AI Magazine Volume 9 Number 4 (1988) (© AAAI)

How Evaluation Guides AI Research

Paul R. Cohen and Adele E. Howe

Evaluation should be a mechanism of progress both within and across AI research projects. For the individual, evaluation can tell us how and why our methods and programs work and, so, tell us how our research should proceed. For the community, evaluation expedites the understanding of available methods and, so, their integration into further research. In this article, we present a five-stage model of AI research and describe guidelines for evaluation that are appropriate for each stage. These guidelines, in the form of evaluation criteria and techniques, suggest how to perform evaluation. We conclude with a set of recommendations that suggest how to encourage the evaluation of AI research.

Evaluation means making observations of all aspects of one's research. Often, we think of evaluation only in terms of how well AI systems perform, yet it is vital to all stages of research, from early conceptualization to retrospective analyses of series of programs. Indeed, some of the most informative observations are not performance measures but, rather, describe why we are doing the research, why our tasks are particularly illustrative, why our views and methods are a step forward, how completely they are implemented by our programs, how these programs work, whether their performance is likely to increase or has reached a limit (and why), and what problems we encounter at each stage of our research. To the research community, these observations are more important than performance measures because they tell us how research should proceed.

Thus, ideally, evaluation should be a mechanism of progress both within and across individual AI research projects. Evaluation provides impetus to the research cycle. It opens new avenues of research because experiments often raise new questions as others are answered and because it identifies deficiencies and, thus, problems for further research. Evaluation provides a basis for the accumulation of knowledge. Without evaluation, we cannot replicate results. Evaluation is how you convince the community that your ideas are worthwhile, that they work and how. Because our colleagues rarely have access to run-time data and programs at various development stages, we have an unprecedented responsibility to evaluate and communicate all aspects of our research. Unfortunately, we rarely publish performance evaluations and, still less, evaluations of other research stages.

Evaluation is not standard practice in part because our methodology is vague. Where other sciences have standard experimental methods and analytic techniques, we have faith—often groundless and misleading—that building programs is somehow informative. Where other sciences expect specific aspects of research to be presented (for example, hypotheses, related research, experimental methods, and analyses and results), empirical AI has no comparable standards. Now, we do not advocate blindly adopting the empirical methods of, say, the behavioral sciences; in fact, we argue elsewhere that they are inappropriate to AI (Cohen and Howe 1988). However, we are saying that progress in AI would be amplified by a methodology which emphasizes evaluation for the overarching reason that we can do our own research better if we know, in detail, the state of our colleagues' research.

Where will this methodology—and, specifically, the evaluation criteria, experiment designs, and analytic tools—come from? The challenge is that we must develop it ourselves. Explorations along these lines have been reported in specific areas, for example, machine learning (Langley 1987), distributed systems (Decker 1988), and expert systems (Geissman and Schultz 1988; Gaschnig et al. 1983; Rothenberg et al. 1987). Discussions of a broader scope focus on the role of experiments in AI (Buchanan 1987; Newell and Simon 1976). Although these papers address different facets of evaluation, a common theme is that we must develop our own evaluation methods appropriate to AI practice. If we try to retrofit the

1. Is the task significant? Why?
   (a) If the problem has been previously defined, how is your reformulation an improvement?

2. Is your research likely to meaningfully contribute to the problem? Is the task tractable?

3. As the task becomes specifically defined for your research, is it still representa tive of a class of tasks?

4. Have any interesting aspects been abstracted away or simplified?
   (a) If the problem has been previously defined, have any aspects extant in the earlier definition been abstracted out or simplified?

5. What are the subgoals of the research? What key research tasks will be or have been addressed and solved as part of the project?

6. How do you know when you have successfully demonstrated a solution to the task? Is the task one in which a solution can be demonstrated?

Figure 1. Criteria for Evaluating Research Problems.

methods of other fields, they will probably become impediments. Thus, in this article, we are motivated not by envy of "scientific method," but by the sense that we are wasting opportunities to understand, by empirical and analytic studies, the intelligent artifacts we build at great expense.

This article adopts a simple multistage model of AI research and describes the kinds of evaluation that are appropriate at each stage. (Bruce Buchanan [1987] offers a similar model.) Evaluative questions for each stage are listed in figures 1–5 and discussed in the text. We ask these questions whenever we hear about new research—questions we want researchers to answer when they publish their work. We also describe in schematic form some common kinds of experiments with AI systems. Where possible, we offer illustrations from the AI literature. A companion paper goes one step further and presents three detailed case studies (Cohen and Howe 1988).

