufmann, 1990.

[14] Michael P. Wellman. Fundamental concepts of qualitative probabilistic networks. *Artificial Intelligence*, to appear.

# PLANNING, REPLANNING, AND LEARNING WITH AN ABSTRACTION HIERARCHY

**Hua Yang** and **Douglas H. Fisher**
Computer Science Department, P. O. Box 1679, Station B
Vanderbilt University, Nashville, TN 37235
hua@vuse.vanderbilt.edu

**Hubertus Franke**
Electrical Engineering Department, Vanderbilt University

## Introduction

In traditional planning scenarios, a sequence of primitive operations (i.e., a plan) is derived from domain knowledge on the applicability of operators and a world model that captures the status of the environment. Most planning systems rely on the closed-world assumption, that is the presence of a complete environmental model. However the real world is often unpredictable. The model may be incomplete or inconsistent because of unanticipated changes in the environment. Thus, during execution an operation can fail to yield its expected results. Recovery from failure and replanning must be considered as an important component of real-world planning systems.

Planning and replanning are often search-intensive processes, with costs that cannot be well managed under real-world time constraints. However, with an appropriate organization of operators and plans the cost of planning can be significantly mitigated. This paper proposes that *conceptual clustering* methods can organize operators and plans into 'similarity' classes based on their shared applicability conditions and effects, thus facilitating the efficient reuse of appropriate past experiences for purposes of planning and replanning.

## Background

Automated acquisition and organization of plan knowledge has been investigated by many researchers. Vere's THOTH (1980) induces a minimal set of relational operators that cover a training set of state to state transitions. For example, having observed the many individual transitions required to build a block tower, THOTH might formulate abstract operator descriptions that correspond to the 'classic' operators of `Stack`, `Pick-up`, etc. However, THOTH does not have a strong notion of 'good' operator organization, other than to discover a minimal set of abstractions that cover the training examples. Nonetheless, THOTH's ability to autonomously discover operator 'classes' make it an early conceptual ancestor of the clustering approach that we propose.

Unlike THOTH, STRIPS (Fikes, Hart & Nilsson, 1972) begins with a set of abstract operator descriptions and conjoins them using means-ends analysis to form plans. Moreover, STRIPS generalizes the applicability of these plans (using analytic methods in contrast to THOTH's empirical approach) and stores them for reuse. However, recent work in learning to plan indicate that a STRIPS approach to saving plans in an unconstrained manner may actually have detrimental effects on planning time: the time to search for applicable past experience may eventually surpass the cost of planning from scratch (Minton, 1988).

Anderson and Farley (1988) suggests a possible way to mitigate the cost of finding applicable past knowledge. Their system, PLANERUS, generates a hierarchy based on common ADD conditions of STRIPS-like operators. ADD condition indices allow PLANERUS to find operators that reduce *differences* in a means-ends planner. In principle a discrimination net over ADD conditions can be very efficient, but like THOTH, PLANERUS appears to lack a strong prescription of operator class quality: its indexing method appears to require an exponential number of indices in the worst case because it groups operators based on combinations of one or more shared conditions. In this regard Minton (1988) points out that even with indexing schemes, systems must also be willing to dispose of past experiences (e.g., abstractions, ADD-condition combinations) that prove to be of low *utility* (e.g., infrequent).

THOTH, STRIPS, and PLANERUS are important precursors to our work, but we hope to extend the ideas illustrated by these systems in several directions. First, a system like PLANERUS is designed primarily to facilitate goal-driven behavior, as its exclusive reliance on ADD-condition indexing indicates. However, work in *reactive* or *situated* planning (Schoppers, 1989) suggests that the current situation should also influence the selection of applicable operators: an ideal operator is one that achieves desirable goals *and* requires minimal al-

