Why Knowledge Systems Research Is In Trouble and What We Can Do About It

Paul R. Cohen

COINS Technical Report 89-81

Experimental Knowledge Systems Laboratory
Department of Computer and Information Science
University of Massachusetts
Amherst, Massachusetts 01003

Abstract

This paper is about the current state of knowledge systems research and how I think it can be improved. It is a personal view, but not an uncommon one. Indeed, one purpose of the paper is to clarify what I perceive to be general dissatisfaction. The second purpose is to introduce some work that I believe addresses some of these concerns.

[0]This research was sponsored by the Office of Naval Research, under a University Research Initiative grant, ONR N00014-86-K-0764 and the Office of Naval Research, contract no. N00014-88-K-0009.

Contents

1 Introduction

I was recently reminded of Allen Newell's paper *You Can't Play 20 Questions With Nature and Win.* Newell wrote it in his role as discussant at a Carnegie Symposium, and addressed it to the cognitive psychology community. The following paragraph appears early in his comments:

> I was going to draw a line on the blackboard and, picking one of the speakers of the day at random, note on the line when he got his PhD and the current time (in mid-career). Then, taking his total production of papers like those in the present symposium, I was going to compute a rate of productivity of such excellent work. Moving, finally, to the date of my chosen target's retirement, I was going to compute the total future addition of such papers to the (putative) end of this man's scientific career. Then I was going to pose, in my role as a discussant, a question: Suppose you had all these additional papers ... *where will psychology then be?* Will we have achieved a science of man adequate in power and commensurate with his complexity. And if so, how will this have happened via these papers I have just granted you? Or will we be asking for yet another quota of papers in the next dollop of time.
>
> – Allen Newell, You Can't Play 20 Questions With Nature and Win. In W.G. Chase (Ed.) *Visual Information Processing.*

When I read Newell's paper for the first time, I was an interdisciplinary graduate student in psychology and AI. The attitude toward psychology in the Stanford Heuristic Programming Project convinced me that a truly interdisciplinary thesis wasn't possible, and Newell's paper was an important reason that I chose to work in AI and not psychology. I have always been strongly influenced by the questions Newell asks in the quote above, and lately I have started to think they apply to AI as well; and if not to AI in general, certainly to expert systems. Add up all the papers on expert systems in IJCAI, AAAI, and AI Magazine over the last five years, and what have we got? What can we expect in 1995? When Newell asks whether psychology will ever add up to a "science of man," he assumes a common purpose among psychologists. Whether or not this is realistic in psychology, it certainly doesn't characterize expert systems research. I can't see any common scientific purpose there, which is one reason I think expert systems research isn't adding up to a science of anything. Newell asks whether individual papers, each excellent and methologically sound, add up to a science. I don't think we have more than a handful of excellent and methodologically sound papers, period. These two issues—scientific purpose and methodology—are at the heart of my dissatisfaction with expert systems.

The following two subsections are about these issues. Let me preface them by characterizing my biases. First, when I talk about research in expert systems or knowledge systems, I am referring to a range of component technologies, including knowledge representation, learning, ontology, case-based reasoning, generic tasks, and control. It is easy to criticize hack work in expert systems, and my comments can certainly be interpreted this way (i.e., as criticisms of the other guys), but I direct them primarily to my colleagues in the research community. Second, I view expert systems as a vehicle for Artificial Intelligence research, not as an engineering subdiscipline. Irrespective of the value of expert systems in the marketplace, they are worthless to me unless they tell me something about intelligence. Third, I think of intelligence primarily in terms of behavior—what an agent thinks or does—and secondarily in terms of structures—what the agent knows, or how it is represented. With these biases, which I think are pretty common, let me now say why expert systems research has become intellectually unsatisfying.

1.1 Learning about Intelligence

I am not learning much about intelligence, artificial or otherwise, from the knowledge systems literature. One reason is that we have come to equate intelligence with knowledge, which is static, instead of with dynamic behavior. This is seen most clearly in the disregard for control in the knowledge systems community, although the consequences aren't felt until someone tries to acquire and reason with knowledge about how experts (and expert systems) should behave. For example, when Clancey tried to represent diagnostic strategies for the purpose of tutoring, he discovered (as we did later in our MUM project [7]) that the only ways to specify the behavior of knowledge systems were brutish chaining, meta-rules, bizarre hacks, or lisp programming [4]. If you are looking for tools to describe behavior, the knowledge system literature offers little besides reassurance that simple control strategies are sufficient. Sufficient for what? Certainly not for modelling the complex behavior that Barbara Hayes-Roth observed in human errand planning [12]; or that Ed Durfee requires for simple cooperative, distributed problem solving [10]; or for the simple meta-planning we require to prune the search space in mechanical design [19]; or for real-time planning against simulated forest fires (see below).

Research on reasoning under uncertainty is another, personally frustrating, manifestation of the emphasis on knowledge and structure over behavior and dynamics. It offers more and more refined ways to calculate degrees of belief, but relatively little work on how problem solvers plan, decide, and act in uncertain environments [6,5]. Once again, intelligence is equated with what you know (more precisely, with maintaining degrees of belief in propositions), not with how you behave.

The uncertainty community evidently feels it is subcontracting to the knowledge systems community. They are the folks who provide the calculus. Judging by the preambles of papers I have recently reviewed, a lot of people think this way. "Knowledge systems need my innovation in ...." You can fill in the dots with uncertainty calculus, explanation mechanism, learning mechanism, reason maintenance system, and so on. The knowledge systems literature, and AI in general, is busily producing component technologies. But it is difficult to learn much about intelligence from a bunch of component technologies. They don't add up to a theory of anything, any more than the unremitting tide of psychology papers add up to "a science of man." (There are also mundane, pragmatic reasons to worry about the component technology approach to knowledge systems, as I'll discuss later.)

