Noticing Opportunities in a Rich Environment

Matthew Brand and Lawrence Birnbaum

Northwestern University
The Institute for the Learning Sciences and
Department of Electrical Engineering and Computer Science
Evanston, Illinois

Abstract

Opportunistic planning requires a talent for noticing plans' conditions of applicability in the world. In a reasonably complex environment, there is a great proliferation of features, and their relations to useful plans are very intricate. Thus, "noticing" is a very complicated affair. To compound difficulties, the need to efficiently perceive conditions of applicability is simultaneously true for the thousands of possible plans an agent might use. We examine the implications of this problem for memory and planning behavior, and present an architecture developed to address it. Tools from signal detection theory and numerical optimization provide the model with a form of learning.

1 The Problem

An agent operating in a rich and rapidly changing environment must constantly monitor the environment as it acts, and be prepared to deal with opportunities and obstacles as they come up. The need to interact with rather than simply act upon the world presents a large variety of constraints on the computational behavior of a planner. We have tried to use these as guidelines for the design of an architecture for the pusuit of multiple goals and plans in a realtime environment.

A convenient route to the issues involved in realtime multiplanning lies in the colloquial language of opportunism. Our daily life is filled with myriad chances to satisfy or advance our goals. How we fare in the world has much to do with our ability to "notice" opportunities, "seize" them, and make use of them before the "window" of opportunity closes. Each of these idioms points to some of the many constraints that bear upon the design of a realtime multiplanner.

In particular, "noticing" means recognizing useful resources in the complexes of low-level, noisy sensory features available to the planner. Not all resources are worth attention; only those that enable plans which serve active goals. Thus, although "noticing" appears to be a perceptual process, its operation depends on decisions that must be informed by the high level goals of the agent. "Seizing" refers to the overhead of selecting a plan and executing it: choosing amongst competing plans and goals, verifying the opportunity, and assigning additional resources to the plan execution. "Window of opportunity" refers to the transience of opportunities: they have to be discovered and acted upon as suddenly as they appear and the agent may have to deal with their disappearing just as suddenly.

In short, realtime multiplanning presents three general constraints:

- The agent must incorporate an opportunity oriented planner, rather than a one-shot or agenda-oriented planners. However, being opportunity-driven does not mean that its behavior is determined bottom-up from perception: actions and indeed perceptions need to be goal-motivated. This is the *integration* constraint: The planner needs to adjudicate between bottom-up opportunities and top-down desires and expectations (see, e.g. [Birnbaum 86]).
- The agent must perform a quick and efficient mapping from surface features to applicable plans. This is the *time* constraint: The agent

must have a fast and fairly constant reaction time in any situation.

- The agent must be constantly replanning as external conditions and internal goals change. This is the *flexibility* constraint: The agent must be prepared at any time to execute, suspend, resume, or abandon a plan in the face of novel circumstances.

These constraints have guide the design decisions that form the system described below.

2 Real-World Constraints

Most compelling in the design of a realtime planner are issues of computational efficiency. The environment will tolerate only a narrow range of reaction times on the part of the agent, and the shorter the better. The agent must move from low-level perceptual features to high-level decision processes in as few steps as possible. Traditional planning tools, especially those based on search, are undesirable because they require indeterminate and exponential time to find or construct applicable plans. An animal does not stop to think how it can relate the sighting of prey to the get-nourishment goal. An efficient alternative to search is matching against rich libraries of planning information. Making this work, however, necessitates unusual commitments vis-à-vis memory, processing, and architecture.

2.1 Memory Issues

In order to be able to recognize which circumstances constitute an opportunity for the inception or resumption of a plan, that plan must already be available–and quite some detail–in the agent's memory. At the very least, it must be present in enough detail for its conditions of applicability to be quickly consulted and compared to the world. This is especially important because of the large number of potentially useful plans. The typical range of activities in an agent's "everyday" behavior will likely require hundreds or thousands of precompiled plans. We call the need to have plans easily consulted the *accessibility* constraint.