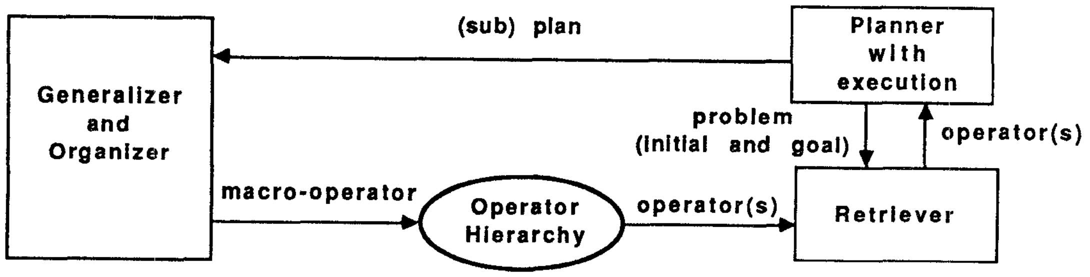


Figure 1: PLOT system: Planning and Learning with Operator Trees

terations to the current situation to do so. Thus, we propose that when using STRIPS-like operators, PRE conditions, as well as ADD conditions should be used to retrieve operators that make progress towards the goal and that best fit the current conditions of the environment. In addition, operator class discovery and indexing should be controlled by a strong heuristic prescription of high utility operator and plan classes. Without these prescriptions, planning with or without the benefit of previous experience remains a search-intensive, often intractable enterprise (Ginsberg, 1989).

## Overview of Our System

Our system, PLOT, is designed to plan, replan in the face of execution failures, and efficiently exploit previous experience in these endeavors whenever possible. Figure 1 illustrates the system structure. There are four basic components in our planning/learning system. Planner constructs new plans, monitors executions, and fixes incorrect plans. Execution and planning are integrated so as to facilitate reactive planning. Retriever gets potential operators for Planner from the Operator Hierarchy. The solution found by Planner is generalized and stored into the Operator Hierarchy. These generalized plans are used in subsequent planning. We now turn to a more detailed description of the system.

## Knowledge Organization

Operators (primitive and macro) are typically specified by their requirements of and effects on the world model; we assume a STRIPS representation of *PRE-*, *ADD-* and *DELETE-* lists. Operators specified in this manner can be organized for efficient reuse through conceptual clustering methods. Conceptual clustering was proposed by Michalski and Stepp (1983) as a method to group observations (objects, situations, facts) into classes based on a 'similarity' measure. In particular, we use an extension of Fisher's COBWEB system (1987), that incrementally forms an abstraction hierarchy of operators based on similarity over PRE- and ADD- lists.[1] Operator classes are represented as nodes in the hierarchy; operators are grouped together using *category utility* (Gluck & Corter, 1985), which is a probabilistic measure of the *increase* of information that can be predicted about members of classes. In planning we want to maximize the ability of the planner to predict both operator effects (ADD conditions) and applicability (PRE conditions). The utility of an operator class, $C_k$, is given by a measure of the expected gain in the planner's ability to predict the conditions (predicates) of each (PRE, ADD) list. For each list the function, $\sum_{j=1}^{m}[P(pred_j|C_k)^2 - P(pred_j)^2]$, is computed where $P(pred_j|C_k)$ stored in node $C_k$ is the proportion of $C_k$ members for which predicate, $pred_j$, is true. Assuming that $C_k$ is one class (i.e. a node) in an abstraction hierarchy, $P(pred_j)$ is the probability that $pred_j$ is true of a member of $C_k$'s parent node; thus, the difference of the two is a measure of information gain provided by $C_k$. 'Good' operator classes are those that have high expected gains over the *PRE* and *ADD* lists.

An operator is added to the hierarchy by using category utility as a partial matching function – the operator is incorporated into the class whose predicate distributions (probabilities) are best reinforced by the operator's conditions. Class distributions are then adjusted dynamically to reflect the new operator's addition. This process is recursively applied so that the operator is classified down the classification tree. If an operator does not sufficiently match any existing class then it may be used to initialize a new operator class. Primitive and macro operators reside at the leaves. Interior nodes of the hierarchy represent operator classes (henceforth, abstract operators) by the conditions that they share over all the lists.[2] Each class defines an abstract operator over its children. In general, the probabilistic, partial-matching capabilities of PLOT are important in domains with uncertainty. It is unlikely that operators will always bring about identical results from identical conditions and in many cases conditions are not completely known.

[1]DELETE-lists are not currently considered in class formation as they have not proved useful in operator retrieval, though they remain part of operator description and are used in planning.

[2]Actually, we use a probabilistic representation that allows exceptions to common conditions.