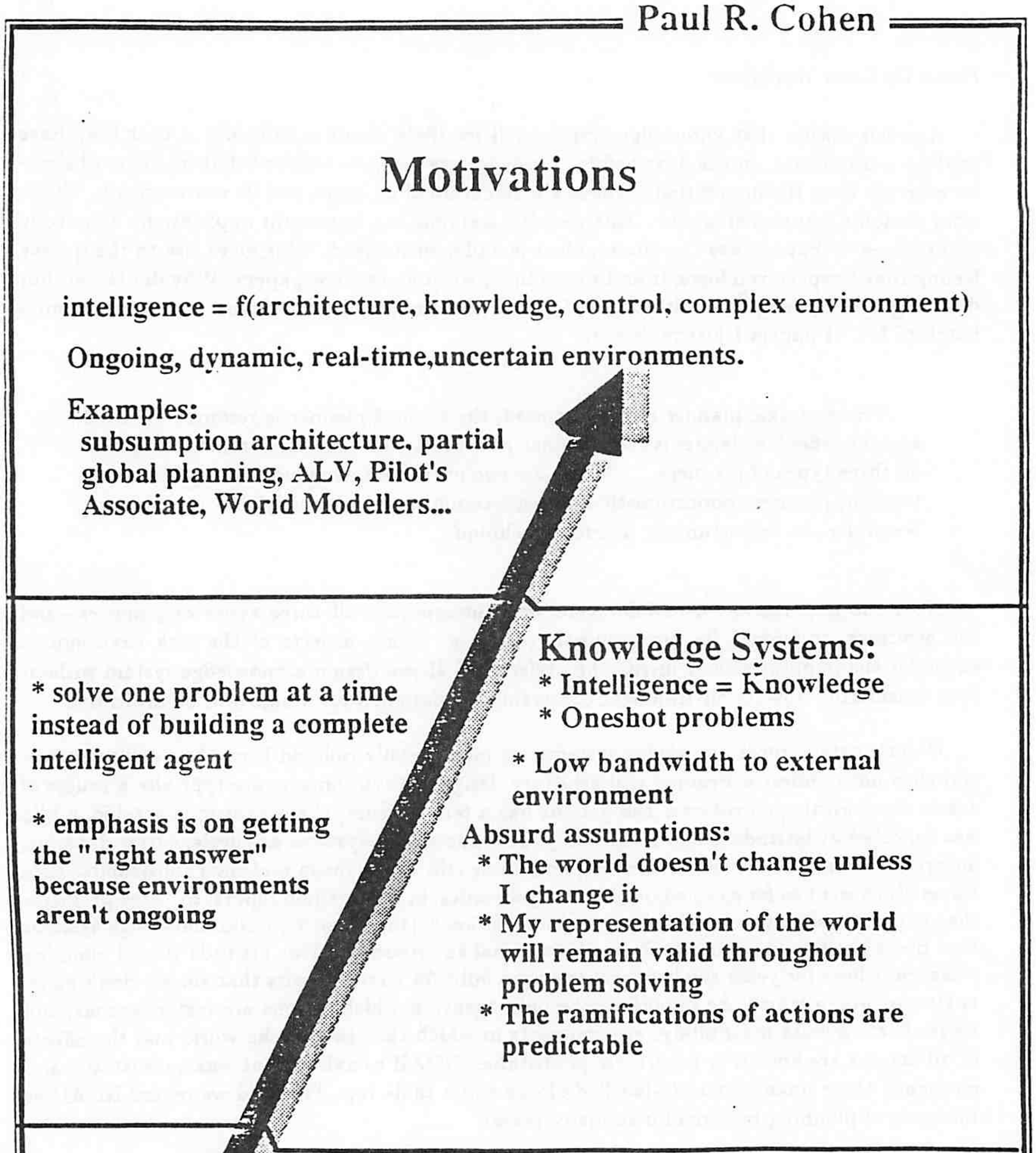
Another reason that knowledge systems tell me little about intelligence is that they have trivial environments. In the days before knowledge systems, we believed that intelligent behavior emerges from the interaction between the structure of an agent and its environment. That's what Simon's Ant was all about. This view has a simple but important implication: Any study of intelligence that ignores the environment is underconstrained. This gives rise to the queasy feeling that I expect you have, from time to time, when you review papers: Why did the author do things this apparently arbitrary way? Here's an example, drawn almost at random, from a batch of IJCAI papers I just reviewed:

> The strategic planner is goal-oriented, the tactical planner is resource oriented, and the reflexive planner is event (signal-) oriented. A mobile agent must incorporate all three types of planners. ...There are two principal techniques to implement [the tactical] planner: opportunistic and least-commitment planning. In our approach, we prefer the opportunistic planning technique.

Note the imperative—a mobile agent *must* incorporate all three types of planners—and the arbitrary preference for opportunistic planning. What aspects of the task environment engender the imperative and justify the preference? If you design a knowledge system without first considering how its environment constrains the design, your design will be arbitrary.

Unfortunately, most knowledge systems are intentionally isolated from their environments, and thus have underconstrained and arbitrary designs. Environments are typically a couple of dozen categorical propositions: the patient has a temperature, the organism is aerobic, a blip was reported at latitude x and longitude y. Few knowledge systems are designed for dynamic, uncertain, or multi-actor environments; and fewer still can manage real-time constraints. Rick Hayes-Roth went so far as to advise British companies, in a Pergamon report, to "Seek problems that experts can solve via telephone communication." [13]. That is, build knowledge systems that don't need to know much about the external environment. This attitude ruined planning research, where for years the best systems were built for environments that simply don't exist: environments in which the planner is the only agent, in which actions are instantaneous, and their effects persist indefinitely; environments in which the state of the world and the effects of all actions are known or accurately predictable. NOAH's environment was quasi-static, and contained three unambiguously-labelled blocks and a table top. Why did we regard NOAH as the apex of planning research for so many years?