Given that we want plans to be tightly coordinated with the perceptual apparatus that cues them, it will prove useful to think of a plan coupled with the computing elements that recognize its conditions of applicability as a unit: a *behavior*. When we talk of a planner endowed with sensors and effectors, we talk of a collection of behaviors which we want to combine in ways salutary to the planner's goals. As with [Agre 88, Agre & Chapman 87, Firby 89, Maes 89] we are interested in the consequences of reinterpreting the planning task as a matter of *coordination of behaviors* rather then the *synthesis of plans*.

How specific should behaviors be? In any complex environment, behaviors do not have unique conditions of applicability; there is always some generality which cannot be captured in simple lists of features. An animal in a forest need not have one behavior for picking up edibles and another for picking up nesting materials. Having both in memory simply adds computational cost to the task of plan selection, probably more than is saved by simple feature matching. The plans in the agent's repertoire should be flexible, even though this makes recognizing each plan's conditions of applicability in the environment more expensive. There is a tradeoff between ease of opportunity recognition and generality of behaviors.

Schemes for flexible plans come with a number of extra burdens: plan synthesis requires search, plan modification requires extensive knowledge about planning, and abstract plan roles require complicated type checking. All three, however, violate the accessibility constraint. Consequently the planner's memory is strongly biased for ease of matching and against "abstract" or "universal" planning methods.

2.2 Attention Issues

Plans exist for many different time scales. Few are executed *en tous* and without interruption. Many cover spans of time so large that an agent cannot afford to execute them without interruption. For example, one cannot follow the house-building plan without breaks to entertain the eat-and-be-refreshed plan. Some plans are so long-term, such as write-thesis, or of such low priority, such as pick-up-money-from-sidewalk, that more time passes during interruptions than during execution.

Proverbially, we want an agent to be able to "walk and chew gum at the same time." This means dividing attention between many plans that are concurrently being executed, some in parallel, some in dovetailed sequence.

The idea of precompiled plans is helpful here too. If a plan may not be executed *en tous*, then conditions of applicability need be computed not only for its beginning, but at any point where it may be interrupted and later resumed. The more concrete

the description of the plan, the less costly this calculation is.

The possibility of interruption also argues for plans that are as short as possible. Rather than try to execute a full plan for building a shelter, a agent will find it easier to have a building plan broken down into its component behaviors, which it can splice together as their conditions of applicability become true in the environment. This has a number of advantages for the planner. Breaking a plan into fragments provides convenient interruption points with precomputed requirements for resumption. Plan fragments can be recombined and reordered according to the vagaries of environmental change.[1]

3 Architecture

In perceiving the world, the agent should calculate an optimal set of features for its purposes–meaning it should compute as few features as possible to identify applicable plans and choose between them. Features should only be computed when necessary *vis-à-vis* likely plans, and when necessary to many plans, a feature should only be computed once.

An efficient architecture which addresses the two priorities of shared computation of features and arbitrary mappings is a feed-forward network. Low-level features pass their information up to increasingly complex features which in turn cue and recue candidate plans. This opens the door to parallelism. However, we cannot assume that the "magic" of massive parallelization will somehow make this network tractable. Limited resources mean that a limited number of features can be computed at any given time.

Optimization in this context means making choices about which nodes in the network will receive computational resources, and which will have to remain dormant. Dormancy means that the feature retains its old value, and is increasingly likely to become incorrect as the environment changes. We wish to construct a description of how efficiently the network is using its features (and thus retrieving appropriate plans). Any network control strategy that optimizes that function will thus provide a means for intelligent attention focusing.

