6 AN OVERVIEW OF EXPERT SYSTEMS

By Bruce G. Buchanan

As computers become smaller and more powerful, smart managers are asking, "What can I do with them?" and—even more to the point—"What can they do for the company's bottom line?" The nature of the business doesn't matter; in every business computers have made numerous changes in record keeping, process control, and decision-making. And there will be more.

One of the most important trends in computing is making computers behave intelligently. The software underneath this intelligent behavior is called an *expert system*, sometimes also called a *knowledge-based system*, or *knowledge system*.

An expert system is a computer program that reasons about a problem in much the same way, and with about the same performance, as specialists.

This chapter is about the trend toward using expert systems: what it means, how it's possible, and how to think about it. There have been lead articles about this in *Fortune*, *Business Week*, and *Newsweek;* most Fortune-500 companies are using expert systems; many are establishing research and development groups for them; even staid IBM is marketing expert systems tools—and using them internally.

There are many reasons why companies want to build an expert system. Most of them are based on the premise that:

Expertise is a scarce resource.

And the corollary (by Murphy's Law):
Even when there is enough expertise, it is never close enough to the person who needs it in a hurry.

Because this is true, almost by definition, an expert system containing some of the knowledge of a company's specialists may have several benefits.

There are several examples of expert systems working in various problem areas. These span many activities, including finance, manufacturing, engineering, medicine, the military, and agriculture. At present, they are used more as "intelligent assistants" than as replacements for technicians or experts. That is, they help people think through difficult problems and may provide suggestions about what to do, without taking over every aspect of the task. Although the problems are quite different they can be categorized into two major classes—problems of interpretation and problems of construction.

Interpretive problem examples include Schlumberger's *Dipmeter Advisor*, which replicates the expertise of some of their company-wide specialists who interpret data from clients' oil wells and then sell the expert system's interpretations around the world. Another example is IBM's *YES/MVS System*, which realizes considerable cost savings in the manufacture of storage devices in their San Jose plant by using an expert system to help with the final checkout of equipment; errors in decisions about the ability to ship the equipment have been cut in half.

Examples of problems of construction include the *Motor Expert* program developed by the Delco division of General Motors, which saves the time of specialists designing springs and brushes for one-of-a-kind electric motors; a job that formerly took 1 to 4 weeks is now done in 15 minutes. Digital Equipment Corp. (DEC) developed the *XCON* expert system to assist less experienced sales personnel with the refinement and configuration of a client's computer order to get it shipped out right the first time. (Part of the expert system has been used to configure more than 100,000 orders for VAX computers.)

In all of these systems computers are using knowledge and reasoning in much the same way as good decision makers do. Forty to 50 years ago computers began to change the nature of science and engineering by manipulating numbers; 30 to 40 years ago they began to change the nature of business by manipulating textual data. The 1970s and 1980s will be seen as the time computers began to transform the nature of decision making everywhere by manipulating and using knowledge.

Computer methods for reasoning about these problems are taken from artificial intelligence (AI) and combine some well-known strengths of machines with some well-known strengths of human minds. Computers are better than people, for example, following long, complicated sets of instructions repetitively, without tiring and without introducing errors. Even highly trained physicians make errors when deprived of sleep or when faced with bewildering combinations of symptoms that don't fit neatly into a well-known pattern. People are still better than computers in many ways, on the other hand, in being creative and flexible in novel situations. One strength of our minds, however, that can be partly captured in expert systems is our ability to use many different elements of knowledge that are not all numeric and that are not all definitional, as in financial investment or medical diagnosis, for example. In performing these tasks, experts and expert systems use judgmental rules gleaned from years of experience, which traditional computer programs do not.

WHAT IS AN EXPERT SYSTEM?

An expert system is a computer program with expert-level problem-solving abilities. We expect more than the right answer from specialists we hire, however, so we also include additional criteria in our characterization of an expert system. An expert system has the following design characteristics:

GOAL: Achieves expert-level performance
DESIGN: Uses reasoning, not just calculation
Explains its line of reasoning
Retains flexibility

One well-known expert system that has become a classic, although

not actively used, is MYCIN. It was developed at Stanford University by E.H. Shortliffe and others in the mid-1970s. Its task is twofold: (a) diagnose the cause(s) of infection in a patient and (b) recommend appropriate drug therapy. (From a medical perspective, MYCIN's knowledge base is now dated; from the perspective of expert systems it represents much of the kind of reasoning that is captured in today's systems.) MYCIN's conclusions were demonstrated to be equal in quality to those of infectious disease specialists at Stanford Medical Center. A sample typescript in Appendix I (in reference section) illustrates MYCIN's requesting information about a case and reasoning to conclusions about the best treatment. Current expert systems rely much more than MYCIN on interactive graphics, but are otherwise very similar.

PERFORMANCE

Naturally we want computer programs to solve problems without error, but that is not always possible. In fact, outside of mathematics and logic, we don't have flawless methods we can put into programs. Specialists in business, medicine, engineering, science, education, the military—every area outside of pure mathematics—must solve problems with less than perfect methods. How do they do it? Mostly by building up specialized knowledge through extra years of training and experience and by reasoning carefully with that knowledge in situations they have learned to recognize. They're not infallible, though. Specialists' decisions are challenged frequently, most noticeably in the courts. So it is also unreasonable to expect computer programs to reason infallibly in all of these areas.

