Special Feature

Teaching an Introductory Course in Expert Systems

A. Terry Bahill and William R. Ferrell
University of Arizona

In the fall of 1985, 25 students and 10 auditors (including three faculty members) attended our new course on expert systems—an introductory engineering course aimed at teaching useful technical skills and applying those skills to real problems. The course had three objectives: to share understanding, to impart techniques, and to apply these in practice. We wanted our students

(1) To understand the nature, limitations, and suitable applications of expert systems;

(2) To effectively use expert system shells (software packages allowing rapid prototyping of expert systems, shells can be considered special-purpose high-level languages designed to help write if-then production rules); and

(3) To produce a small expert system.

In these objectives, the course differed markedly from many other AI and expert system courses. Other courses have attempted, variously, either

(1) To teach the design and programming of inference engines;

(2) To provide an understanding of heuristic programming methods;

(3) To survey AI research;

(4) To explore AI and expert system applications in specific problem domains; or

(5) To teach programming methods or languages especially suited to knowledge-based systems.

The teaching experience convinced us that students can learn, in a one-semester course, to apply effective fundamental skills in expert systems development—skills providing a sound basis for further growth. But we also found, from observing student attempts at applications, that the course needed improvement in imparting technique and in developing the understanding needed for technical application. Consequently, we have revised the course for the fall of 1986.

We shall first present what we did, then discuss the students' projects, and finally describe the changes needed in light of those projects.

The course

We used Randy Davis videotapes[1] for a broad AI overview, Forsyth's edited volume on expert systems,[2] and an expert systems textbook by Harmon and King.[3] We bought the M.1 shell instructor's package from Teknowledge including 10 copies of M.1, overhead transparencies, and lecture notes. The package, while expensive ($5000), was worthwhile.

Building a knowledge-based system using Pascal, Prolog, or other general-purpose, high-level language is too big a task for a one-semester course. And using a big knowledge-engineering tool such as KEE or ART is too complicated for an introductory course. We concluded that the best way to teach expert systems in a one-semester, introductory course would be to have students build an expert system on a PC using an expert system shell.

0885-9000/86/1100-0059 $01.00©1986 IEEE

The class discussed several recent papers comparing currently available expert system shells that run on PCs.[3-6] Comparing and contrasting these PC shells, and examining such historically famous expert systems as MYCIN,[7] was an important part of the course.

Shells we considered for course use

Even back in early 1985, when we were planning this course, there were scores of commercially available expert systems shells. We purchased three—two for the VAX, and one for PCs.

The Rand Corporation's ROSIE,[8] though several years old, is not heavily used. It was inexpensive ($200), ran on a VAX, and needed InterLisp (although the latest version runs with portable standard Lisp). We abandoned ROSIE because it was big, cumbersome, and slow.

M.1 from Teknowledge, a backward-chaining shell for making expert systems on an IBM-compatible PC, was easy to learn and easy to produce user-friendly systems with. M.1 has an effective system for dealing with uncertainty, many automatic features (such as formulating its own questions when they are not specified by the developer), good trace capability, and its knowledge base can be written without complicated formats. We did not buy other shells similar in power and features, such as Personal Consultant Plus, because we could not afford two tools from the same class.

OPS5, the expert system shell supported by DEC, is a forward-chaining system suitable for different problems than are the backward-chaining M.1 and ROSIE—forward chainers work *toward* a goal state, while backward chainers work *from* a goal state. DEC has used OPS5 in-house for dozens of projects, although soon DEC will also support S.1 from Teknowledge. There is a textbook explaining OPS5,[9] and the shell is available for PCs and the VAX for between $75 and $3000. We found OPS5 was difficult for students to use and it was difficult for them to make user friendly systems with.

After completing the course, we discovered Micro Expert from McGraw Hill—costing $50.[10] It could serve as the basis for a low-budget expert systems course.