The tangible contribution of the article is the organization of these evaluative questions and experiment schemas into five categories that correspond with stages of research. More important, by far, is the possibility that the article will promote discussion and the development of evaluation techniques and, more broadly, AI methodology. We recognize the impediments to methodological discussions. Some might regard methodology as dangerously prescriptive, while others regard it as irrelevant: A reviewer said we are wasting our time because AI methodology is just fine. A friend dismissed this presentation as a metapaper, and not worth the time it would detract from substantive research. However, evaluation was a central issue at the recent meeting of the Program Committee of the American Association for Artificial Intelligence—not a general, perennial complaint such as bad writing but a vital, current problem in need of creative solutions. How does one evaluate AI research? This article doesn't answer the question fully; it is intended to give us a place to begin.

Evaluating Each Stage of Research

We view empirical AI in terms of a five-stage cycle: In the first stage, a topic is refined to a task and a view of how to accomplish the task; in the second, the view is refined to a specific method; in the third, a program is developed to implement the method; in the fourth, experiments are designed to test the program; in the fifth, the experiments are run. finally, the cycle begins again, informed by the experiments' results.

You might object that this approach isn't how AI research works. First, this superficial description seems severely top down. However, we show how evaluations can produce unexpected conclusions and, thus, iterations at each stage of the cycle. Second, the model seems idealized because it suggests that AI researchers always design, run, and analyze experiments, and we don't. However, although it is idealized, it isn't unattainable and, in fact, won't require many modifications to standard AI practice. Third, this model is probably more appropriate to individual research projects than to huge, collaborative efforts such as the Pilot's Associate or the Autonomous Land Vehicle. These projects require different models and different evaluation criteria; however, we believe the five-stage model characterizes much AI research.

Stage 1: Refine the Topic to a Task

Empirical AI begins when researchers find particular topics fascinating. The first stage of the research cycle involves simultaneously refining the research topic to a task and identifying a view. A task is something we want a computer to do, and a view is a predesign, a rough idea about how to do it. This stage takes a lot of effort; researchers don't simply say, "Ok, we are fascinated by discovery, so let's try mathematical discovery as a task and heuristic search as a view" (Lenat 1976). The process is iterative because if the task and view don't fare well by the evaluation questions in figure 1, you have to modify and reevaluate them.

The questions in figure 1 address two basic evaluation concerns: Can you justify the research task to yourself and the community, and is your view of how to solve the task viable? (In the questions, we sometimes refer to the problem, by which we mean the task and the view combined.)

Question 1 asks us to justify the task: Why is it interesting? If it has been studied before, and you are advocating a novel view (a reformulation), why do you expect this new perspective to be an improvement? Question 2 asks whether one's view of the task will work. Perhaps the task itself is intractable or is impossible given the resources, or requires expertise or hardware that isn't available. The question is especially important if one's research method is to "build it and see what happens." The problem with this approach is not that it is relatively undirected; all sciences have exploratory aspects. The problem is that AI is expensive. We generally cannot afford to spend months or years implementing a system unless we're pretty sure something interesting will happen. (A related question, whether the interesting stuff is obscured or observable, is discussed in the subsection on stage 4.)

Question 3 suggests reevaluating whether after narrowing the scope of a task, it still exemplifies a research topic. Most AI tasks, although specific, are designed to exemplify general tasks. For example, we built a system to design pulley systems and heat sinks, confident that they exemplified the class of mechanical design problems which can be solved without decomposition into smaller subproblems (Howe et al. 1986; Howe 1986). Question 4 asks what has been left out of research tasks. In the case of our work on mechanical design, we intentionally dropped the tasks of decomposing and reintegrating design subproblems; so, we knew in advance that we would produce a model of designing objects which required no decomposition, not a general model of mechanical design.

Questions 5 and 6 address whether we have a productive research plan. Most large projects require solutions to multiple problems, such as designing representations and control strategies and building knowledge bases. Question 5 concerns which of these solutions will be research contributions of the project. Question 6 asks whether you understand the problem well enough to recognize a solution.