REASONING

When we say that expert systems are reasoning—and not just calculating with numbers—we are saying that they belong to a class of programs using the methods of *Artificial Intelligence* (AI). In the 1940s, during World War II, computers were used almost exclusively for large mathematical problems. At Los Alamos, for instance, scientists had to solve complex mathematical equations in order to calculate elements in the design of the hydrogen bomb. These applications are usually referred to as large-scale scientific computation, or "number crunching." In the 1950s, IBM and other computer manufacturers

realized the enormous value in helping business solve problems of record keeping, payroll, and the like. These applications extended the concept of computer-as-calculator to computer-as-data-manager. They became known as business data processing applications.

In both of these classes of applications, the method of computation is error-free. There is no question that the result is correct, providing of course that the computer has been programmed correctly and the initial information given to it is correct. A mathematical equation is solved correctly; an employee roster is sorted correctly—if the methods are followed precisely. And computers are better able to follow complex instructions than people are. In computer science, logic, and mathematics, we call these error-free procedures *algorithms*. They are procedures that can be guaranteed to provide a correct answer in a finite time, if there is one, and otherwise will provide a statement that the problem is not solvable.

Some algorithms are too expensive to use, however, even in computers. A classic example is finding the shortest route a travelling salesman can take to visit many cities at once and end up at home. With more than a handful of cities, algorithmic methods will not finish in time to be useful. For this reason, alternative methods have been developed.

Around the mid-1950s and early 1960s an alternative style of computing came to be recognized as important. Instead of always using algorithms, a computer may use *heuristics*—rules of thumb that aid

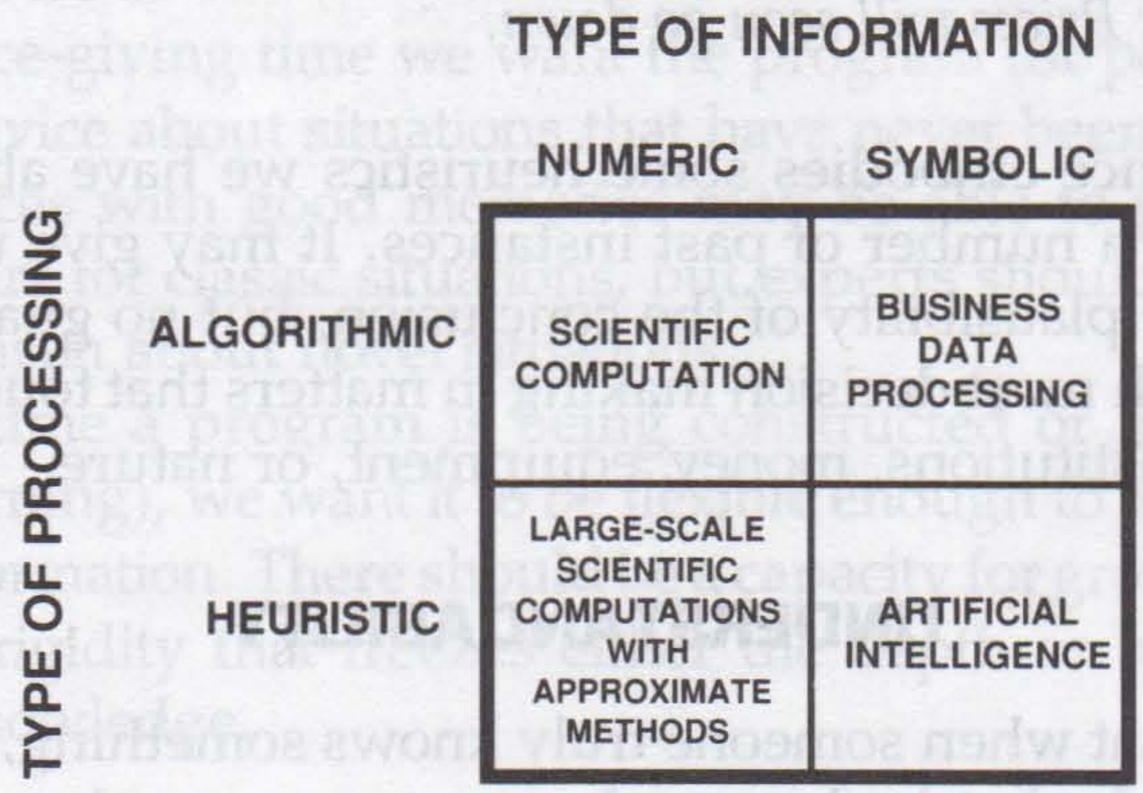

Figure 6-1. Two dimensions of computer software that distinguish artificial intelligence programs from other styles of computation.

in finding plausible answers quickly without guaranteeing the correctness of the results. Sometimes these rules of thumb are introduced into large numerical simulations in order to get the simulations to crank out answers more quickly. Or approximate methods may be substituted for more precise ones for the same reason. The assumptions may not all be correct; thus the results of the simulation may not be correct.

When *heuristic* (non-algorithmic) methods are combined with *symbolic* (non-numeric) data, we are dealing with that part of computer science known as artificial intelligence. This is summarized in Figure 1.

This is what we mean when we say that a computer may reason symbolically. A simple example, after Aristotle, is:

Every computer company is a corporation.
IBM is a computer company.
Therefore: IBM is a corporation.

This syllogism doesn't require numbers, only symbolic names of things. In this case, drawing conclusions about one man based on properties of every man is valid. In other cases, our inferences are plausible, even if not logically flawless. For example:

Stock market prices generally move in cycles.
The market is at an all-time high.
Therefore: Prices will soon go down.