## Planning and Replanning

An operator hierarchy is the key component for managing knowledge in our system. We assume that PLOT is initially given a set of primitive, operator definitions. These operators are then organized into a hierarchy by the methods and heuristics that we sketched above. This initial hierarchy is then used to limit the search for operators. Given a planning problem in the guise of a current state and goal state, operator retrieval is performed by classifying the (appropriately formatted) current and goal conditions against the first level of the hierarchy. In particular, PLOT factors initial and goal conditions into PRE and ADD lists: the *initial* state conditions become the PRE-list, and the *goal* state conditions minus the *initial* become the ADD-list.

Classification returns an ordered list of best-matching (possibly abstract) operators that appear to bring about desirable changes under current conditions. The means-ends planner selects the 'best' matching operator for examination. One of the following situations is possible.

1. If the operator is a primitive or macro operator, it will be tried to solve the given problem. If it can achieve some of the goals and it does not violate protection constraints, then planning recurses on the subgoals required by the selected operator's preconditions (if necessary) and again on the unsatisfied conditions of the original goal.
2. If the operator is an abstract operator, refinement is performed by recursively classifying the current/goal conditions down another level of the tree (i.e., using the current abstract operator as the root of classification). After getting a list of ordered operators, the planner picks up the 'best' one, and begins the examination procedure for that operator.
3. Otherwise, the operator is not applicable. The next operator in the list is selected for examination. If there are no operators available, backtracking is invoked, and an alternative solution path will be tested.

Subgoals are achieved recursively by the same procedure we use to achieve the main goal. In summary, a plan is constructed via a hill climbing search, but one that allows backtracking (i.e., a heuristically-guided depth-first search) by selection from alternative operators returned from classification.

Operator retrieval also facilitates replanning in the face of execution failures. However, a plan execution failure suggests that certain operator effects proved not to be reliable. In fact, the environment may have changed in some unknown manner. Thus, rather than backtracking to old choice points that were retrieved under conditions that may no long be valid, a better strategy at failure time is to reassess the environment and replan from the current state.

## Learning and Reusing Old Plans

Following successful planning the system can incrementally assimilate newly discovered plans into the abstraction hierarchy. In particular, the plan is mapped onto a three-list representation and sorted down the hierarchy based on its similarity to current operator classes. For example, consider the following plan:

`gotod(dr); open(dr); gothrudr(dr).`

The PRE conditions of this plan are those PRE conditions of the plan's individual operators that are not introduced in the body of the plan. The plan's ADD list is composed of those conditions that are ADDed by constituent operators and that are not subsequently DELeted. The plan's DELete list contains conditions of the previously computed plan PRE list that are subsequently DELeted in the body of the plan. For our example, the three lists on which we will base classification are:

PRE: `inroom(Robot,room1), status(dr,closed), connect(dr,room1,room2)`

DEL: `inroom(Robot,room1)`

ADD: `inroom(Robot,room2)`

Thus the newly-acquired knowledge is incorporated into an existing hierarchy. Figure 2 shows an operator hierarchy with the above plan (MOP-2) and additional one (MOP-1: gotod; gothrudr) incorporated. A new class was created for these two new macros because they share more similarities than any other operators.

Learning also can occur during planning: whenever a subgoal has been achieved the partial solution can be learned. This process is a simple recursive extension of the process that incorporates complete plans, but it has the great advantage of allowing 'within-trial' learning: the reuse of plan fragments within the construction of the same global plan.

There are two kinds of learning implicit in the planning and hierarchy incorporation procedures described above. Along a 'horizontal' dimension, sequentially related operators are concatenated into a single *macro-operator*. Macro-operators can be reused as components to solve more complicated problem, thus increasing the problem-solving power of the planner (Fikes, Hart & Nilsson, 1972). However, unless it is generalized the macro may have little applicability in future problem solving; thus, a 'vertical' dimension is concerned with generalizing the macro. STRIPS used an analytic method the maximally generalize the plan. In a more subtle way PLANERUS also uses a simple analysis of ADD condition commonalities around which it forms operator classes. In contrast, we generalize plans empirically by clustering them into an existing conceptual hierarchy. The advantage of an empirical approach is that it is sensitive the 'structure' of the environment; the probabilistic representation will tend to weight the importance of operator classes by the frequency that they