Many of these arguments are summarized in Figure 1, which is a crude history of how AI has characterized intelligence. Initially, intelligence was viewed as the behavior that emerged from the interaction between autonomous agents and their environments. But knowledge systems later became isolated from their environments, behavior was de-emphasized, and research on complete, autonomous agents was replaced by work on component technologies. Answers mattered; the process of deriving them did not. The majority of tasks were "one-shot," meaning that we would solve the problem now, once and for all, and not monitor or revise the solution



Motivations

intelligence = f(architecture, knowledge, control, complex environment)

Ongoing, dynamic, real-time,uncertain environments.

Examples:
subsumption architecture, partial global planning, ALV, Pilot's Associate, World Modellers...

* solve one problem at a time instead of building a complete intelligent agent

* emphasis is on getting the "right answer" because environments aren't ongoing

Knowledge Systems:

* Intelligence = Knowledge
* Oneshot problems

* Low bandwidth to external environment

Absurd assumptions:
* The world doesn't change unless I change it
* My representation of the world will remain valid throughout problem solving
* The ramifications of actions are predictable

Simon's Ant:

Intelligence = f (simple structure, complex environment)
Human Problem Solving, GPS

FIGURE 1

in future. One-shot problems denied us the opportunity to study behavior in ongoing environments; for example, instead of building expert systems to "wait and see" how a patient's condition develops, we built systems to give the best possible recommendation *now*, even if the data were poor.[1] Again, this is seen clearly in the uncertainty literature, where virtually all the research concerns what to do with the evidence you already have, and none concerns how to get evidence, corroborate it, hedge against future outcomes, or other strategies for coping with ongoing, uncertain environments.

More recently, there has been a renaissance of old ideas about intelligence. A few projects are beginning to acknowledge the role of the environment. The planning literature has been reinvigorated by ideas about situated action [11]. Major DARPA-sponsored efforts, such as Pilot's Associate and the Autonomous Land Vehicle, are forcing us to contend with dynamism, real time, and multiple actors. But in the following section, I will ask whether we have the methodology to capitalize on this positive change in emphasis, or whether we will end up doing the same kind of inconclusive, "look ma, no hands" research in yet another set of task domains.

1.2 Methodology

One methodological problem for knowledge systems research is that the ratio of science to engineering is too low. Back in the old days, it took many years to build an expert system, but one at least had the chance of discovering something. Today, knowledge systems are bigger, and still take years to build, but they are rarely built to discover anything. One has to work very hard to get a system that does ... pretty much what one expects it to do. One can't dismiss this as applying only to hack applications; with few exceptions, none of us are discovering enough to warrant the engineering effort of our projects.

Knowledge systems are built as demonstrations, not as experiments. Researchers rarely say what they intend to learn by doing their work, or what they actually learned by doing it. More often, proposals and papers assert that "We need X, and here's how we expect to provide X, and (later), here's a demonstration of X." Lenat and Feigenbaum put it this way:

> If one builds programs that cannot possibly surprise him/her, then one is using the computer either (a) as an engineering workhorse, or (b) as a fancy sort of word processor (to help articulate one's hypothesis), or, at worst, (c) as a (self-) deceptive device masquerading as an experiment. [15]

Their alternative, the empirical inquiry hypothesis, says what we should be doing although not how to do it:

[1]You might object that sometimes we can't afford to wait and see. I agree that we need to build systems that reason about what they afford (e.g., following Lesser's idea of approximate processing [17]). This is precisely the kind of reasoning that has been absent from knowledge systems until recently.

> The most profitable way to investigate AI is to embody our hypotheses in programs, and gather data by running the programs. ...Progress depends on the experiments being able to falsify our hypotheses; i.e., these programs must be capable of behavior not expected by the experimenter.

What hypotheses? The general form of a hypothesis in knowledge systems research is "X is sufficient to produce Y"; for example, a rule-based representation of expert knowledge and a backward chaining interpreter are sufficient to produce therapy recommendations at an expert level. Given the emphasis on component technology, mentioned earlier, most hypotheses are more specific; for example, the Dempster-Shafer method is sufficient to combine evidence in MYCIN. But unlike hypothetico-deductive science, we never show the necessity of one hypothesis by *rejecting* a mutually exclusive one. Our principle mode is to accept the null hypothesis, to accrue demonstrations of sufficiency.

It doesn't have to be this way, and the few counterexamples suggest our science would be more productive if we tried more often to show that mechanisms *don't* work. I think the best results of Lenat's work with EURISKO and AM emerged from his failed attempts to apply AM's techniques to heuristics themselves. The failure, and the ensuing enquiry, led to this remarkable conclusion:

> It was only because of the intimate relationship between Lisp and Mathematics that the mutation operators (loop unwiding, recursion elimination, composition, argument elimination, function substitution, etc.) turned out to yield a high "hit rate" of viable, useful new math concepts when applied to previously-known, useful math concepts—concepts represented as Lisp functions. But no such deep relationship existed between Lisp and Heuretics, and when the basic automatic programming (mutations) operators were applied to viable, useful heuristics, they almost always produced useless (often worse than useless) new heuristic rules. [16]

This is one of the few strong results of knowledge systems research, one of the few papers I would cite if asked what we have learned by building all these systems.