Evaluations during this stage direct one's own research and also provide the AI community with carefully justified tasks, views, and reformulations. It is sometimes worth publishing work at this stage, long before anything has been implemented and demonstrated. Good contemporary examples are some early papers on reactive planning (Agre 1985; Chapman and Agre 1987) and papers related to the CYC project (Lenat, Prakash, and Shepherd 1986; Lenat and Feigenbaum 1987). Many of the questions in figure 1 are asked and answered in these papers. Although preliminary, these evaluations can be informative, and it would be foolish to reject these

1. How is the method an improvement over existing technologies?
   (a) Does it account for more situations (input)?
   (b) Does it produce a wider variety of desired behaviors (output)?
   (c) Is the method expected to be more efficient (space, solution time, development time, and so on)?
   (d) Does it hold more promise for further development (for example, because of the introduction of a new paradigm)?
2. Does a recognized metric exist for evaluating the performance of your method (for example, is it normative, cognitively valid)?
3. Does it rely on other methods? (Does it require input in a particular form or preprocessed input? Does it require access to a certain type of knowledge base or routines?)
4. What are the underlying assumptions?
5. What is the scope of the method?
   (a) How extendible is it? Will it easily scale up to a larger knowledge base?
   (b) Does it exactly address the task? Portions of the task? A class of tasks?
   (c) Could it or parts of it be applied to other problems?
   (d) Does it transfer to complicated problems (perhaps knowledge intensive or more or less constrained or with complex interactions)?
6. When it cannot provide a good solution, does it do nothing or does it provide bad solutions or does it provide the best solution given the available resources?
7. How well is the method understood?
   (a) Why does it work?
   (b) Under what circumstances, won't it work?
   (c) Are the limitations of the method inherent or simply not yet addressed?
   (d) Have the design decisions been justified?
8. What is the relationship between the problem and the method? Why does it work for this task?

Figure 2. Criteria for Evaluating Methods.

papers because they don't answer the questions in figures 2–5.

Stage 2: Design the Method

At this stage, one's view is refined to a method for solving the task. The word method implies a single algorithm, such as A*, candidate elimination, or Waltz filtering. Frequently, however, the method for a task combines several algorithms and assorted knowledge structures. For example, the method in Hearsay-II involved many knowledge sources, data-driven and opportunistic control, and a novel communication structure (Erman et al. 1980). Although this complexity strains the word method, we maintain it to remind us that we don't jump immediately into building programs but first decide how we want to solve tasks.

Criteria for evaluating methods are presented in figure 2. The first two questions suggest general criteria for comparing the method with previous work. These questions are prospective because we might not be able to answer them without implementing the method. However, before we build anything, it's worth asking whether we expect our approach to be better, in some sense, than those which are already available. Question 1 documents some common interpretations of "better." Note that in the early going, a method need not necessarily be better, just different if the difference holds promise (question 1d). Question 2 asks what aspects of the method's performance will be evaluated. If the answer to question 1 is, "My method is better because it is more general," then question 2 asks, "How will this difference be demonstrated?"

Question 3 asks whether and to what extent these evaluations are qualified by the method's reliance on other factors. Sometimes, the answer is obvious; for example, one might have a method for discovering physical laws, but one's evaluations of the method should note whether it plans its own experiments or, instead, relies on a person or another method to present data in an order that facilitates discovery. However, question 3 can also be difficult to answer prospectively because the word rely is vague. Interacting components of a method rely on each other in the trivial sense that without them all, the method wouldn't work; however, other kinds of reliance are more difficult to see prospectively. Indeed, it took several years for Lenat and Brown (1983) to recognize and document the reliance of AM on its representation language.

Question 4 asks about one's assumptions. In our work on mechanical design, we initially assumed that all design variables (for example, diameter of a pulley) were continuous. Some planning algorithms assume that the state of the world is known and that the effects of all actions are predictable. Some vision algorithms assume noise is distributed uniformly. It is important to state our assumptions if only to prevent others from misstating them! For example, the uncertainty literature has suffered a decade-long debate about what the conditional independence assumption really means.

It might be difficult to answer the remaining four questions prospectively. Sometimes, one knows the scope of a method before implementing it, sometimes not (question 5). One might anticipate that a method will easily scale up, especially if it has been designed to replace one which will not, but one typically cannot prove it. Question 6 is perhaps easier to answer prospectively because it reflects an important design criterion for methods. Some methods are designed to optimize, some to satisfice, and some to simply quit when resources run out. For example, Victor Lesser and his colleagues recently described a set of approximate processing methods for real-time problem solving (Lesser, Durfee, and Pavlin 1987). By design, these methods are intended to give the best possible performance in the time available. However, lacking analytic tools, the actual performance of these methods might be impossible to predict. Then, question 6 cannot be answered prospectively. Similarly, questions 7 and 8 can rarely be answered except by empirical work. Indeed, our inability to answer these questions analytically is often the impetus for building programs.