Student projects

We required students, individually or in pairs, to find an expert and (using an expert system shell) to make an expert system. Their experts formed a motley group—professors, physicians, siblings, roommates, parents, teachers, and user consultants. The availability of experts was a prime factor determining expert system topics, although our suggestions on their one-page proposals had some effect.

Most students enjoyed their projects. Several have continued working while enrolled for independent study, which brings up a disappointing aspect of Teknowledge's license agreement: Students cannot take a working version of their expert system with them. They are allowed to use M.1 only while enrolled in a course.

From these class projects, listed in the accompanying box, we gleaned clues about what problems are suitable for expert systems and what problems are best left for conventional computer programs. Some class problems were inappropriate for expert systems because the implementations were simple checklists or questionnaires—some because they involved numerical computation rather than symbolic manipulation. Others were inappropriate because no human expert could solve the problem, or because the solution could have been represented unequivocally as a decision tree in which users were shunted up different branches depending on what answer the system posed. For such cases, a diagram on paper would have been as effective as the computer system—a simple recipe for an answer.

How can one identify a task that is appropriate for an expert system? First, there must be a human expert who performs that task better than most other people. For example, Julia Child is an expert chef; most of us do not cook as well. Designing an expert system to add single digit numbers is silly, because almost everyone does this well. On the other hand, designing an expert system to predict the stock market is doomed to failure because no human expert does this consistently well (if someone does, they are just quietly salting away their millions). Figure 1 shows the type of performance histogram ideal for an expert system.

Second, the task's solution must be explainable in words rather than requiring explanatory pictures—imagine Michelangelo creating an expert system to direct the painting of the Sistine Chapel!

The constraint that experts not draw pictures leads to a third criterion—the telephone test: Can the problem be solved routinely in a 20-minute (or even a one-hour) telephone conversation with the expert? If so, the problem suits a PC-based expert system. If the problem takes a human two days to solve, however, then it's far too complicated for an expert system. And if a human can answer in two seconds, it's too simple.

Fourth, problems inviting one of many possible solutions are ideal candidates for expert systems—problems such as "What disease does the person have?"

Evaluating student projects was difficult. We could not afford to hire experts to do our grading or to test the "expert systems" on various realistic problems and compare their performance with the humans whose knowledge they were intended to capture. Furthermore, it would be unfair for two generalist professors of Systems and Industrial Engineering to judge student expert systems designed for use by specific-domain authorities. Therefore, our evaluations tended to emphasize (1) how much we could "fool" systems by plausible (but nonsensical) inputs, and (2) the difficulty we had using systems or making sense of their queries or output. We found a typical consultation's tree diagram greatly helped us to understand the knowledge base, and more fairly evaluate the expert systems.

It is also helpful having an overseer expert to critique systems. We had four students, working with four different experts, making expert systems on autism. When our primary autism expert, Linda Swisher of our Speech and Hearing Department, ran one of those expert systems she exclaimed "That lady is no expert! I mean, the rules are correct, but I can see what books she got them from." Swisher was right. The student had interviewed a pediatric neurologist who, being too busy, volunteered the services of her resident—a very intelligent professional but not an autism *expert*. Swisher correctly noted such lack of real expertise in two student-generated expert systems.

Our experts enjoyed participating and often told us so. "Working through these production rules," Linda Swisher observed, "and seeing the decision tree of a consultation made me realize what I actually do during a consultation. If nothing else, this experience will help me teach better in the future. What I have been telling my students to do for the last 15 years is not really what I do myself." As has been noted before, expressing a knowledge base can lead experts to clearer understandings of their own expertise.[11]

We were surprised to find the best (or what seemed to us the best) expert systems were written not by our systems engineering students, but rather by psychology, communications, and business majors. Moreover, our brilliant foreign students did not produce the best expert systems; interacting with their programs was far too difficult, leading us to conclude that, for making expert systems, communication skills are at least as important as engineering analysis skills. This conclusion complements advice given by Feigenbaum and Davis in the AI video conference broadcast November 13, 1985: They described ideal knowledge engineers (1) as reductionists who love to wade through myriad rules, heuristics, hunches, and intuitive notions, digesting them into a production-rule nutshell, (2) as aggressive and not afraid to take a risk, and (3) as engineers who enjoy poking their noses into other people's business. For that is what knowledge engineering is about, after all—to go into someone else's domain, learn it, cast this knowledge into an expert system, then a half a year later move off into another person's domain and repeat the process.