This inference embodies some heuristics we have about noticing similarities in a number of past instances. It may give us some confidence in the plausibility of the conclusion, but no guarantee. Such is the case with most decision making in matters that touch on human lives, social institutions, money, equipment, or nature.

UNDERSTANDABILITY

Plato noted that when someone truly knows something, he can "give an account" of what he knows. In our terms, good performance is not enough to call a person (or program) an expert: he/she (it) should also be able to explain why the solution is plausible, what features of

the situation were noted to be important, what knowledge and problem solving methods were used. We label winners who can't explain their reasoning "incredibly lucky," and generally avoid betting the mortgage on their advice. Each field has its own standards of what a reasonable explanation is. A surgeon who recommends amputation of a leg generally talks about the process of disease or extent of injury and what will happen if it is not amputated. A broker who advises liquidation of one's stock portfolio may appeal to technical charts, historical trends, or some economic principles that point to a stock market collapse. In their own communities, both the surgeon and the broker can usually justify, in court if necessary, the advice they give. And we regard them as experts partly because they have the knowledge that lets them do this.

We don't often pay expert-level fees to someone whose lucky performance can only be attributed to good intuition. Nor would you let a surgeon amputate your leg if the only "reason" was intuition—that's a time for a second opinion.

FLEXIBILITY

We expect experts to be flexible in their thinking. And we regard persons as amateurs, or worse, when we discover that their opinions are rigid, that they have locked in single ways of dealing with problems, or they are unable to deal with new situations.

In particular, there are two situations in which we want expert systems to be flexible:

1. At advice-giving time we want the program (or person) to provide good advice about situations that have never been encountered before. Novices with good memories may be able to provide "textbook" answers for classic situations, but experts should, in addition, be able to reason about novel situations.

2. At the time a program is being constructed or modified (or a person is learning), we want it to be flexible enough to assimilate new bodies of information. There should be a capacity for growth of knowledge, not a rigidity that freezes either the depth or breadth of the program's knowledge.

KEY CONCEPTS

The principles—performance, reasoning, understandability, and

flexibility—that more or less define expert systems can be achieved with a few key methodological ideas in their design and programming. In this section, the key ideas will be introduced; in successive sections they will be elaborated so as to explain how they basically work. The main organizational principle of expert systems is to keep specialized knowledge separate from the logical and heuristic inference methods that use it. This is easy to say but difficult to follow, for reasons that will be described later.

Key Idea #1

Keep Specialized Knowledge About the Problem
Separate from General Reasoning Methods

For example, a broker's strategy for—or method of reasoning about—a widow's portfolio can be separated from specific knowledge about individual stocks. In retrospect, Key Idea #1 seems pretty simple to advance as the main idea, but that doesn't diminish its value; it only makes it easier to understand.

Key Idea #2

Keep independent pieces of knowledge independent.
Keep the rest as nearly independent as possible.

Modularity at the level of knowledge about the problem area implies conceptual separation of elements in the knowledge base. For example, medical knowledge about penicillin, although not totally independent, can often be separated from knowledge of other drugs. Knowledge of penicillin can be modified in major ways, or deleted, without altering the program's knowledge of other drugs. This is to say that the concepts used to talk about other objects in the domain should be chosen so as to allow talking separately about an individual object, a single property of an object, or a single relation of one type of object with another. Within the program itself, good programmers have many ways of keeping distinct parts of a program from interacting.

Key Idea #3

Strive for uniformity of language—
at both the conceptual and programming levels.

The underlying intuition of the third key concept is that it is easier for a person or a program to build, understand, and modify a body

of knowledge if it doesn't mix and merge a variety of different types of things. This is as true at the knowledge level as the programming level. For instance, one of the most compelling aspects of Newton's Laws is that all physical bodies are treated as quantities with mass. He didn't need one set of laws for planets and another for apples. So it is desirable to build an expert system with a "conceptually clean," well-organized, simple collection of concepts. And it is important to use a simple, well-organized collection of programming constructs as well. Otherwise there are too many different kinds of things to keep track of and reason with. A spreadsheet, for example, can be constructed with many different constructs in its matrix, including numbers, text, and formulas. The more constructs the designers allow in these cells, the more complicated the whole spreadsheet program (and manual) becomes.

There is dispute among artificial intelligence specialists about this principle. There are good reasons to violate it, as we shall see, in the interest of being able to say more about the objects and relations of interest than can conveniently be said in a single language. We are frequently told by bilingual friends, for instance, that there are some concepts that just can't be expressed fully in English. The same is true for programming constructs, but the basic principle for constructing expert systems is to try to maintain uniformity as much as possible.

Key Idea #4

As much as possible, use the same vocabulary and methods in the program as the experts and practitioners use.

The fourth Key Idea is to design the expert system to mirror the ways experts think about problems in their domains. That means using the same terms and the same rules of reasoning the experts use. One reason for this is that building and debugging a knowledge base depends necessarily on the expert, and using less familiar terminology or methods will introduce confusion and error before the knowledge base is completed. Also, after it is completed, it needs to be comprehensible and unambiguous to the practitioners using the system or else confusion and error will result.

Note that we are assuming here that the expert designing the system knows how to make it understandable to users as well as to

himself/herself. Great care must be taken when building a system to insure that this assumption is true.