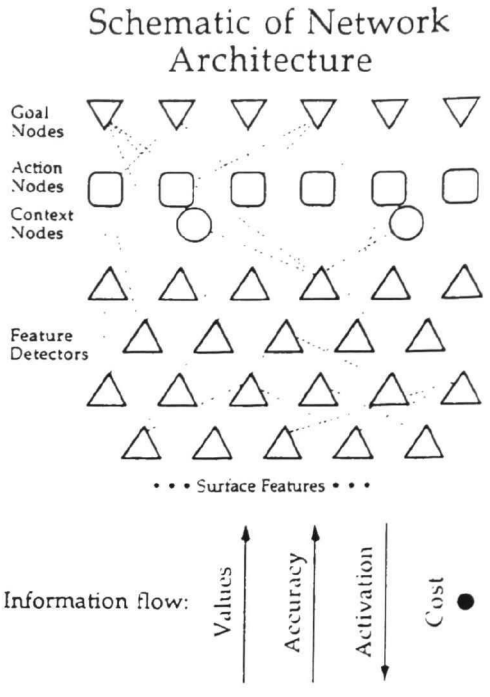

Figure 1: Connectivity and information flow in the planning network.

We have arranged the knowledge sources of our planner in a network for this purpose (see figure 1). Each node represents a kind of computation the planner can do: either computing a feature of executing a plan step. The links between nodes are pathways through which nodes interact, passing information and competing and cooperating to control the effectors and computational resources of the agent.

4 Representation

There are four basic types of knowledge in the agent: goals, action sequences, feature detectors, and memory pools. Between them pass three kinds of information: values, accuracy estimates, and activation.

4.1 Information Flow in the Planner

Values are scalars that are passed up from lower-level nodes to higher-level nodes. They indicate whether the feature computed by some node was present in the world last time the node was computed.

Accuracy estimates are scalars indicating how likely it is that a feature has been computed using

[1] This also relieves the agent of much of the memory burden involved in remembering its place in a long process. The environment is exploited as a mnemonic device, like a mechanic layings out pieces from a machine to mark his plac in the disassembly and reassembly of a machine.

obsolete information. Features can be inaccurate because subfeatures they depend upon may have been dormant while something in the world has changed. Accuracy estimates flow up the network along with values, and are reduced as they pass through each obsolete node.

Activation, also a scalar, flows down through the network over weighted connections. It is used to determine what parts of the network should be computed. Essentially, the activation at a node is an indicator of how interested the planner is in committing computational resources to it.

4.2 Knowledge Structures in the Planner

At the top of the network, goals provide the impetus behind the agent's choices. They are not predicates describing world states, as is the tradition in planning. Rather they are generators that supply activity to the action sequences which satisfy them and the features which recognize conditions in which they are satisfied.

Feature detectors are arbitrary functions which read in sensory data and/or other feature values and output a value indicating some state of affairs in the environment and the planner.

Action nodes encapsulate planning knowledge. Moving upwards, they signal their execution status to the goals that they serve, inhibiting or satisfying them. Moving downwards, they pass activation on to the feature detectors that correspond to their preconditions, priming them. From goal nodes they receive activation, and from feature detectors they receive information about their applicability. As the agent monitors the world, each action node jockeys for the right to determine the next action.

There is no basic unit of plan representation inside an action node node. Instead, these nodes hold plan fragments, single operators, and occasionally even entire canned plans.

Plans and plan fragments are expressed wholly in terms of effector instructions. There is no explicit representation of subgoals, else the planner would have to resort to search. Although this would seem to severely limit the potential sophistication of the plans the agent can express, we are take hope from two hypotheses. The first is that a large range of interesting and useful behaviors *can* be achieved under this limitation–perhaps even enough for a reasonable simulation of animal behavior.

The second is that where subgoaling really is necessary, we may expect subgoaling-like behavior in the way the agent sequences its behaviors. The reason why is that, should action node A need a precondition achieved, it will prime the feature detectors that are looking for satisfying conditions in the world. If action node B has those feature detectors as a description of what it achieves, and they are partially true, node B is likely to be executed opportunistically. Once this has happened, node A's preconditions have been satisfied, and it will be executed. Though not dependable, this scenario points a way to rudimentary subgoaling.