There are really two, interrelated methodological issues here. One is how we select research problems, the other is what we do with them. We are very good at selecting new research problems because we are very poor at studying them. Instead of trying to find out why something works or doesn't work, as Lenat and Brown did, we are content to show merely that something works. Once we have demonstrated sufficiency, we move on to another problem. I call this the strip mining heuristic: Once you have grabbed the gold near the surface, move on. The gold nearest the surface is by convention demonstrations of sufficiency. Knowledge systems research trashes the space of questions about intelligence in much the way that slash-and-burn cultures trash the rain forest. Both make very inefficient use of resources and impress upon me a horror of waste. Even when the resources are used well and we get all we can out of each project, as in Buchanan and Shortliffe's superb collection of papers on MYCIN [3], most of our work is still demonstrations of sufficiency.

With Adele Howe, I have started to outline methods for getting results out of knowledge systems [8,9], and I know that other areas of AI are engaged in similar efforts [14]. All this work is pretty preliminary, and it needs to be done whether one works with knowledge systems or in some other area of AI. I still spend a lot of time wondering how to do research instead of doing it. But this is preferable to messing around with expert systems as we have in the past.

2 What Now?

In the previous section I raised two issues, scientific purpose and methodology, that cause me (and I believe others) to be dissatisfied with knowledge systems research. In this section I describe a problem that, I believe, focusses my research on the right scientific issues, and the methodology that I think is appropriate to study it.

We have built a large simulation of forest fires and the equipment commonly used to put them out. We are building a planner called PHOENIX that operates in this dynamic, real-time world. The planner's goal is to manage the fire—limit the loss of human life, limit the damage to forest and buildings, and limit the monetary costs of achieving these goals. Our simulation consists of a large geographical area ("Explorer National Park") in which there is a considerable variety of topography and ground cover, as well as roads, lakes, and streams. These features affect how forest fires burn. Equally important features are wind speed and direction, both of which can change unpredictably; and the moisture content of the ground cover, which varies in time and geographically. To fight the fire, the simulation provides bulldozers, crews, transport vehicles, planes and helicopters. These cut fire line, move firefighters, spray water, or dump retardent.

Originally, these fire-fighting agents were directed by a human player in what was essentially a complex, real-time video game. We gained considerable insight into (and respect for) the dynamics of this mini-world by playing against the simulation—often losing many lives and considerable real estate to a seemingly slow and containable fire. It is difficult for a planner, human or AI program, to do very well at the game (i.e., put out the fire with reasonable costs, no loss of life, etc.) because:

- The simulation is real-time with respect to the fire. While fire-fighting agents move, cut line and drop retardent, the fire keeps burning. Any time the planner devotes to deliberation is claimed as real estate by the fire.
- The player's knowledge of the fire is limited to what the agents in the field can "see." Crews and bulldozers can see only short distances; watchtowers and aircraft can see further. The planner rarely, if ever, has complete knowledge of the extent or location of

the fire. This is evident in Figure 2, which shows in the right panel the world as it "really is," and in the left panel the world as seen by a watchtower—the spindly icon a little north of the center of the panel. (The figure also shows roads, houses, and a lake in the southeast. This is a small fraction of Explorer National Park. Normally, the display is in color.)

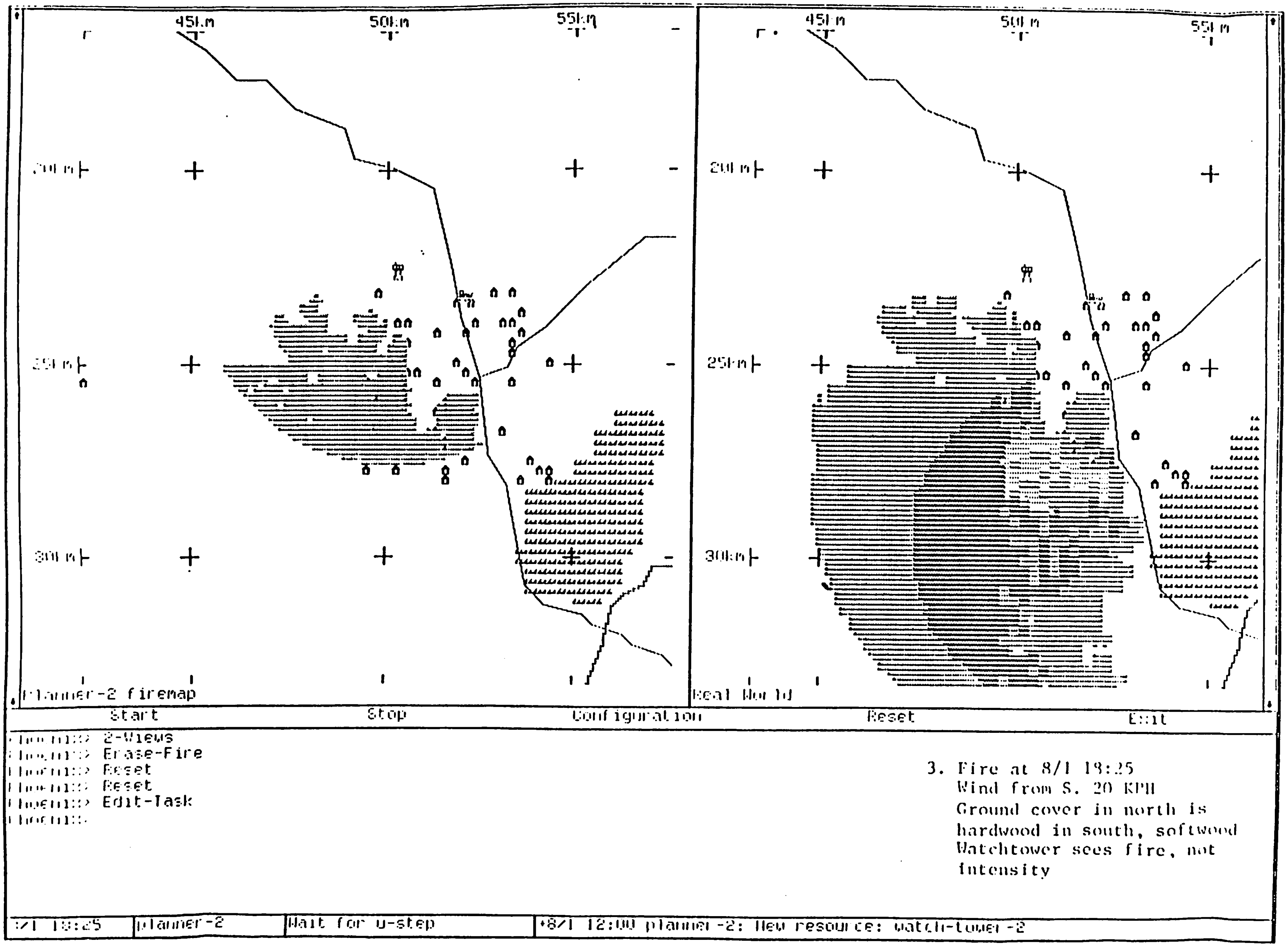
- The behavior of the fire cannot be accurately predicted because some factors that affect it, terrain, ground cover and the moisture content of the ground cover, are known only approximately. Moreover, wind speed and direction can change unpredictably.
- The behavior of the fire-fighting agents cannot be accurately predicted. In particular, the time required to move to a location or perform some task depends on terrain and ground cover. Fire-fighting agents also have varying degrees of autonomy, so the central planner cannot always be sure of their location.