If they'd had time to think about it, we suspect Feigenbaum and Davis would also have mentioned the need for communications skills that aid in understanding problems, dealing with experts, and making systems that work.

Planned improvements

We want to improve interviewing techniques. Half of our students' time went into gathering knowledge from human experts and translating that knowledge into if-then production rules. The few students who were their own experts, or who got their knowledge out of textbooks, were deprived of a valuable experience.

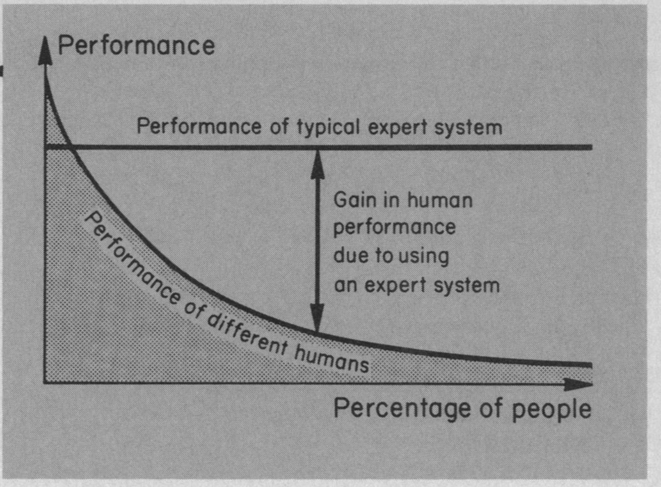


Figure 1. Problems having a performance histogram like this are ideal for expert system technology. An expert system's purpose is not to replace humans, but to increase mass productivity; systems should perform almost as well as human experts, as the figure shows.

We will acquire an induction system such as Expert-Ease, 1st-Class, or KDS because they're different in that, instead of requiring if-then production rules, they accept examples and case studies and then derive production rules. However, we're not sure it will be easier for students to generate adequately general examples than to provide the rules.

Not surprisingly, students were reluctant to face dealing with uncertain data and vague rules—expert system requirements. We are similarly reluctant. But expert systems must incorporate capacities to do so if they are to deal effectively with real problems.

To deal with imprecision and vagueness, M.1 uses certainty factors. Chapters 5-7 of Forsyth's book (rated by our students as the best chapters) helped us broaden the discussion of uncertainty by including Bayesian probability updating and fuzzy sets. In future classes, we'll spend more time dealing with imprecision, uncertainty, and their effects on solutions. Although no dogma leads to salvation (these matters being perennial problems), appreciating them thoroughly is, we think, prerequisite for designing good expert systems.

After studying the 22 student-built expert systems, we realized that students do not intuitively build good human/computer interfaces allowing easy and effective communication. In fact, they are not adequately aware of the problems involved. For example, most systems require users to enter large amounts of data. When requests for data come in long lists of questions, each presented as a separate frame without comment or explanation, users have no sense of control, no idea of where the consultation is going, and no idea of why the information is needed. When questions jump from one context to another, users think the computer is scatterbrained. For example, would you trust a computer that asked you, in order, "What is your height? What is your weight? What is the capital of Kurdistan?"

We can ameliorate these problems by (1) entering data in related blocks with visual prompts, as when filling out a form, (2) ordering questions so that meaningful preliminary conclusions can be drawn and presented to users (for example, "That rules out such and such"), and (3) removing interrogative redundancy (if you ask a child's age, for instance, you should not subsequently ask if the child is past the age of puberty, or if the child is of preschool age). In the future, our expert-system course will deal more fully with such problems. But a companion course in human/machine interaction would be better. Expert systems should be designed so that users almost feel they are conversing with an intelligent human.