Memory pools are frozen contexts. Each name a plan in execution, the goal it serves, and the resources that are tied to the various roles in the plan. They point to the feature detectors that recognize their role fillers. In a carpenter's attach plan, for example, roles might be Object1, Object2, SupportingSurface, MeansOfAttachment, ClampingDevice, and Tool. Their fillers could be, respectively, a broken chair, its leg, a clear space of floor, wood glue, rope, and the right hand.

Memory pools are used to ensure that a plan can be continued. They are quite expensive, and tend to evaporate after time. This is because an agent must have a means of forgetting thwarted plans after some passage of time, in order to be reasonably free to exploit new opportunities.

5 Process Model

The steps of a typical planning cycle are:

- Choose the features to be recomputed.
- Recompute features.
- Propogate information throught the network.
- Choose an action node to control effectors.
- Execute one instruction from the node's plan fragment.
- Construction, update, or activate a memory pool to hold the current context.

The important steps in this cycle are the selection steps. These choices govern the agent's external and internal attention, and are governed by the parameters which flow through the network.

In addition to the three parameters described above–value, activation and accuracy–each node has a stationary parameter: cost. It represents the computation expense associated with that node, and is used to determine when the feature is worth recomputing. Low-level features, which are worth comput-

ing often because they are close to sensory information and are the basis for all other computation, are assigned low costs. High-level features, which are in a sense more speculative, are assigned high costs. In this way the network is biased to pay close attention to the environment.

How is the distribution of computation question decided? In the case of feature detector nodes, the feature is recomputed if

$$f(n) = \frac{activation_n - cost_n}{1 - accuracy_n} > T_F$$

where T_F is a global threshhold.[2] This equation expresses the main economy of feature computation. It simply ensures that the features are recomputed when they receive large amounts of activation or are very likely to be obsolete. A similar function $F(n)$, governs action nodes and memory nodes.

The purpose behind these parameters, besides providing the basis for interaction between knowledge sources, is to allow the planner to judge what is worth computing. The basic premise is that some kinds of computation are more expensive than others. Passing the parameters around the network and calculating $f(n)$ and $F(n)$ are cheap; we pretend that they are properties of some imaginary computational substrate. More importantly, they happen in constant time, regardless of the agent's circumstances. Actually computing the features also takes constant time, because a fixed fraction of the network is considered.

Computing the features is held to be more expensive than propagating information, and thus we limit the number that are computed each cycle. This is not just to conserve computation: if too many feature nodes are active, too many action nodes that will merit consideration. Thus as features receive inadequate activation they report increasingly obsolete information, and decay to an off-state, in which they are incapable of supporting any action node's bid for execution.

An action node is queued for evaluation if $F(n) = f(n) + K \cdot s(n) > T_A$, where K is a global coefficient,[3] $s(n)$ is a weighted vote of precondition features for that node, and T_A is another global threshhold like T_F. The node with greatest $F(n)$ is picked out of the queue, its role bindings verified, and the first unexecuted operator in it is used to control the effectors. If the role fillers fail to verify, that node can be "suspended" by construction of a memory pool, or a different node can be taken off the queue. Only a fixed number of nodes are checked, and the queue is discarded after each cycle.

Action selection is also constant in time relative to the input, and is logarithmic in the size of the network. Cycle time is thus constant, and adjustable via the two global threshholds T_F and T_A.

When the current action node is usurped by a node with greater $F(n)$, it is suspended. A memory pool is set up for it which saves the context at time of suspension. This memory pool is treated like an action node in subsequent cycles: it receives activation from the action node's sponsor goal, it is queued and evaluated like action nodes, and when it is selected the action node it is attached to resumes execution. Memory pools have a decay term in their $F(n)$ which grows increasingly negative with time, making their resumption less and less likely. When their $F(n)$ itself becomes negative, they "evaporate" and the planner loses any trace of having been in the midst of their plans.

6 Learning

Optimization is plausible in our architecture because the flow of information is restricted to specific pathways between knowledge structures, greatly simplifying credit assignment, and because information is restricted to scalars, allowing us to construct a mathematical characterization of the network's performance.