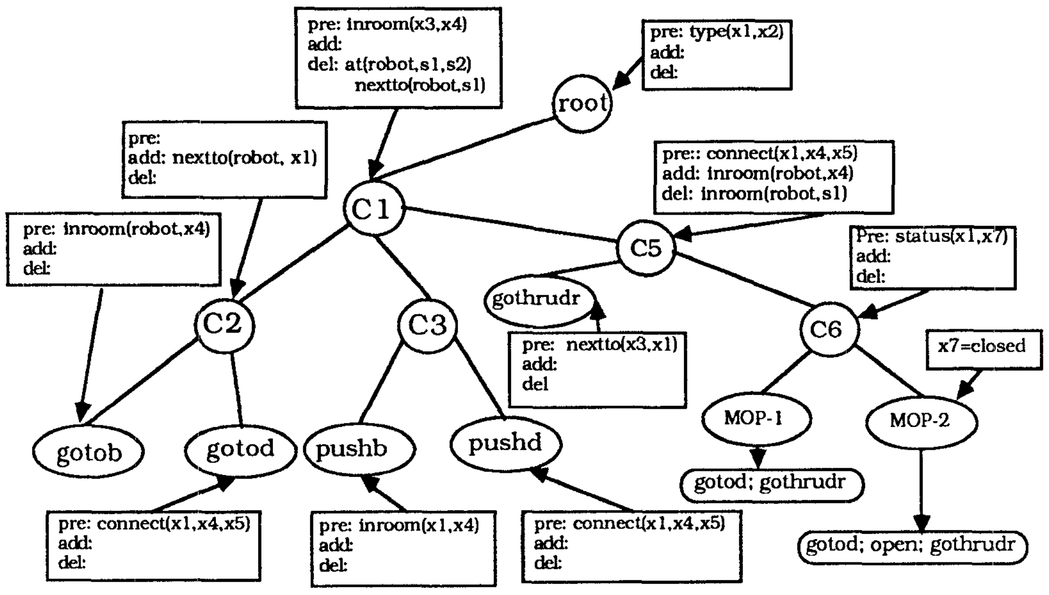


Figure 2: Partial Operator Hierarchy with Macro-Operators

prove useful in the environment. Moreover, an empirical approach will not enumerate operator classes that will never prove useful in the domain since they will never be constructed. Finally, empirical strategies can in principle track environmental changes and/or adapt an initially 'incorrect' set of operator specifications to better model the actual structure of the environment. To do so, it is important that plans not be assimilated into memory until after they have been successfully executed, perhaps with the intervention of replanning mechanisms triggered by execution failures.

## Discussion

Our means-ends planner was developed as a vehicle for testing the efficiency facilitated by an operator/plan hierarchy. Our initial experiments with the system have uncovered several advantages and problems. Most notably, operator retrieval does a good job of excluding irrelevant operators and retrieving relevant operators: those that will eventually participate in the final plan. However, in the blocks-world examples that we have tested the 'best' operator retrieved from the hierarchy often is among the last operators of the final plan, since they often ADD a bulk of the final goal conditions. Unfortunately, planning around this initially selected operator will often cause excessive backtracking because of subgoal interactions. For example, if a plan calls for a robot to push material from a room A to a room B, then the initially chosen operator of plan expansion may call for the robot to go from room A to B (without the material), thus requiring a later backtrack to insure that other conditions of the plan can be satisfied (e.g., the robot must be in room A in order to push material through to B).[3]

There appear to be two approaches to overcoming this problem. The first strategy seeks to salvage the depth-first means-ends approach by differentially evaluating (qualitatively or quantitatively) PRE condition and ADD condition matches depending on the state of plan construction. PRE condition matches may be preferred early in construction with ADD conditions taking priority later in plan (and subplan) construction. An alternative is to exploit the fact that relevant operators are well targeted during retrieval, and to employ a deferred-commitment planner that simultaneously considers the applicability of all retrieved operators and (re)orders them as constraints dictate (Waldinger, 1977; Sacerdoti, 1975).

Another difficulty with our approach relates to the utility problem (Minton, 1988) described earlier. Thus far, our system does not eliminate this problem. To overcome this problem, some techniques must be applied to retain only useful operator classes. During classification, we have methods (again, based on our work in conceptual clustering) of selectively utilizing past plans by descending to an 'optimal' point of abstraction in the hierarchy. We are going to investigate this problem in the future.