The purpose of the questions in figure 2 is to convince oneself and the community of the utility, scope, assumptions, performance, and limitations of a method. Often, these evaluations are sufficient, and papers can be published without further evaluation. An example is DeKleer's first paper on the assumption-based truth maintenance system (ATMS), which answers many of the questions in figure 2 analytically: "[It] is possible to work effectively and efficiently with inconsistent information, context switching is free, and most backtracking (and all retraction) is avoided. These capabilities motivate a different kind of problem-solving architecture in which multiple potential solutions are explored simultaneously. This architecture is particularly well-suited for tasks where a reasonable fraction of the potential solutions must be explored" (DeKleer 1986, abstract).

Analyses of methods are rarely so thorough in AI, but even when they are, one can elect to continue to subsequent stages in the research cycle. This continuation usually involves building a program. We discern three good reasons to build programs. first, it is often worthwhile to implement the method to experiment with problematic aspects of its behavior (for example, Forbus and DeKleer [1988] recently built a couple of simple programs to explore focusing in the ATMS). Second (and more frequent), no other way exists to answer some of the questions in figure 2: Why does the method work for this particular task? Do tasks exist for which it won't work? Are the limitations of the method inherent? and so on. We eventually implemented our mechanical design method in a program and ran 125 experiments on different configurations of the program, selectively ablating components of the method because we had no other way to probe how interactions between these components affected design performance (Howe 1986).

The third role of programming is exploratory. We might try out some ideas to help us refine our method. We think it is important to distinguish this role of programming—refining a method—from programming for the purpose of experimenting with and evaluating a method. Unfortunately,

exploratory programming drifts easily into building systems, and we begin to focus on solving the task and forget that from the standpoint of empirical AI research, the purpose of building systems is to tell us something about our methods which we don't already know and can't learn by analysis. We build too many systems and evaluate too few.

Stage 3: Build a Program

If our method requires programming not merely to implement it but to understand whether and why it works, then we must build a program that supports experiments. Although in practice this stage might be indistinguishable from exploratory programming in the previous stage, its purpose is different and so are the evaluations we do at this stage.

The criteria in figure 3 concern whether the program is informative. Question 1 asks whether its internal and external behavior clearly demonstrates the method. Question 2 implies that programs are rarely informative if they are designed to run on only a single example. Question 3 suggests specific ways to assess how well the program implements the method. Finally, because the purpose of building a program is to tell us something we didn't already know, question 4 asks whether we can predict performance in advance.

The questions in figure 3 (especially question 4) illustrate the iterative nature of evaluation at each stage: We don't build a program first and then throw it away if it is uninformative or utterly predictable; we develop informative, interesting programs by continuing evaluation—just as we iteratively evaluate and develop tasks, views, methods, and experiments. However, unlike the earlier evaluations of tasks, views, and methods, evaluation at this stage is primarily for the individual researcher, not for the community at large.

Stage 4: Design Experiments

The fourth stage of the research cycle is to design experiments with the newly implemented system. Just as we evaluate the design and construction of systems from the standpoint of

1. How demonstrative is the program?
   (a) Can we evaluate its external behavior?
   (b) How transparent is it? Can we evaluate its internal behavior?
   (c) Can the class of capabilities necessary for the task be demonstrated by a well-defined set of test cases?
   (d) How many test cases does it demonstrate?
2. Is it specially tuned for a particular example?
3. How well does the program implement the method?
   (a) Can you determine the program's limitations?
   (b) Have parts been left out or kludged? Why and to what effect?
   (c) Has implementation forced a detailed definition or even reevaluation of the method? How was this reevaluation accomplished?
4. Is the program's performance predictable?

Figure 3. Criteria for Evaluating Method Implementation.