In the network, goals are served both by feature detectors, which report goal satisfaction and action nodes, which attempt goal satisfaction. Using the feature detectors as reference signals, it is easy to collect statistics on an action node's success *vis-à-vis* its sponsoring goal. The first optimization open to us is to strengthen the weights on activation links between goals nodes and the action nodes that most reliably serve them. This allows local improvements in the ability to select plans.

A more subtle and important optimization applies to the relationship between action nodes and

[2] T_F controls what fraction of the network's features are recomputed in any given cycle. A higher threshhold makes the agent like a panicked animal; it needs to respond very quickly and thus can only attend to its most immediate goals and sensations. A lower threshhold makes the agent slower, but more alert to uncommon opportunities.

[3] K allows us to balance the relative importance of need, expressed by activation, and opportunity, expressed by sensory information.

the feature nodes that detect their conditions of applicability. Here it is important to learn which features best cue an action node, and thus are most deserving of the activation that the action node can distribute. Here we use some simple tools from signal detection theory [Tanner, Swets, & Green 56, Green & Swets 66] to provide a platform from which we can do numerical optimization.

By comparing records of feature node firings with the statistics described above, we can tabulate the feature's utility (via the plan fragment) to the goal in terms of hits $p(F \cdot A)$, false alarms $p(\neg F \cdot A)$, misses $p(F \cdot \neg A)$, and correct rejections $p(\neg F \cdot \neg A)$. F refers the the presence of the feature and A refers to the decision to take an action on the basis of that feature.

From the costs and activation assigned to each node we can construct a payoff matrix. Using these and the statistics described above, we can adapt from signal detection theory the equation describing the expected value of relying upon that feature:

$$EV = W_{A\rightarrow F}p(F\ \ A) + C_A p(F \cdot \neg A) + C_F p(\neg F \cdot A) + C_G p(\neg F\ \ \neg A) \quad (1)$$

In this notation, $W_{A\rightarrow F}$ is the weight on the activation link from the action node to the feature node, C_A is the fixed cost of evaluating the action node, C_F is the fixed cost of the feature node, and C_G is a global cost assessed for missing opportunities.

By toggling $W_{A\rightarrow F}$ between two close values and collecting statistics on the relative values of EV, we discover local information about the first derivative of the actual probability curve relating feature firing to plan appropriateness.[4] For local hillclimbing, we can simply choose to remain with the value of $W_{A\rightarrow F}$ that produces the greater EV

More interesting is the case when we apply a global optimization technique. We are investigating the use of simulated annealing [Kirkpatrick, Gelatt & Vecchi 83] in the selection of weights. Under simulated annealing, one chooses one value of $W'_{A\rightarrow F}$ over another $W_{A\rightarrow F}$ with the probability

$$e^{-\frac{EV'-EV}{T}}$$

for some temperature T which determines the "volatility" of the decision. This is a particularly interesting prospect because we may be able to regulate the temperature of the annealing decisions as a function of the accuracy reported at the nodes involved.

[4] There is a strong assumption of Gaussian noise in the functioning of the network in order for this to work. We can rely on the obsolescence of some features for noise, but it is not clear that it will approximate a Gaussian.

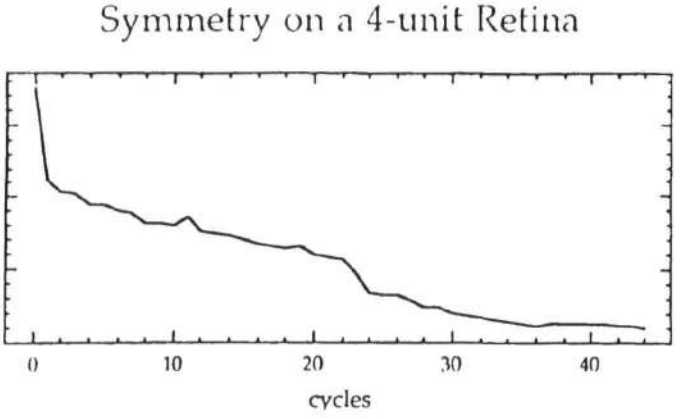

Figure 2: Error curve of a primitive variant of the optimization algorithm applied to learning symmetry in a 4-bit vector.