There are times when this principle will be, or should be, violated. For example, when efficient computer algorithms can solve part of a problem, it doesn't usually make good sense to use anything else for that part, even if the experts don't think about it in that way.

These key ideas help us achieve all of our four goals in the following ways.

PERFORMANCE—In problems whose solution methods are not already well formalized, which are considerable, much of the effort in building a knowledge base from an expert system lies in building the conceptual framework. Which properties and relations of objects to describe is often not well specified at the beginning. So the knowledge base is built incrementally, when experience with one knowledge base guides future modifications, extensions, or reformulations.

REASONING—When there is no pat formula for solving a problem and the solution methods are not well characterized, it is important to encode heuristics that experts say they use. Storing these separately and in a simple form allows them to be changed easily. Since it is nearly impossible for an expert to articulate a complete and consistent set of heuristics at one sitting, it must be easy to add, remove, or modify the heuristics that determine the reasoning.

UNDERSTANDABILITY—With modularity, individual elements of the knowledge base can be displayed meaningfully in isolation. Moreover, with the separation of knowledge base and general reasoning methods it is possible to peruse the knowledge base in order to find just those elements that were used to reason about a new case. And with uniformity of data structures, it is possible to build one set of procedures that produce explanations.

FLEXIBILITY—When the elements of the knowledge base are in separate data structures, and not intertwined with code for inference procedures, we can add more knowledge with considerably more ease. By keeping the individual items in the knowledge base nearly separate, we have fewer interactions to worry about when we change an item. And by keeping the representation homogeneous, we can more easily write other programs that act as "editing assistants" or explainers that help us insure correctness of new items and help us understand what is in the knowledge base.

WHAT ARE EXPERT SYSTEMS GOOD FOR?

In general, expert systems can reduce costs or increase quality of goods and services. In a single phrase, they can increase productivity in an organization. If you believe either that there is not enough expertise in the world, or that it is not well distributed, then you are ready to consider the idea that putting human expertise into an easily replicated form may answer some productivity problems. Consider medical diagnosis. Specialists at university medical centers generally see more of the unusual disorders than a rural practitioner and thus stand a better chance of diagnosing them correctly. Putting some of that expertise more directly at the service of the rural practitioner could allow more effective treatment, and save patients the time and trouble of travel to the medical center.

Specialists with the most field experience are often the ones promoted to desk jobs in the central office. When subtle combinations of causes keep a less experienced field service technician from fixing a mechanical failure, you can guess who gets called out to fix it. Depending on travel time and the critical quality of the work flow in the central office, calling the experienced specialist out may be a very expensive repair procedure.

The following situations are all cases where it may make good sense to build an expert system:

1. Too few specialists for the number of problems
2. Specialists not at the sites of problems when they occur
3. Long training time for a specialist
4. High turnover among technicians
5. Combination of complex equipment and poorly trained technicians
6. Organization's best (or only) specialist in an area is nearing retirement
7. Too many factors for a person to think through carefully in the time available.

HOW DO EXPERT SYSTEMS WORK?

Expert systems constitute one class of computer programs. As such, they work the same way as every other program: they process input data to produce output data. But the nature of the processing is

different from most conventional programs. The key ideas mentioned earlier are the key differences in the design and implementation of expert systems.

In order to design a reasoning program, we need to provide knowledge to reason with and reasoning methods to use. To paraphrase a Japanese proverb: Knowledge without reasoning ability is like a load of books on the back of an ass. Over the last few decades, research in artificial intelligence has elucidated programming methods for making inferences and storing knowledge. We provide an illustration of both of these topics below, although with some reservations about oversimplifying. (See Figure 2.)

Inference Methods

Aristotle's theory of the syllogism defined acceptable inference methods outside of mathematics for about 2000 years. His theory has been extended in this century by Russell and Whitehead, and others, in a