1. How many examples can be demonstrated?
   (a) Are they qualitatively different?
   (b) Do these examples illustrate all the capabilities that are claimed? Do they illustrate limitations?
   (c) Is the number of examples sufficient to justify the inductive generalizations?
2. Should the program's performance be compared to a standard such as another program, or experts and novices, or its own tuned performance? Should the standard be normative, or cognitive validity, or outcomes either from the real world or from simulations?
3. What are the criteria for good performance? Who defines the criteria?
4. Does the program purport to be general (domain-independent)?
   (a) Can it be tested on several domains?
   (b) Are the domains qualitatively different?
   (c) Do they represent a class of domains?
   (d) Should performance in the initial domain be compared to performance in other domains? (Do you expect that the program is tuned to perform best in domain(s) used for debugging?)
   (e) Is the set of domains sufficient to justify inductive generalization?
5. Is a series of related programs being evaluated?
   (a) Can you determine how differences in the programs are manifested as differences in behavior?
   (b) If the method was implemented differently in each program in the series, how do these differences affect the generalizations?
   (c) Were difficulties encountered in implementing the method in other programs?

Figure 4. Criteria for Evaluating the Experiments' Design.

whether they are informative, so we should evaluate whether experiments with these systems will be informative. Criteria for evaluating experiments (as opposed to their outcomes) are presented in figure 4. The purpose of question 1 is to encourage us to test our programs with many, qualitatively different examples that illustrate the abilities and limitations of our programs. Questions 2 and 3 ask whether our programs should be compared with a standard (for example, an expert) and what specific comparisons will be made. Question 4 asks whether our experiments provide enough evidence to claim that a program is general, and question 5 suggests criteria for comparing programs. Both questions acknowledge that programs (and their underlying methods) are evaluated in the context of other research projects and, so, raise the methodological issues of how results accumulate and generalize over projects (see also Conclusion and Cohen and Howe [1988]).

These questions address what can be evaluated in experiments but don't say how the experiments should be conducted. AI has been evolving rudimentary, informal experiment schemas that address some of the questions in figure 4. Here, we describe five experiment schemas:

Comparison studies. In the basic form of a comparison study, we select one or more measures of a program's performance; then, both the program and a standard solve a set of problems; and, finally, the solutions are compared on the measures. For example, we can compare the average number of subgoal violations generated by one planning program on a set of problems (the measure) with the same measure on another extant program (the standard). Typically, the programs implement different methods, or they can be different configurations of a single program.

Variations on the basic form depend on what you want to demonstrate. For example, if you want to measure whether the program's performance is consensual, you can compare the program to a panel of human experts. You can also include novices—an interesting control condition to ensure that successful performance requires expertise (for example, Shortliffe 1976 ran a panel of experts and novices in his studies of MYCIN). Sometimes, the performance of a program can be compared with objective, recognized standards. Normative theories, such as probability theory, provide one kind of standard; for example, some researchers argue that because human experts are incapable of integrating probabilistic information consistently, their performance should not set standards. Another kind of standard is provided by real or simulated worlds; we might evaluate a complex planner by seeing whether it generates plans that succeed in the world. All these examples suggest that our measures and standards depend heavily on what we want to demonstrate and, ultimately, on our research goals.

Direct assessment. A related scheme, though not strictly a comparison study, has humans judging or scoring the program's performance. Direct assessment happens when test problems have so many acceptable solutions that a program and a standard cannot be expected to generate the same ones. For example, we built a system some years ago that generated portfolios for investors (Cohen and Lieberman 1983), none of which were identical with portfolios generated by an expert for the same problems. Our program's portfolios were not bad but, rather, there are many good ways to generate portfolios, and portfolios rarely coincide by chance. In these cases, one cannot compare the program's performance with the expert's but must instead rely on the expert's direct assessment of the program. Unfortunately, this test is weaker than a comparison because experts can be overly generous to the program. Moreover, direct assessment does not tell us whether the program is performing better than the expert.

Ablation and substitution studies. We can evaluate the contribution of individual components to the performance of complex systems by removing or replacing these components. Removing components (ablation [Newell 1975]) is informative in systems that can solve problems without these components; for example, we might assess the contribution of caching by running a system without caching. It takes little insight to predict some effect; the goal is to find out whether performance on all types of problems is equally affected by the presence of a cache and, if not, what interactions between the cache and the problem type explain the variance.

Many AI systems are so brittle that they collapse when components are removed. In these cases, we might substitute dumb components for those we hope to show are smart; for example, we might substitute an exhaustive control strategy for a sophisticated, opportunistic one.

Tuning studies. By tuning a system to perform as well as possible on a set of test data, we can learn how much performance can be improved, how difficult it is to achieve, and whether the resulting system can still solve other test cases. From a research perspective, it seems wasteful and potentially misleading to tune systems just to increase their performance without addressing these questions.