Simulated annealing is appropriate here because it allows us to globally optimize the interactions between "opaque" functions. By "opaque" we mean that the functions in the nodes are arbitrary, and we cannot expect to have information about their first derivatives (especially if they are symbolic!) as in backpropagation, or have the time to reason about their internal structure (since we have no search).

We have been experimenting with this optimization scheme in miniature systems, as a prelude to committing to it with a full realtime multiplanner. Our first miniature merely tested the ability of the algorithm to select connections between nodes computing miscellaneous functions in order to detect symmetry in a vector of bit-values. The network was seven layers deep, and included both numeric and logical (symbolic) functions indiscriminately connected as start. The learning curve is shown in figure 2.

We are currently working on a miniature which, given the operators toggle-bit and swap-bits, will learn how to make the vector symmetrical with a minimal number of operators.

7 Determining the Content of the Nodes

AI systems that move away from "universal" methods such as search depend heavily on well-chosen and well-organized knowledge to compensate for their

limited power. Especially crucial in our variety of planner are the repertoire of plans, how they are broken up into planning fragments for storage in action nodes, and features which serve these nodes. The planner designer must steer a course between plan fragments that are too general to be served by any reasonable set of features, and plan fragments that are so specific that each one requires its own special set of features. Ideally, we would like to have moderately general plan fragments that can be cued by relatively low-level features.

How to organize and mediate between behaviors is an open question in agent-based architectures. Architectures such as the Society of Mind [Minsky 86] and Subsumption Architectures [Brooks, Connell, & Peter Ning, 88] rely upon the idea of increasingly sophisticated layers of behavior, each higher layer handling circumstances that are too subtle or complicated for the layer below. We have avoided this layering approach for two reasons. First, the low-level process model of our architecture already mediates between competing behaviors. Second, layering clouds the prospects of an implicit ideal of these systems: that behaviors can be added or altered without having to modify all other behaviors already in the system.

In looking for an organizing principle for the construction of our behavior corpus, we are more concerned with the retrieval issues discussed above. An insight that we find useful in the selection of plan fragments is the following line of reasoning:

- Plan fragments can be classified by strategy, for example, *hoard*, *assume availability of item*, or *wait until item is more accessible*.
- Strategies in turn can be organized by resource types, for example, *perishable*, *permanent*, *core*, or *self-replicating*.
- Resource types often correlate with relatively low-level features, for example *is-food* with perishable resources, *is-moving* with time-specific resources, or *is-anchored* with location-specific resources.

Consequently, there is a useful link between relatively low-level features and highly abstract characterizations of plans. To exploit this in the design of our planner, we are collecting a set of low-level features which effectively organize a list of strategy-types we would expect a forest animal to use. These features are then used to group a corpus of plan fragments. Within each group we then determine which middle-level features would be necessary to distinguish between plan fragments, and which additional features are necessary to identify possible fillers for plan preconditions. Thus we are able to produce a full specification of the features an agent would need to compute, given an environment.

8 Problems and Acts of Faith

A realtime multiplanner is capable of acting in constant time because of the severe limitations we have placed on its means of computation. Many of these are inspired by the challenges of a real, volatile world. Others stem from the need to work around the lack of a "universal" computational mechanism. Although this planner may prove to be a good set of commitments in view of the environmental demands, it is not ideal. There are several useful properties of search-based projective planners which realtime multiplanning lacks.