These are some of the specifics of the fire environment, and the difficult technical problems they raise for the PHOENIX planner. More generally, they exemplify the kind of environment we should be studying in knowledge systems research if we want to learn about intelligent behavior. Two salient characteristics of the fire environment are uncertainty and real-time dynamics, described above. Others are that the environment is *ongoing* (as opposed to one-shot), so the emphasis is on controlling a process through one's behavior, as opposed to solving a problem through inference. The environment supports *multiple agencies*, such as wind, rain, and the fire itself; and it supports multiple actors which need to be coordinated to get a global view of the situation and to control it. The environment also has several measures of success (or failure). Most importantly, it doesn't have just one "right answer," but requires a planner to evaluate tradeoffs between plans, even during execution.

Given this focus, how should we proceed? We have three goals, related to the problems we discussed earlier:

- Work in complex environments, in which behavior matters.
- Justify design decisions by reference to aspects of the environment, instead of accepting the first sufficient design.
- Design and build complete, autonomous agents, and de-emphasize component technologies.

I am hedging my bets with respect to these goals by working simultaneously within two methodological frameworks. The first, which guides the development of PHOENIX, is a ***top down*** design effort in which we identify the abilities that we believe a planner will need to excel in the fire domain. We designed the domain itself to ensure that providing these abilities would solve open technical problems in AI (specifically, problems in real-time planning and distributed AI). As predicted by the empirical inquiry hypothesis, we are discovering unexpected behaviors. One



```
Phoenix::> 2-Views
Phoenix::> Erase-Fire
Phoenix::> Reset
Phoenix::> Reset
Phoenix::> Edit-Task
Phoenix::>
```

3. Fire at 8/1 18:25
   Wind from S. 20 KPH
   Ground cover in north is
   hardwood in south, softwood
   Watchtower sees fire, not
   intensity

```
8/1 18:25 | planner-2 | Wait for u-step | *8/1 12:00 planner-2: New resource: watch-tower-2
```

FIGURE 2

surprise is that purely reactive behavior is sufficient to put out some fires. But to follow through on the goals, above, we need to explain this result by reference to aspects of the environment. For example, one reason that reactive planning works is that the fires are typically convex, so it is rare for reactive bulldozers to get trapped in "pockets" of fire (but see Figure 3) for a counterexample.) Note also that we can't explain getting trapped (or avoiding it) by reference to any single component of the bulldozer's architecture. Getting trapped is a function of the bulldozer's radius of view, the frequency with which it updates its view, and its set of reactions. We can't explain performance in terms of a single component, nor can we improve performance by developing component technologies in isolation.

I call PHOENIX a top-down design effort because it is driven by a longish list of design goals. Eventually, PHOENIX will handle multiple fires, and coordinate multiple fire-fighting objects. It will monitor the progress of plans in real-time, and modify plans to give the best performance for the available resources. Where do these goals come from? One source is the judgment that the PHOENIX planner will need these skills to put out fires; the other is the recognition that these are open technical problems in AI. Now, ordinarily, a researcher is congratulated for finding a task, like fire-fighting, that requires methods which AI hasn't yet developed. Fire-fighting is a good task because, apparently, we will need to advance the state of the art to do it right. But in the context of my previous comments, I think we should be a bit suspicious. Are these design goals really mandated by the environment? I believe that these skills will enable a planner to put out fires (i.e., will enable a demonstration of sufficiency), but are they *necessary*? Do we need to modify plans during execution, or is this just a bit of technical showmanship? The trouble with top-down research is that this question is often very difficult to answer. For this reason, I have recently started another project with the same methodological goals—to work in complex environments, eschew component technology, and justify design decisions in terms of structure and dynamics of the environment—but with a different methodology.