Limitation studies. By testing a program at its known limits, we can better understand its behavior in adverse conditions—we can learn whether it is robust. We can push a program to its limits by providing imperfect data (rearranged, noisy, incomplete, or incorrect), restricted resources (computation time or available knowledge), and perverse test cases.

Inductive studies. One way to support claims of generality is to solve new and different problems. If we claim that, say, a mechanical design system is general, then we might want to run problems in many areas of mechanical design—pulley systems, I beams, extrusions, and so on (Howe et al. 1986; Orelup 1987). Even if we don't claim a program is general (and we question whether inductive studies, in fact, demonstrate generality in the conclusion), we must, at least, test it on problems other than those we used to develop it.

The purpose of evaluation at this stage is to convince the researcher and the community that studies of a program (or programs)independent of their results—are well designed and

complete. Experiment schemas, if we could specify them in adequate detail, would offer researchers a shorthand to describe their studies (for example, "We ran a comparison study between versions 1, 2, and 3 of the program, measuring hit rate on three data sets that were characterized by their average signal-to-noise ratio").

Stage 5: Analyze the Experiments'' Results

Now that our method has been implemented in a fully instrumented program, and our experiments are well designed, we can ask whether the system works and why it works. The purpose of evaluation at this stage is to convince the research community that your methods are viable, and their limitations in both performance and scope, and suggest further research. The first three questions in figure 5 evaluate the program by its performance on the task. Questions 4, 5 and 6 assess the utility of the program. Questions 7 and 8 identify the program's frailties and strengths: When does it break, and what components contribute to its successful operation? Question 8e is most salient to the generalization of results because it justifies good performance for the program in terms of the task, a relationship that might be exploited in the design of other methods.

1. How did program performance compare to its selected standard (for example, other programs, people, normative behavior)?
2. Is the program's performance different from predictions of how the method should perform?
3. How efficient is the program in terms of space and knowledge requirements?
4. Did the program demonstrate good performance?
5. Did you learn what you wanted from the program and experiments?
6. Is it easy for the intended users to understand?
7. Can you define the program's performance limitations?
8. Do you understand why the program works or doesn't work?
   (a) What is the impact of changing the program even slightly?
   (b) Does it perform as expected on examples not used for debugging?
   (c) Can the effect of different control strategies be determined?
   (d) How does the program respond if input is rearranged, noisy, or missing?
   (e) What is the relationship between characteristics of the test problems and performance (either external, or internal if program traces are available?)
   (f) Can the understanding of the program be generalized to the method? To characteristics of the method? To a larger task?

Figure 5. Criteria for Evaluating What the Experiments Told Us.

The questions in figure 5 are what most people regard as evaluation. Consequently, they neglect the earlier questions and end up with unclear or unconvincing answers at this stage. If a program hasn't been constructed in such a way that its performance is observable (figure 3), and the experiment hasn't been designed with clear or convincing criteria and control conditions (figure 4), then this stage of evaluation is not likely to be informative. Evaluation begins early in the research cycle. It can't be tacked to the end. We can't ask how a program compares with its standard (question 1) unless we have a standard, and it isn't convincing to dream one up after we have built the program! Even if you have a standard in mind, you should assure yourself that it makes sense: As noted earlier, we had a vague idea that we would compare the performance of our portfolio management program with the performance of a portfolio manager, but it turned out (too late) there was no way to make this comparison. Similarly, question 2 builds on earlier stages of evaluation: To compare a program's performance to predictions about performance, one needs predictions! One cannot simply dream up a few strawman predictions to give the impression of empiricism. Yet, we see few reports that include predictions and fewer still in which the program performs other than as predicted. The same point can be made about several other questions in figure 5: It will be difficult to determine performance limitations or why a program works or doesn't work in a post hoc manner.

Conclusion

Each stage in our simple model of research affords opportunities for evaluation; indeed, iterations at each stage and cycles throughout the model are driven by evaluation. This is the first step in what should become an ongoing discussion of AI methodology. We need to develop the model further—or alternative models as necessary—so we have a common language for discussing the stages of our research and the kinds of evaluation appropriate to each. The research cycle and the questions and experiments outlined here form only a skeleton of a methodology. One way to slowly fill out this skeleton is to carefully describe our experiments and results. Another is to

publish methodological papers. For example, we want to see extensive discussions along the following lines.