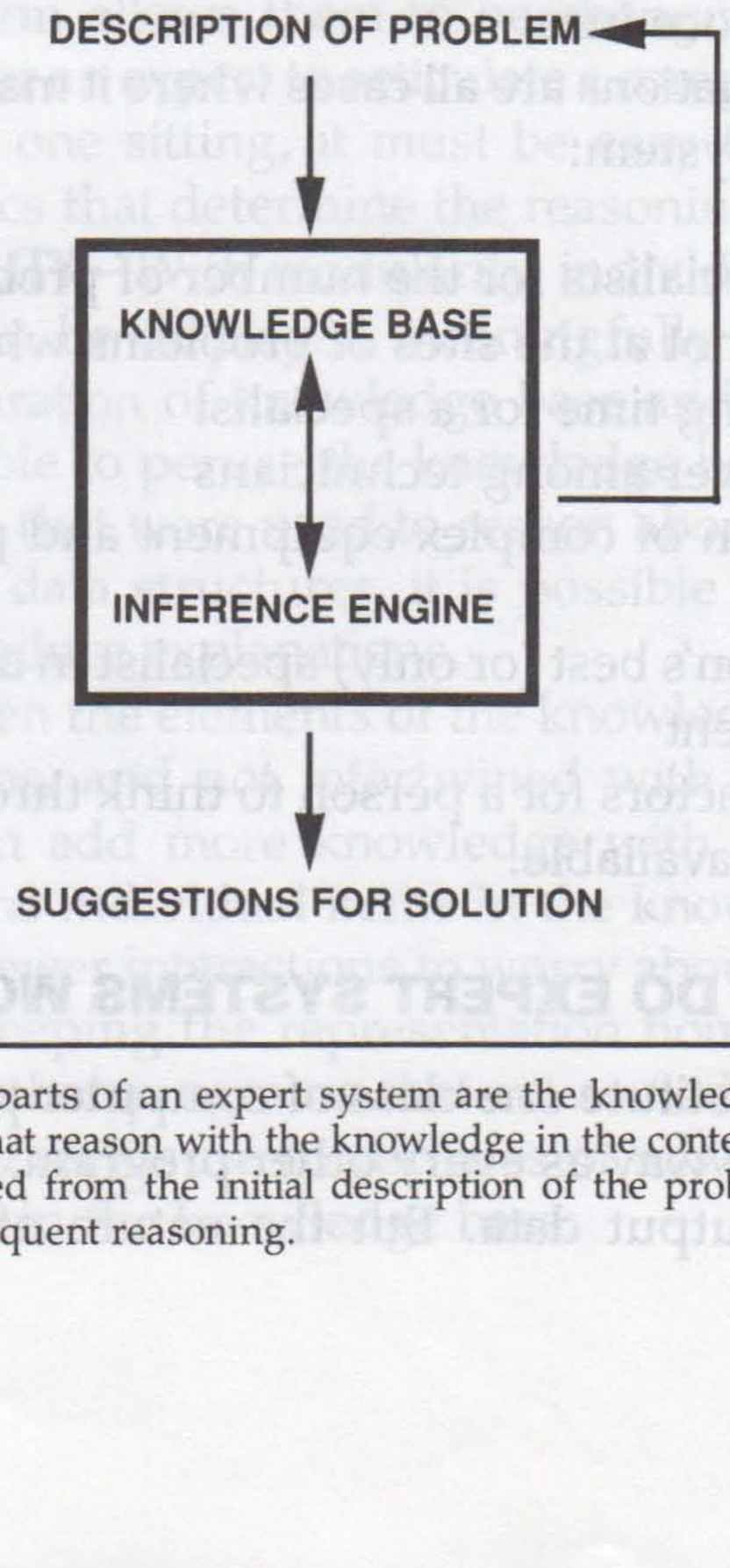

Figure 6-2. The two major parts of an expert system are the knowledge base and the inference methods (the "engine") that reason with the knowledge in the context of a particular problem. New facts may be inferred from the initial description of the problem and may themselves become the basis of subsequent reasoning.

formal theory that includes methods of reasoning with several statements and several variables in an argument. For example, we can infer that IBM has a corporate attorney, even without knowing who it is, from:

Every computer company is a corporation;
IBM is a computer company;
Every corporation has at least one corporate attorney;
Therefore IBM has at least one corporate attorney.

Formal logic defines several inference rules which are guaranteed to create true conclusions if the premises of the argument are true. The chain rule, for example, allows us to chain together a string of inferences:

If A then B;
If B then C;
If C then D;
therefore D.

Many of the inferences we make in our lives are not guaranteed by the rules of logic, however; nor do we have certain knowledge about the truth of our premises. Whenever we argue that the future will be like the past, as in stock market predictions, we have to be prepared for exceptions. These inferences, labeled "plausible inferences" by George Polya, are the ones of most interest in AI.

One set of programming methods used in AI for making plausible inferences is to assert the facts categorically—as if they were known to be true with certainty—and then reason about exceptions that might force revisions to the conclusion. For example, in concluding that IBM has a corporate attorney, we stated categorically that every business has at least one. To be more accurate, we should have noted some exceptions such as kids' money-making operations, small businesses, and so forth. By explicitly noting exceptions to this use of the term "business," we can continue to reason categorically without probabilistic qualifications.

Another set of methods deals explicitly with the degrees of uncertainty in the facts and in the associations. An example of this style of reasoning is:

Utility stocks are usually conservative (90%);
Widows often want or need conservative investments (80%);
Therefore: there is some evidence that Mrs. Jones should invest in utilities (70%).

Usually the degrees of uncertainty implied by words like "often" and "may" are expressed as numbers. And often these numbers are interpreted as probabilities.

A third, and most powerful, set of methods is to introduce heuristics, or rules of plausible inference, into the reasoning. They often produce satisfactory answers more quickly than their algorithmic counterparts produce the very best answer. When the algorithms entail exhaustively searching billions of possibilities, and require years of computer time, most of us will settle for less than the very best. (Also not all problems have algorithmic solution methods at all.)

Some general rules of plausible inference used in artificial intelligence programs are as follows:

"When you hear hoofbeats don't expect zebras."
That is, in normal situations expect the most common things.

"Be selective in what you consider."
Drop from consideration whole classes of solutions, without examining subclasses or individual solutions explicitly, if there are several other classes that look more promising.

"Don't complicate things unnecessarily."
If a problem can be explained by one cause, stop before considering multiple causes. (Note that this heuristic, called Occam's Razor, dates from the 14th century and is a guiding principle of science.)

In principle, the rules of inference, both logical and plausible, may be applied again and again to describing a situation, in any order, and the resulting conclusions will be the same. This is not always possible in practice, however. There may not be enough time to reason exhaustively about all possibilities and contingencies. For that reason artificial intelligence researchers talk about controlling the inferences

as being a more important, and more difficult, problem than making the inferences.