We can approach intelligence from the bottom up by starting with simple structures, extending them only to provide adaptability in environments, but making the environments increasingly complex, and the required behaviors increasingly sophisticated. Bottom-up work is rare in AI, but is typified by Brooks' approach to robotics [2] and Braitenberg's Vehicles [1]. Bottom-up research is applied in the sense that all our technology must be immediately useful, that is, must have "adaptive significance" to the automata for which it is developed. We don't introduce technology for its own sake, but only to make our automata more capable in increasingly complex environments. Progress is driven by primarily by environments.

Perhaps the most important aspect of bottom-up research is that it uncovers *unintended, emergent* behavior. By emergent I mean behavior that is due to interactions between the agent and its environment, or within the agent. Our recent work suggests that unintended, emergent behavior is common in agents that interact with even simple environments over time. Moreover, this behavior has assymetric consequences for the bottom-up and top-down research strategies. For bottom-up research, every emergent behavior is an opportunity to expand the repertoire of behaviors; that is, many emergent behaviors are serendipitous. For top-down research, unintended, emergent behavior is rarely serendipitous. It usually messes up the design. This

G-(7f?

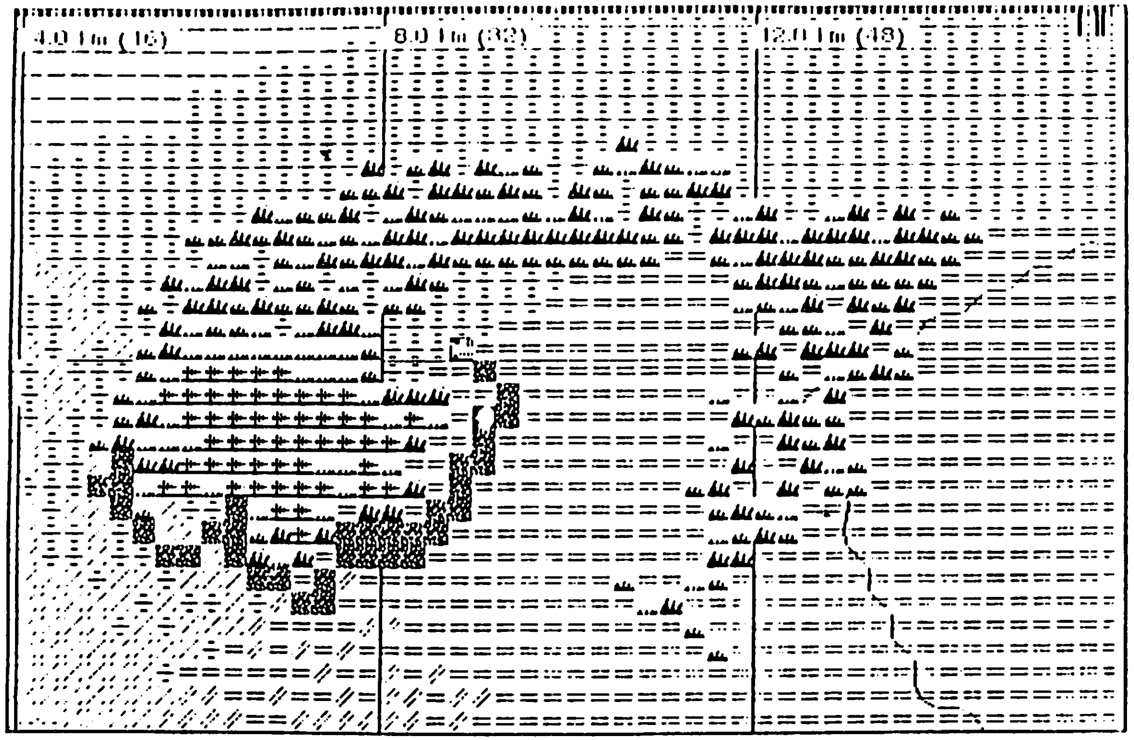


FIGURE 3

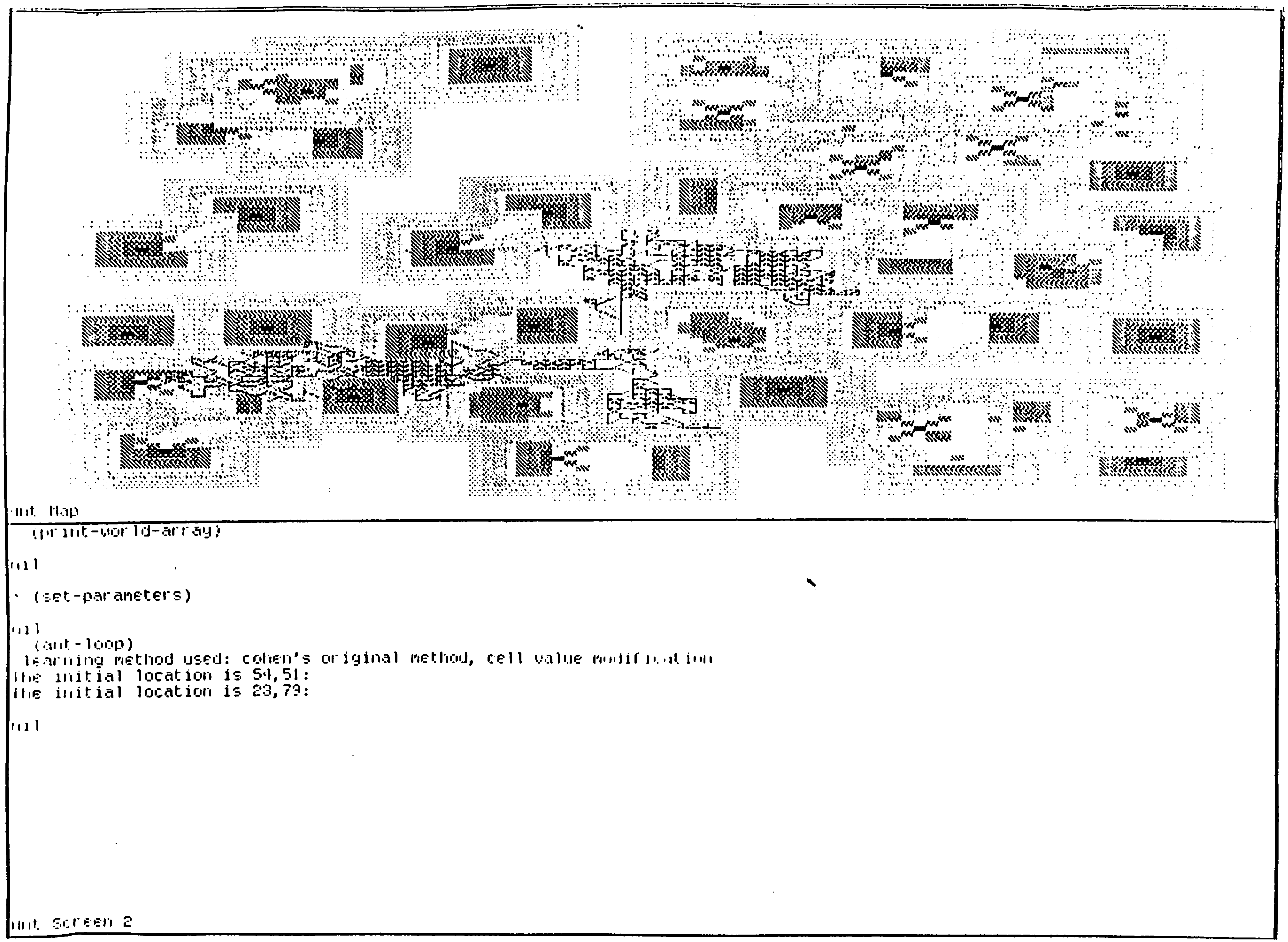
suggests that if we are trying to design agents to interact with ongoing, dynamic environments, it may be more efficient to design them bottom-up than top-down. For example, imagine I wanted an automaton to learn how to find its way downhill from any point in a landscape to the lowest accessible plateau, and to stay on the plateau thereafter. A top-down approach might involve three components—one to get the automaton onto a plateau (presumably using what it had previously learned), one to learn from the current problem, and one more to keep the automaton on the plateau once it gets there. In fact, a recent bottom-up approach to this problem found that the third component is unnecessary: in learning how to go downhill, the automaton learns not to go uphill, and so by the time it reaches the plateau it already knows enough to avoid moving off the plateau.

The bottom-up project, which we call PM, is in its early stages. Eventually, we hope to have autonomous agents capable of putting out fires in Explorer National Park, but the methodology calls for approaching this goal incrementally, so now we are working with simple automata in simple environments. For example, Figure 4 shows a "beach" over which a version of Simon's ant perambulates.

Recall that one of my three methodological goals is to build complete automata. We don't really know the minimum set of skills an automaton needs, but I settled on these four as a first cut:

1. Automata should perceive their environments. Roughly, this means at least that they construct internal representations of some or all of their environments.
2. Automata have internal state. In some cases, internal state will be no more than the internal representation of the environment. In others, it will include "forces" like "hunger." In more sophisticated automata, internal state might result from processing information beyond perceptual processing. Internal state (including perceptions) determines how automata behave.
3. Automata act. This may be definitional, since the state of an object that doesn't act is determined exclusively by its environment, so that object isn't autonomous.
4. Automata learn.

Since the inception of the project, we have designed a dozen automata for the environment in Figure 4, and for related environments that present more difficult learning and performance problems. We have observed automata getting stuck in corners, getting trapped on plateaus, exhibiting "superstitious" behavior on downhill trajectories, decreasing their rate of learning, and cycling and other repetitive behaviors. These were all unintended behaviors. There was no way to predict them by looking at the automaton's code, nor were they intended by the programmer. They are emergent behaviors in this sense: None can be explained by a single aspect of the automaton's design. Take susperstitious behavior as an example. When an automaton moves downhill, it remembers the context in which it started the move, and the



```
  (print-world-array)

nil

· (set-parameters)

nil
  (ant-loop)
 learning method used: cohen's original method, cell value modification
the initial location is 54,51:
the initial location is 23,79:

nil
```

FIGURE 4

move itself, and increases the score of that move in that context. Furthermore, it always selects the move with the highest score. This means that the first positively-scored move will have a higher score than any other, and will always be selected in that context. Moreover, because of the way the environment is constructed, and because of the way contexts are constructed to access moves in memory, a move that led downhill in a context will typically lead downhill the next time the context is encountered, so the move will typically have its score increased each time it is repeated. Note that to explain a single observed behavior, I have had to discuss the agent's learning mechanism, the structure of the environment, how memories are accessed, and how moves are selected. For brevity, I left out the influence of the perceptual system, but it too plays a role in the emergent behavior.

These behaviors aren't necessarily desirable, but after years of AI programs that do *exactly* what's expected, any surprises are refreshing. More to the point, the desirability of emergent behaviors depends on the environment; superstitious behavior on downhill trajectories is only a problem if the environment demands that automata find the *fastest possible* path down a hill.