First, we encourage discussions of experiment schemas, with appropriate control conditions, measures, standards, and analyses of results. We hope to see schemas that are more detailed and task specific than the few we discussed in the stage 4 subsection. We also hope to see discussions of specific evaluation criteria in the context of experiment schemas. We know that the criteria we presented in gures 1–5 are, in some respects, a bad compromise: too general to be informative and not general enough to encompass all research in empirical AI (for example, they don't tell us how to evaluate methodological papers such as this one).

A second topic for discussion is the methodological role of programs. Why should we build programs? This topic is an important part of a broader debate on the empirical AI research cycle, which requires considerable elaboration beyond the sketch given earlier.

Third, and most important: What does it mean to replicate and generalize results? When can we claim that a particular method is general—when it has been implemented in one program and run on many test cases, or when it has been implemented in many programs in different task domains? Perhaps we should be asking, instead, how we can bound the applicability of methods. Are we obliged to claim generality on inductive grounds (for example, the program ran in several domains)? We don't say, "The sun will rise because it always has." Rather, "Here's why the sun will rise." Is there a similar deductive notion of generality in AI? This methodological problem is perhaps our most difficult: How do we know that a result is general, that we don't have to build any more systems to illustrate it? How do we know, conversely, that two apparently different results are, in fact, demonstrations of the same thing? Given that AI programs are different (they have different code, algorithms, machines, domains, performance characteristics, assumptions, and underlying purposes), can we abstract any results from them that we can agree are common? (See Cohen and Howe [1988] for further discussion.)

One concrete recommendation is that editors, program committees, and reviewers should begin to insist on evaluation. Editorial policies rarely preclude evaluation—we all think it's a good idea—but neither do they require them. This is one reason that papers almost never describe an entire research cycle—a problem, a task, a view, methods, programs, experiments, and results. Editorial policy is perhaps the most effective way to guide empirical AI practice toward complete evaluation. (Lately, the machine learning community has explicitly mentioned evaluation criteria in calls for papers, and Langley (1987) has advocated evaluation in his editorials.)It can also encourage the following uncommon kinds of papers.

Short studies with extant systems. Many of the experiments discussed in the subsection on stage 4, which address how and why systems work, are extensive, self-contained, publishable studies. Even after a system has been thoroughly explored, it can still serve as a test bed for studies with other methods, knowledge representations, control structures, and so on. AI systems take so long to build that we really cannot afford to drop them until we have learned all we can from them. A good model of this kind of work is a book on extensive experiments with MYCIN edited by Buchanan and Shortliffe (1984).

Negative results. For example, we need to know when methods don't work as expected, when systems perform less well as they become more knowledgeable, when scaling up causes problems. When did you last read an AI paper that said something didn't work?

Progress reports. We should publish progress reports throughout the development of large AI projects, not wait until they are finished. Lenat's CYC project (Lenat, Prakash, and Shepherd 1986), for example, is slated to continue throughout the 1990s. We cannot afford to wait until then to see how CYC is progressing—especially because it involves pioneering techniques and ideas that we need much sooner.

When we began writing this article, we took the view that science is not legislated, controlled, or forced upon its practitioners but is a voluntary business in which loose organizations of individuals somehow produce coherent, cumulative progress. We obviously believe evaluation and methodology will help this process; it isn't hard to do, and it suggests new research, generalizations, and hypotheses that one wouldn't discover otherwise.

Acknowledgments

This research is funded by the Office of Naval Research (ONR) under a University Research Initiative grant, contract #N00014-86-K-0764, and ONR contract AFOSR 4331690-01. We are indebted to Scott Anderson, Carole Beal, Carol Broverman, David Day, Mike Greenberg, Tom Gruber, Janet Kolodner, Pat Langley, Cynthia Loiselle, and Paul Utgoff for their comments on drafts of this paper and to Peter Patel-Schneider for sending us guidelines for reviewers.

References

Agre, P. E. 1985. The Structures of Everyday Life, Technical Report, Working Paper 297, Massachusetts Institute of Technology.

Buchanan, B. G. 1987. Artificial Intelligence as an Experimental Science, Technical Report, KSL 87-03, Knowledge Systems Lab., Dept. of Computer Science, Stanford Univ.

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

Chapman, D., and Agre, P. E. 1987. Abstract Reasoning as Emergent from Concrete Activity. In Reasoning About Actions and Plans, Proceedings of the 1986 Workshop, eds. M. P. Georgeff and A. L. Lansky, 411-424. Los Altos, Calif.: Morgan Kaufmann.