Controlling inferences breaks down into two subtasks: (a) deciding which knowledge to apply now, at this stage of the problem-solving process, and (b) deciding which part of the problem to work on now. Since we believe these subtasks require some intelligence, all of the principles for building knowledge-based systems also apply at this level of reasoning. In particular, it is desirable to make this control knowledge explicit and separate from the inference methods. An example is shown in Figure 3.

KNOWLEDGE ABOUT JIGSAW PUZZLES

Knowledge About Puzzle Pieces

P1. Rectangular puzzles have 4 corner pieces.
P2. Corner pieces have at least 2 straight sides at a right angle (and usually only 2).
P3. Corner pieces are joined by one or more edge pieces (usually many).
P4. Edge pieces have at least one straight side (and usually only one).
P5. Pieces with an unusual color probably join together.
P6. Sky and water are usually shades of blue.
P7. Sky pieces go at the top (unless reflections).

Knowledge About Control

C1. Work on corners first.
C2. If the pieces already placed create an unusually shaped hole, look for the piece that fits.
C3. Leave the sky pieces for your mother-in-law.

Figure 6-3. Knowledge about jigsaw puzzles can be separated into knowledge about puzzle pieces themselves and knowledge about the strategy of putting together the pieces. Strategies control the sequence of actions and thus are sometimes called control rules. In this example, control rule C1 directs a puzzle solver's attention to rules P1-P3.

Representation of Knowledge

We have said that a key idea in building expert systems is storing knowledge separately from the inference methods. Another idea is to avoid, as much as possible, representing it in a low-level computer language. But we have not said how to represent for the computer what an expert wants to tell it. English is too difficult for a computer to interpret unambiguously; FORTRAN and BASIC are too low-level for an expert to deal with efficiently. Clearly we need some stylized representations that are somewhere in between.

Several different representation methods have been developed in the last couple of decades. There is no single one that is best in every case; each has strengths and weaknesses. One of the fundamental trade-offs in thinking about the representation of knowledge is between simplicity and expressive power. We want a simple set of conventions for storing knowledge because that makes it easier for a person or a program to understand what is in the knowledge base at any moment. It is also easier to write simple statements without error. Aristotelian logic ("All As are Bs," etc.) and arithmetic are pretty simple representations. The difficulty, however, is that they lack the expressive power to let us say everything we think is important about a problem. A hundred years ago DeMorgan noted the lack of expressive power in Aristotelian logic (and a weakness in its inference methods); if you know that all horses are animals, he said, you cannot prove that the head of a horse is the head of an animal. This sort of problem led Russell and Whitehead to develop a formalism with more expressive power.

There are two major classes of representation methods, reflecting two different ways of viewing the world: *action-centered* or *object-centered*. Different problem areas may focus on one or the other, or different experts in the same problem area may. For example, physicians talk about disease and classes of diseases as entities with expected properties and also about clinically relevant actions that determine what to do—e.g., asking questions, measuring things, relating signs and symptoms to possible causes, matching likely causes to acceptable therapies. Neither point of view is wrong, but each focuses on medical phenomena quite differently. An expert system would similarly have one focus or the other.

Action-centered representations focus on conclusions that can be drawn from facts or, more generally, on relations between situations and actions. The formalism of mathematical logic is one popular choice for representing the actions of reaching a conclusion. Another popular formalism is "If-Then" rules. "If-Then" rules encode knowledge of what to do—what action to take—when certain conditions obtain. The action statement may be to sell a stock, turn a valve, add a piece to a design, or conclude that some other condition must obtain. The premise conditions in the "if" part of a rule may describe features of data, events that may have happened, parts of a situation, or facts that may be true.

Object-centered representations focus on the organization of objects in the world, for instance into hierarchies. They still allow conclusions to be drawn when an object is found to have some properties, but those inferences are triggered from "within" an object rather than from outside. That means that objects and their properties, and changes to any of them, drive the inferences. In an action-centered model, on the other hand, the inference rules drive the creation of new objects and properties. The net effect may be identical, as we said, but the way one thinks about the domain of discourse is different.

Also, in object-centered representations there is more machinery for saving storage space by using hierarchies. Properties of classes of objects, for example, may be associated with the class name and implicitly inherited by all of the instances without having to store them with each instance. The manager of a group is the manager of each person in the group, so the program only needs to store (once for each group) the name of the group manager and can use that, plus the class-instance hierarchy, to find the name of any individual's manager.

There are as many different conventions for representing knowledge as there are researchers working on this topic. This can be confusing when reading the literature. But they are basically all variations—usually mixtures—of the two different styles just discussed. By paying careful attention to the form in which knowledge about the world is represented, programmers can greatly facilitate the construction and maintenance of expert systems. No single set of conventions is right for all applications, so one must always be mindful of the design considerations and the trade-offs mentioned above.

HOW ARE EXPERT SYSTEMS BUILT?

Building an expert system requires finding out how an expert solves a problem and translating that expertise into a stylized form that can be read by a computer. This is no different in principle from building a conventional program in which programmers find out what equations or algorithms experts use and then write FORTRAN or COBOL programs that embody those procedures. The main difference in practice is that expert systems must incorporate knowledge that is much more qualitative and judgmental, and much more subject to

change. In fact, much of the time the expert's "know-how" is not yet written down (e.g., when what he/she does is regarded as an art) and the process of building the expert system codifies this "know-how" for the first time.

Because the expert's knowledge is often not already codified, building an expert system requires patience. It generally works best as a team effort involving one or more experts and one or more so-called knowledge engineers. A knowledge engineer is a programmer of knowledge-based systems who understands the conventions of the computing framework and who assists the expert in mapping judgmental knowledge into that framework. The dialogue between expert and knowledge engineer is often called *knowledge engineering*.