For each automaton, in each environment, we ask some or all of these questions:

1. What is the minimum structure necessary to achieve a level of adaptation for an automaton in an environment?
2. How robust is the automaton to ranges of environmental conditions?
3. Could the amount of learning necessary to achieve adaptation be reduced by making another evolutionary step? What other aspects of the interaction between environment and automaton seem to require a more sophisticated automaton?
4. As much as possible (given emergent behavior, interactions, and nondeterminism) explain *why* the agent behaves as it does in particular environments. This is especially important for emergent (unexpected) behaviors. Why does the ant describe a sawtooth? Why does it get stuck in corners? Are these behaviors adaptive, given one's definition of adaptive?
5. Are there parallels between the design of one's automaton and biological systems? There needn't be, and we'd be fools to reject designs because they couldn't occur in organic systems, but if the parallels are there, I'd like to know about them.

To date, the PM project has not achieved any major results, but we have discovered some minor ones. For example, an automaton with a relatively poor ability to discriminate contexts can actually outperform automata with greater acuity. This is because an inability to distinguish contexts is de facto generalization over contexts, so automata that can't distinguish specific situations in effect learn classes. A simple example is the "beach" in Figure 4. All our automata learn which of eight neighboring cells to visit from any given cell, and all construct a context for a move from the altitude values of the eight neighboring cells. But the original automata discriminated contexts by the actual altitude values, while later automata simply asked whether neighboring cells were "up" or "down" from the current cell. The latter case is de facto generalization over contexts. Moreover, it is a good generalization, because the automata are punished or rewarded not for moving to particular altitudes, but for moving up or down. Now, in the long run, the automaton that can discriminate more contexts will outperform the one that discriminates fewer; but it takes a lot longer to learn all the detailed contexts. You can have performance at one level quickly, or you can wait longer and get higher performance. The choice is determined by the environment, as all design decisions should be.

3 Conclusion

I started this paper with Newell's provocative questions to the psychology community, and I want to end it by reviewing his recommendations. I found it remarkable that his questions were so pertinent to AI, and equally remarkable that his advice to the psychology community is so pertinent to us. Here is what Newell recommends:

> The first recommendation is to construct complete processing models, instead of partial ones as we do now. ...The second ...is to accept a single complex task and do all of it. [18]

Newell's paper was written 15 years ago, when psychology seemed stuck and AI seemed to offer an alternative. Today, knowledge systems research seems stuck because we ***haven't*** built complete processing models, but have focussed on component technologies; and because we haven't accepted complex tasks, but have stayed within trivial environments. If we are to fare better than cognitive psychology, it's time to follow Newell's advice.

References

[1] Valentino Braitenberg. *Vehicles: Experiments in Synthetic Psychology.* MIT Press, 1984.

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

[3] B. G. Buchanan and E. H. Shortliffe. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Addison-Wesley, Reading, MA, 1984.

[4] William J. Clancey. From guidon to neomycin and heracles in twenty short lessons. *AI Magazine,* 7(3):40–60, 1986.

[5] Paul R. Cohen. The control of reasoning under uncertainty: A discussion of some programs. *The Knowledge Engineering Review,* 2(1), March 1987.

[6] Paul R. Cohen and David S. Day. The centrality of autonomous agents in theories of action under uncertainty. Eksl technical report, University of Massachusetts, January 1988. To appear in the *International Journal for Approximate Reasoning.*

[7] Paul R. Cohen, David S. Day, Jeff Delisio, Michael Greenberg, Rick Kjeldsen, Dan Suthers, and Paul Berman. Management of uncertainty in medicine. *International Journal of Approximate Reasoning,* 1(1):103–116, 1987.

[8] Paul R. Cohen and Adele E. Howe. How evaluation guides AI research. *AI Magazine,* 9(4):35–43, 1988.

[9] Paul R. Cohen and Adele E. Howe. Is there a method to our madness?: Case studies in evaluation. Eksl technical report, University of Massachusetts, 1988. To appear in *IEEE Systems, Man, and Cybernetics.*

[10] E. H. Durfee and V. R. Lesser. Incremental planning to control a time-constrained, blackboard-based planner. *IEEE Transactions on Aerospace and Electronic Systems,* 1987. To appear.

[11] Michael P. Georgeff and Amy L. Lansky, editors. *The 1986 Workshop on Reasoning about Actions and Plans,* Timberline, Oregon, 1987.

[12] Barbara Hayes-Roth and Frederick Hayes-Roth. A cognitive model of planning. *Cognitive Science,* 3:275–310, 1979.

[13] F. Hayes-Roth. Knowledge-based expert systems–the state of the art in the us. *Pergamon Infotech State of the Art Report,* 1984.

[14] Pat Langley. Research papers in machine learning. *Machine Learning,* 2(3):195–198, November 1987.

[15] D. B. Lenat and E. Feigenbaum. On the thresholds of knowledge. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 1173–1182, Milan, Italy, 1987.

[16] Douglas B. Lenat and John Seeley Brown. Why am and eurisko appear to work. In *Proceedings of the AAAI-83*, pages 236–240, 1983.

[17] Victor R. Lesser, Edmund H. Durfee, and Jasmina Pavlin. Approximate processing in real-time problem solving. *AI Magazine*, pages 49–61, Spring 1988.

[18] A. Newell. You can't play 20 questions with nature and win. In W. G. Chase, editor, *Visual Information Processing*, pages 283–308. Academic Press, NY, 1973.

[19] Mark E. Orelup, John R. Dixon, Paul R. Cohen, and Melvin K. Simmons. Dominic ii: Meta-level control in iterative redesign. In *AAAI88*, pages 25–30, 1988.