Cohen, P. R., and Howe, A. E. 1988. Toward AI Research Methodology: Case Studies in Evaluation. IEEE Transactions on Systems, Man, and Cybernetics. In press.

Cohen, P. R., and Lieberman, M. D. 1983. Folio: An Expert Assistant for Portfolio Managers. In Proceedings of the Eighth International Joint Conference on Artificial Intelligence, 212215. Menlo Park,

Calif.: International Joint Conferences on Artificial Intelligence.

Decker, K. S., Durfee, E. H., Lesser, V. R. Evaluating Research in Cooperative Distributed Problem Solving. Dept. of Computer and Information Science. TR 8889, 1988. Univ. of Massachusetts.

DeKleer, J. 1986. An Assumption-Based TMS. Artificial Intelligence 28:127162.

Erman, L. D.; Hayes-Roth, F.; Lesser, V. R.; and Reddy, D. R. 1980. The HEARSAY-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty. Computing Surveys 12: 213253.

Forbus, K., and DeKleer, J. 1988. Focusing the ATMS. In Proceedings of the Seventh National Conference on Artificial Intelligence, 193198. Menlo Park, Calif.: American Association for Artificial Intelligence.

Gaschnig, J.; Klahr, P.; Pople, H.; Shortliffe, E. H.; and Terry, A. 1983. Evaluation of Expert Systems: Issues and Case Studies. In Building Expert Systems, eds. F. Hayes-Roth, D. A. Waterman, and D. B. Lenat, 241280. Reading, Mass.: Addison-Wesley.

Geissman, J. R., and Schultz, R. D. 1988. Verification and Validation. AI Expert 3(2): 2633.

Howe, A. 1986. Learning to Design Mechanical Engineering Problems, Technical Report, EKSL Working Paper 86-01, Univ. of Massachusetts.

Howe, A.; Dixon, J. R.; Cohen, P. R.; and Simmons, M. K. 1986. Dominic: A Domain-Independent Program for Mechanical Engineering Design. International Journal for Artificial Intelligence in Engineering 1(1): 2329.

Langley, P. 1987. Research Papers in Machine Learning. Machine Learning 2(3): 195198.

Lenat, D. B. 1976. AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search. Ph.D. diss., Dept. of Computer Science, Stanford Univ. Also, Technical Report, STAN-CS-76-570, Dept. of Computer Science, Stanford Univ.

Lenat, D. B., and Feigenbaum, E. A. 1987. On the Thresholds of Knowledge. In Proceedings of the Tenth International Joint Conference on Artificial Intelligence, 11731182. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

Lenat, D. B.; Prakash, M.; and Shepherd, M. 1986. CYC: Using Common Sense Knowledge to Overcome Brittleness and Knowledge Acquisition Bottlenecks. AI Magazine 6(4): 6585.

Lenat, D. B., and Brown, J. S. 1983. Why AM and Eurisko Appear to Work. In Proceedings of the Third National Conference on Artificial Intelligence, 236240. Menlo Park, Calif.: American Association for Artificial Intelligence.

Lesser, V. R.; Durfee, E. H.; and Pavlin, J. 1988. Approximate Processing in Real-Time Problem Solving. AI Magazine 9(1): 49-61.

Mitchell, T. M. "Version Spaces: A Candidate Elimination Approach to Rule Learning." In Proceedings of the Fifth International Joint Conference on Artificial Intelligence, 305310.

Newell, A. 1975. A Tutorial on Speech Understanding Systems. In Speech Recognition: Invited Papers Presented at the 1974 IEEE Symposium, ed. D. R. Reddy, 354. New York: Academic.

Newell, A., and Simon, H. A. 1976. Computer Science as Empirical Inquiry: Symbols and Search. Communications of the ACM 19(3): 113126.

Orelup, M. F. 1987. Meta-Control in Domain-Independent Design by Iterative Redesign. M.S. thesis, Mechanical Engineering Dept., Univ. of Massachusetts.

Rothenberg, J.; Paul, J.; Kameny, I.; Kipps, J. R.; and Swenson, M. 1987. Evaluating Expert System Tools: A Framework and Methodology, Technical Report, R-3542-DARPA, Rand Corporation.

Shortliffe, E. H. 1976. Computer-Based Medical Consultations: MYCIN. New York: American Elsevier.