Finally, we found our students all too willing to join in uncritical enthusiasm for expert-system technology. We presented problems and criticism, and emphasized negative as well as positive points of view—but we will pursue this approach more systematically next time. We plan to assign and discuss such readings as articles by Parnas[12] and by Dreyfus and Dreyfus.[13]

Specific suggestions about M.1

If you want users to choose menu items, then identify those items with letters instead of numbers because M.1 treats numbers strangely; in addition, many experienced typists enter the lowercase letter "l" instead of the numeral "1."

Your system will be more user friendly if your last menu entry is "none of the above." Although this will create more work for knowledge engineers, it will accommodate inadvertent menu displays.

The consultation's beginning will probably give users instructions that will then disappear forever. It might be nice to provide an optional review of these instructions, perhaps

A list of the systems our students produced.

Program	Commentary
autism	Help a psychiatrist diagnose autistic children.
autism2	Help a neurologist diagnose autistic children.
AUTIS	Help a special education field worker diagnose autistic children.
ESIAC	Help diagnose autistic children based on speech behavior.
stutter	Detect disfluent (stuttering) children and suggest prognosis.
ANES	Aid an anesthesiologist during surgery.
Chromie	Congenital chromosomal defect diagnosis system.
Cogito	Help install 4.2BSD Unix on a VAX computer or add new devices (this project was expanded into a masters thesis and is a useful expert system; if you would like to use the system, please contact the authors).
diagnosis	Discover cause of failure for RS232 terminal/computer interface (written in OPS5).
labdes	Help design a PC laboratory.
invest	Help develop an individual investment portfolio.
solar	Help design solar-energy home-remodeling plans.
STATCON	Aid a biomedical engineer in selecting BMDP statistical-analysis programs.
Rockbolt	Help design rockbolt support systems for coal mines.
plans	Process planner and operations scheduler for a machine shop.
schedule	Find the best scheduling rule for a job shop.
advice	Help SIE graduate students formulate study plans (a routine, but very complicated problem; this expert system used M.1, C, C_to_dBase hooks, and dBaseII).
major	Help incoming freshmen choose a major (unsuitable for M.1 because it was just a question-and-answer session).
Diplomacy	Help a person to play the game of Diplomacy.
backgammon	Offer advice about the best move for backgammon (unsuitable for M.1 because it primarily involved numerical computation, not symbolic manipulation).
EXSYS	Identifies problems where an expert system is appropriate (unsuitable for an expert system because there is no human expert who does this much better than everyone else).
bid	Select correct bid in a bridge game (unsuitable for M.1 because it was a simple table lookup).

by labeling the banner knowledge base entry of your end-user system as follows:

```
instructions:
configuration(banner)=
['Welcome to your M.1 advisor.
Type "list instructions." to get this message.
Type "help." to get M.1"s help message.
Remember to end all your answers with a period and a
return.
Good luck.',nl].
```

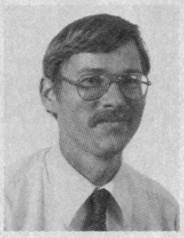
When the semester began, we were not sure what we would teach in this experimental course on expert systems; however, the course evolved nicely. We ended up using the MIT video tapes for nine hours, and the Teknowledge M.1 instructor notes for 10 hours. Guest lectures occupied four hours, Forsyth's book nine hours, Harmon and King's book five hours, and other material took eight hours. Although it seems a hodgepodge, everything fit together well and (even if we say so ourselves) it proved satisfactory. Our students seemed to agree—they rated the course as "very good" on the CIEQ course evaluation questionnaires. E

Acknowledgement

Grants from Bell Communications Research and the AT&T Foundation supported this course and related research.