[3]Of course once a plan is formed it may be reused so as to mitigate this ordering problem on similar problems, but nonetheless this undesirable phenomena appears to be a possibility on any subsequent, relatively novel problem.

Finally, there are important similarities and differences between our strategy and a variety of other approaches. In fact our approach can be viewed as a hybridization of some previous approaches. Our strategy shares the general strategy of trying to maximally reduce important differences with means-ends planning strategies, but an attempt is made to initially retrieve a set of applicable operators that collectively satisfy many of the subgoals in a current problem simultaneously. Like hierarchical planning the selected operators may be abstract, thus requiring step-wise refinement; However, in our system the plan abstraction space is organized autonomously through conceptual clustering. However, as of now we have not exploited deferred commitment techniques that are present in many of these systems. Finally, we view planning as a process of memory-based classification as do case-based approaches. However, our approach appears to have the desirable and emergent effect that if little is known about the current situation then retrieval of primitive operators will simulate weak methods, but specialized plans can bring about greater gains as experience accumulates. In many of the above respects our work is related to several recent research efforts. Of these it is most similar to Allen and Langley's DAEDALUS (1989); an operator hierarchy is constructed to support a means-ends analysis planner. It indexes primitive operators and plans by the difference they reduce, in a strict means-ends analysis manner.

## Acknowledgments

We thank Jungsoon Yoo, Pat Langley, and John Allen for interesting and insightful discussions on this material. This research is supported by NASA Ames grant NCC 2-645.

## References

Allen, J. and Langley, P. (1989). Using Concept Hierarchies to Organize Plan Knowledge. *Proceedings of the Sixth International Workshop on Machine Learning*, Ithaca, New York, Morgan Kaufmann.

Anderson, J. S. and Farley, A. M. (1988). Plan Abstraction Based on Operator Generalization. *Proceedings of the Seventh National Conference on Artificial Intelligence*, (pp. 100-104). St. Paul, MN: Morgan Kaufmann.

Fikes, R. E., Hart, P. E. and Nilsson, N. J. (1972). Learning and Executing Generalized Robot Plans. *Artificial Intelligence*, 3, 251-188.

Fisher, Douglas H. (1987). Knowledge Acquisition Via Incremental Conceptual Clustering. *Machine Learning*, 2, 139-172.

Ginsberg, M. L. (1989). Universal Planning: An (Almost) Universally Bad Idea. *AI Magazine, Vol 14, No 4, Winter 1989*, 40-44.

Gluck, M. A., and Corter, J. E. (1985). Information, Uncertainty and the Utility of Categories. *Proceedings of the Seventh Annual Conference of the Cognitive Science Society, 283-287.* Irvine, CA: Lawrence Erlbaum.

Michalski, Ryszard S. and Stepp III, Robert E. (1983). Learning from Observation: Conceptual Clustering. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach.* Los Altos, CA: Morgan Kaufmann.

Minton, S. (1988). Quantitative Results Concerning the Utility of Explanation-Based Learning. *Seventh National Conference on Artificial Intelligence.* (pp. 564-569). St. Paul, MN: Morgan Kaufmann.

Sacerdoti, E. D. (1975). A Structure for Plans and Behavior. *Tech. Note 109, AI Center, SRI Internation, Inc. Calif* (Doctoral Dissertation).

Schoppers, M. J. (1989). In Defense of Reactive Plans as Caches. *AI Magazine, Vol 14, No 4, Winter 1989*, 51-60.

Vere, S. (1980). Multilevel Counterfactuals for Generalization of Relation Concepts and Productions. *Artificial Intelligence 14*, 139-164.

Waldinger, R. (1977). Achieving Several Goals Simultaneously. In E. W. Elcock and D. Michie (Eds.), *Machine Intelligence 8.* New York: Halstead/Wiley.

Yang, H. and Fisher, D. (1989). Conceptual Clustering of Means-Ends Plans. *Proceedings of the Sixth International Workshop on Machine Learning*, Ithaca, New York, Morgan Kaufmann.

Yang, H., Yoo, J. and D. Fisher (1989). Improving Performance by Conceptual Clustering and Concept Formation. *Technical Report 89-12*, Department of Computer Science, Vanderbilt University, Nashville, TN.