One of the key ideas in knowledge engineering is to focus on case studies. It is much easier for any of us to tell someone how we would approach a specific situation than to say in general terms how we solve problems of a type. Of course, if we have a set method (a "recipe" or a "canned procedure") that we always use, we can describe it. "Oh yes, I always use the French variation of the Alekhine-Gorbachev wave theory in situations like that," you might say. But then the knowledge engineer wants to know what you do next and—more interestingly—when would you make exceptions to your set policy. And the best way for you to think about those things is to focus on cases. As long as problem-solving requires more than the application of set procedures, knowledge engineers will need to go through many cases, and variations on them to help codify the expert's judgmental expertise.

Tools to Aid in the Construction of Expert Systems

Just as carpenters can construct houses faster with the right tools, knowledge engineers can build expert systems faster with software tools that boost their productivity. These come in several forms. The main idea, however, is to provide programmers with mechanized intelligent assistants that know about programming conventions, including abbreviations and shortcuts, that can help locate and fix errors, that can display the contents and interrelationships in a program or knowledge base, and so forth. These are the kinds of extra capabilities that distinguish system-building environments from programming languages.

Expert system-building environments (sometimes called "shells")

have been commercially available since about 1980. The powerful ones tend to be expensive (several thousands of dollars). But here, as elsewhere, you pretty much get what you pay for. There are many inexpensive languages ($100–$500) in which it is possible to build expert systems, but with about the same amount of help a carpenter gets from working only with 2 × 4's. One characteristic of a shell is its commitment to a set of representation conventions of the sort outlined above.

Steps Involved in Knowledge Engineering

It may take months or years to build an expert system, with the time depending largely on the complexity of the problem and the extent to which expertise is already codified. One reason it takes so long is that there are many steps involved. And at each step, the knowledge engineer or the expert may decide it is necessary to undo some results of previous steps.

The major steps involved in building an expert system follow. Many factors influence the amount of time involved in each step and the extent to which single steps have to be done more than once. For example, defining the precise nature of the problem to be solved often takes considerably more time than is expected at the outset, and often requires more than one refinement after a small prototype system is constructed.

Problem Assessment —what is the problem to be solved, how important is it, where are the example problems, who are the members of the team, in what setting will the final system be used, who are the intended users, etc.?

Prototyping —what is the correct level of detail, what terminology should be used, what representation of knowledge is appropriate, which shell system is appropriate, what cases should the first prototype be able to handle, what are the preliminary results of the first implementation, etc.?

Development —what are the difficulties encountered in the prototype and how should they be overcome, what knowledge is needed to extend the scope of the prototype to the full range of problems the

system should solve, how can the system respond to problems most efficiently, etc.?

Testing —how well does the system perform on test cases, how well in controlled field use, how well in the field, what performance measures are most meaningful to the company—money, time quality, job satisfaction, availability of information, etc.?

Fielding —how does the system couple to measurement devices and other computers in the fielded environment, what are the human engineering refinements that would make the system easier to use, what software engineering is needed to make the system an integral part of the environment, etc.?

Learning

At present, expert systems do not learn from experience. This is a defect that many research groups are working to remedy. Early prototypes of learning systems promise some automated assistance in maintaining and extending a knowledge base through the experience of routine use, but these are not yet available.

It is possible, however, to learn an initial set of rules from a case library (collected past experience) and use it for classification problems. Induction programs are being used to build simple rule sets for expert systems in which there is little chaining of the rules and little use of uncertain inferences. Current research may well extend the scope of these induction programs to more complex rule sets.

What are the Costs?

The major cost involved in building an expert system is in personnel time. Shell systems now run on most common computers, so it is not necessary to buy new equipment and, most importantly, it is not necessary to build the complete set of programming tools found in a shell.

Purchasing the shell and some training in how to use it are recommended. The amount of time needed from a team of experts and knowledge engineers is variable—as are their salaries.

A problem needs to be precisely defined before beginning; a case library of at least a half-dozen typical and hard cases needs to be assembled; a commercial shell running on an available computer is

recommended; and the expert(s) and knowledgeable engineers need to be identified and made available. Since the team's primary responsibility is this activity, they should secure the blessing of their management.

One of the main factors that determines the length of time a project will take, not surprisingly, is the nature of the problem. This includes both the scope of the problem and the extent to which a commercially available shell is appropriate for it. Another main factor is the definition of the "deliverable," that is, the terms of the contractual agreement specifying whether the product delivered is a prototype or a smoothly polished software package. There is a hidden cost, which is the cost of covering for the expert while he/she is out of service. When this person is one of the best in the business, the same level of coverage may be difficult to obtain. That fact, in turn, may result in somewhat less efficient operations—possibly a higher rate of equipment failure or increased downtime—in the expert's sphere of influence in the company. And this uses up one of the company's scarcest resources—the patience of managers. So the direct costs don't tell the whole story.

There are added gains in building an expert system, however, that offset some of the indirect costs just mentioned. Besides the obvious gains showing up in profits, there are very noticeable gains in the quality of information available. We sometimes notice higher individual and group morale. An expert finds new challenges in articulating lessons learned over years of experience. Group managers feel they are making a direct contribution to ensure future productivity of the company. And corporate management is able to participate in innovative "high tech" changes without restructuring the organization or replacing capital equipment.