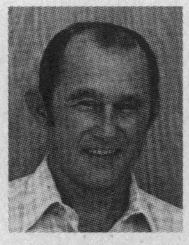
References

1. R. Davis, "Expert Systems" videotapes, MIT, Cambridge, Mass., 1984.
2. R. Forsyth, ed., *Expert Systems: Principles and Case Studies,* Chapman and Hall, New York, N.Y., 1984.
3. P. Harmon and D. King, *Expert Systems: Artificial Intelligence in Business,* John Wiley and Sons, New York, N.Y., 1985.
4. E. Tello, "Knowledge Systems for the IBM PC," *Computer Language,* July 1985, pp. 71-83, and Aug. 1985, pp. 87-102.
5. J. Goldberg, "Experts on Call," *PC World,* Sept. 1985, pp. 192-201.
6. *Expert Systems,* Vol. 2, 1985, pp. 188-265.
7. B.G. Buchanan and E.H. Shortliffe, *Rule-Based Expert Systems,* Addison-Wesley, Reading, Mass., 1984.
8. F. Hayes-Roth, D.A. Waterman, and D.B. Lenat, *Building Expert Systems,* Addison-Wesley, Reading, Mass., 1983.
9. L. Brownston, R. Farrell, and E. Kant, *Programming Expert Systems in OPS5,* Addison-Wesley, Reading, Mass., 1985.
10. B. Thompson and W. Thompson, *Microexpert,* McGraw-Hill, New York, N.Y., 1985.
11. M.J. Horvath, C.E. Kass, and W.R. Ferrell, "An Example of the Use of Fuzzy-Set Concepts in Modeling Learning Disability," *Am. Educational Research J.,* Vol. 17, No. 3, 1980, pp. 309-324.
12. D.L. Parnas, "Software Aspects of Strategic Defense Systems," *Am. Scientist,* Vol. 73, No. 5, Sept.-Oct. 1985, pp. 432-449.
13. H. Dreyfus and S. Dreyfus, "Why Computers May Never Think Like People," *Technology Rev.,* Vol. 89, 1986, pp. 42-61.



A. Terry Bahill is a professor of systems and industrial engineering at the University of Arizona at Tucson. His research interests include control theory, modeling physiological systems, head and eye coordination of baseball players, expert systems, and computer text and data processing. He is the author of *Bioengineering: Biomedical, Medical, and Clinical Engineering* (Prentice-Hall, 1981). He received his BS from the University of Arizona and his MS from San Jose State University in electrical engineering, and his PhD in electrical engineering and computer science from the University of California at Berkeley.

In addition to being on the *IEEE Expert* editorial board, Bahill is a member of several IEEE societies including Engineering in Medicine and Biology, Automatic Controls, Professional Communications, and Systems, Man, and Cybernetics. He was vice president for publications, and is now vice president for meetings and conferences and an associate editor for the Systems, Man, and Cybernetics Society. He is a member of Tau Beta Pi, Sigma Xi, and Psi Chi.



William R. Ferrell, professor of systems and industrial engineering at the University of Arizona, teaches courses in human factors and in mathematical modeling of human performance. He earned his BA with honors in English literature at Swarthmore College. He studied mechanical engineering at MIT where he earned his SB (also with honors), his SM, ME, and PhD. For four years during that period he worked in machine and product design at Polaroid. From 1962 to 1969, he taught at MIT where he was ultimately associate professor and codirector of the Man-Machine Systems Laboratory in the Mechanical Engineering Department.

His research has covered a wide range of human performance: driver behavior, aids for the handicapped, remote manipulation, robotics, human information processing, subjective judgment, and expert systems. He has more than 30 publications in such areas and, with T.B. Sherman, is author of *Man-Machine Systems: Information, Control, and Decision Models of Human Performance* (MIT Press).

Ferrell is a member of the administrative committee of the IEEE Systems, Man, and Cybernetics Society, a fellow of the Human Factors Society, a founder and currently a director of its Arizona chapter, and a member of Sigma Xi.

The authors' address is the Systems and Industrial Engineering Dept., University of Arizona, Tucson, AZ 85721.