8.1 Labelling is Hard

Naming is hard for an agent that doesn't have labels supplied to it by the environment or tutor (e.g. "the red block"). Naming not only presumes that the agent has a category for an object it encounters, but that it has some basis for distinguishing the object from other category exemplars it may encounter, even if this is the first such object it has ever seen. This means it has to know what is unique about the object with respect to other possible exemplars of the category. Without this ability to discriminate within categories, the agent is incapable of powers such as object permanence. The agent may often begin a task with one object, be interrupted, and continue the task with another object in the same role, oblivious to the change. This is disastrous if, for example, the agent is building its shelter, stops, and resumes in another place, completing the shelter of a rival.

One possible solution lies in more detailed descriptions of contexts in a memory pools. Presumably there are some features which are not important to an object's role in the plan, but distinguish it both from other objects in the same category and from the larger scene in which it was noticed. A fairly reliable example of this kind of distinguishing feature is location.

8.2 Planning without Protections

Most conspicuous of our planner's shortcomings is a lack of protections. The extreme parsimony in use of variables, the difficulty of labelling, and the sheer number of plans that may be simultaneously active all make checking for protection violations prohibitive. Every time the planner selects a plan it is making resource commitments: effectors in the short term and objects in the environment in the long term. When the planner suspends a plan it does not forfeit those commitments. But it *doesn't* check to see if any of those commitments are violated by subsequent plan inceptions. This would be too expensive, checking every resource against every plan. It is easy to imagine an agent picking up a tool and then promptly putting it down because it saw another, and then picking up the first again.

There are two reasons why a lack of protections may not be a stumbling block. The first is a variation on the Friendly World Assumption: in most environments the range of useful plans makes little use of protections, and where protection violations lead to trouble, the environment will soon change so that only one of the plans competing for the resource is still applicable.

The second we may call the Underspecification Assumption: many goals which seem to require protections need them only because they contain too little information in their specification. The classic example, stack block A on block B and block B on block C, would be beyond our agent. The stack two blocks plan fragment would put A back on B every time the agent figured to clear B to put it on C . However, a plausible real-world alternative, stack-the-blocks-in-size-order, is easily handled by repeated executions of the put-the-biggest-free-block-on-the-pile plan fragment.

9 Acknowledgements

This work was supported in part by the Defense Advanced Research Projects Agency and monitored by the Air Force Office for Scientific Research under contract number F49620-88-C-0058, and in part by a National Science Foundation Graduate Fellowship. The Institute for the Learning Sciences was established in 1989 with the support of Andersen Consulting, part of The Arthur Andersen Worldwide Organization.

References

[Agre 88] P. Agre. *The Dynamic Structure of Everyday Life*. PhD Thesis, MIT Department of Electrical Engineering and Computer Science, 1988. Available as Report AI-TR 1085.

[Agre & Chapman 87] Philip E. Agre and David Chapman. Pengi: An Implementation of a Theory of Activity. In *Proceedings of AAAI-87*, 1987.

[Birnbaum 86] L. Birnbaum *Integrated Processing in Planning and Understanding*. PhD Thesis, Yale University Computer Science Department, 1986. Available as Report RR-489.

[Brooks, Connell, & Peter Ning, 88] Rodney Brooks, Jonathon Connell, and Peter Ning, Herbert. A second generation mobile robot, AI Memo 1016, MIT Artificial Intelligence Laboratory, 1988.

[Firby 89] R. James Firby. *Adaptive Execution in Compplex Dynamic Worlds*. PhD Thesis, Yale University Computer Science Department, 1989. Available as Report RR-672.

[Green & Swets 66] David M. Green and John A. Swets. *Signal detection theory and psychophysics*. New York: J. Wiley & Sons. 1966.

[Kirkpatrick, Gelatt & Vecchi 83] Scott Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science* 220:671-680. 1983.

[Maes 89] Pattie Maes. The Dynamics of Action Selection. In *Proceedings, IJCAI-89*. 1989.

[Minsky 86] Marvin Minsky. *The Society of Mind*, New York: Simon and Schuster, 1986.

[Tanner, Swets, & Green 56] W. P. Tanner, Jr., J. A. Swets, and D. M. Green. Some general properties of the hearing mechanism. University of Michigan: Electronic Defense Group, 1956. Technical Report No. 30.