WHAT ARE THE TRENDS?

What we see now is just the beginning of a wave of intelligent software that can have as great an effect as business data processing software has had. It is impossible in any area of technology to make accurate predictions. However, there are many parallels between the growth of expert systems and computing hardware, with about a 25-to-30-year lag.

When electronic computers became available commercially, busi-

nessmen began to ask about applications that would make a difference to them. In 1955, several of these innovators assembled at Harvard to discuss the benefits of using electronic digital computers. Experience with expert systems some 30 years later leads us to similar conclusions. The following are conclusions they drew based on three ways in which automatic data processing allows managers to retain control and increase productivity:

"Produce information sooner."
"Produce more information from available data."
"Process paperwork cheaper, or with less human error or bias."

The parallels with expert systems are clear: we have just begun to explore the first set of applications and have only a limited view of possibilities in the future. Over the next decade, however, we do know what extensions in capabilities will be possible because they are the ones that are active research areas in the present. Which of these will be introduced is largely a function of perceived value outside of the research laboratory.

Very Large Knowledge Bases

To date, expert systems have used knowledge bases of modest size. Although size is difficult to define, most mention only a few thousand different facts and relations. Probably the largest is the INTERNIST (now QMR) medical diagnosis system in which about 250,000 facts are encoded. Some of this limit results from our own inability to keep in mind the interrelationships among more facts as much as from the technology of storing and retrieving them. Nevertheless, in the future we will have improved the technology to build and maintain knowledge bases of much larger scale.

Shared Knowledge Bases

Today's systems use single-knowledge bases that have been built specially for them. As more and more systems are constructed, however, it will be important to use knowledge bases in different contexts and then reuse one system's knowledge base in another system. It is wasteful, and not always necessary, to duplicate the contents of an old knowledge base in a new application.

Distributed Data Bases

Data bases exist now on many machines. Yet it is nearly impossible to treat several of them as if they were one logical unit. Expert systems also need this capability. Current research will allow much broader sharing of data among different data bases than is currently available in commercial systems.

Parallel Computation

Computers are fast, but never fast enough. In addition to the immense speedups from improvements in the hardware there are potential speedups from software. When a problem can be divided into nearly independent subproblems, it is conceptually easy to see that multiple computers could be used to solve the subproblems in parallel, thus saving considerable time. Work in the research laboratories indicates that this is feasible. Thus it will almost certainly become a commercial reality in the near future if it is cost-effective.

Real-Time Monitoring

As expert systems become faster, it will be easier to build systems that monitor other devices or processes with rapid changes. A difficult problem is managing time-dependent relations efficiently. This is one of the necessary components of a monitoring system.

Richer Input/Output

No one likes to interact with computers by typing. Considerable work on interactive graphics has reduced the need for typing. But it will be even easier when we can communicate with programs by giving voice commands and receiving spoken English output in return. Current research is promising.

Learning

Expert systems should learn from experience. That is, it should be easy to instruct them to avoid making the same stupid mistake twice. Prototypes of programs that learn already exist in research laboratories and will be integrated into future commercial systems.

CONCLUDING OBSERVATIONS

Expert systems already save companies millions of dollars. The number of applications of today's technology is nearly boundless; consider,

for example, the number of pieces of equipment in our lives that we don't know how to fix when they break.

The first commercial shells on the market are robust enough to be used effectively. Coupled with advances in electronics and telecommunications, expert systems have the potential for changing the quality of decision making throughout every business and professional organization and consequently across all of society. Just as electronic calculators made each of us more efficient at multiplication and division, expert systems can make each of us better at specialized reasoning tasks. The consequences will be profound.

They have their limitations, of course, as did Ford Model T's and early UNIVAC computers. But they are becoming as ubiquitous and as utilitarian as computers themselves. Future developments will make the technology even more broadly applicable and easier to use. Every professional activity—from medicine to manufacturing—contains problem-solving tasks that are not mathematical. The development of artificial intelligence methods that can solve many of these problems will change the nature of applications of computers in our society as much as the development of transistors changed the architecture and applicability of computers themselves.

What will be the consequences? From a business perspective, organizations will become more productive as more and more time-consuming tasks are performed more efficiently and more consistently. Every decision maker will have more knowledge at his/her fingertips, plus intelligent assistance in avoiding information overload. Students in every field will be able to pose hypothetical questions to the electronic equivalent of the world's best experts. Engineers and scientists will have intelligent assistants to do the routine data interpretation, calculations, and bookkeeping that take time away from creative thinking. Individual consumers can have do-it-yourself knowledge packaged for their own repair or hobby projects.

Everyone wins if organizations run more efficiently, people are healthier, students learn faster, scientists are more productive, and individuals can get expert help readily. Insofar as every aspect of our lives is becoming more complex, one can argue that we are headed for total collapse without more technological assistance. We need a smarter work force with more resources to call on just to maintain our current standard of living. On the other hand, policy makers must be mindful that some segments of society are not sharing in

today's benefits. And the disparity between segments will increase with further technological advances in the future unless strong policies favoring education, job retraining, and applications of technology to social problems are instituted.

BRIEF LESSONS IN HIGH TECHNOLOGY

Understanding the End of this Century to Capitalize on the Next

by

James D. Meindl

James S. Harris

John L. Hennessy

David R. Cheriton

Bruce G. Buchanan

Mark R. Cutkosky

Fouad A. Tobagi