Robinson, V. B., and A. U. Frank. "An Introduction to Expert Systems." Paper presented at the ASCE Specialty Conference on Integrated Geographic Information Systems: A Focal Point for Engineering Activities, 3-5 February, 1986 1986.

AN INTRODUCTION TO EXPERT SYSTEMS*

By Vincent B. Robinson[1] Andrew U. Frank[2] Matthew Blaze[3]

Abstract. Expert systems have recently received a great deal of attention and are likely to play important roles in future applications of computers to geographic information systems. This paper introduces the surveying engineer to expert systems and a subsequent paper assesses their potential contribution to integrated geographic information systems. Expert systems are computer systems that advise on or help solve real-world problems requiring an expert's interpretation. They solve real-world problems using a computer model of expert human reasoning. Facts and rules are part of what is known as the knowledge base. These can be represented using first order predicate logic, semantic nets, and frames. Exploitation of the knowledge base is controlled by the strategies of state space search and the use of pattern matching programs. Construction of expert systems is an expensive affair where knowledge acquisition plays a critcal role. One the major limitations on building expert systems is our current lack of understanding of human experts decisionmaking. However they provide a powerful vehicle for furthering our understanding of those processes and applying that understanding to the solving of engineering problems.

Introduction

Expert systems have received a great deal of attention in the professional literature (10), popular computing literature (26), and government agencies such as NASA (9, 16) have assessed the nature and potential contribution of expert systems in the accomplishment of many of their organizational tasks. Much of the attention accorded expert systems is a consequence of their representing the advancement of artificial intelligence to the point of accomplishing some practical results. Given current levels of interest, support and progress, expert systems are likely to become increasingly important in a number of areas of society. This paper briefly presents the definition and nature of an expert system. A subsequent paper critically assesses current efforts and suggests future trends in the developing relationship between expert systems and geographic information systems.

* A version of this paper was presented at the ASCE Specialty Conference on Integrated Geographic Information Systems: A Focal Point for Engineering Activities, February 3-5, 1986.

[1] Assoc. Prof., Dept. of Geology & Geography, Hunter College-CUNY, 695 Park Avenue, New York, NY 10021

[2] Assist. Prof., Dept. of Civil Engineering, Univ. of Maine at Orono, Orono, ME.

[3] Systems Programmer, The Rockefeller University, New York, NY.

Defining Expert Systems

Most definitions of expert systems agree on two major points. In general, expert systems are computer systems that advise on or help solve real-world problems requiring an expert's interpretation. They solve real-world problems using a computer model of expert human reasoning, reaching the same conclusions that the human expert would reach if faced with a comparable problem (28). They consist of two independent parts: One being a domain independent inference engine and the other being domain specific knowledge.

Other important qualifications often cited are that expert systems should interact with humans in appropriate ways, including the use of natural language, function despite some errors in the data and uncertain judgemental rules, contemplate multiple competing hypotheses simultaneously, explain why they are asking for additional information and finally, justify their conclusion.

Another approach to defining expert systems is to determine what they do not do. There is not absolute agreement on these limitations but it seems useful to highlight some of the limitations of expert systems. They do not not think as a human does, resort to reasoning from first principles, drawing analogies, or common sense (10).

Contemporary expert systems generally lack any but the crudest ability to learn. Development of powerful learning ability and its incorporation into the architecture of expert systems remains largely an area of intense research activity.

Organization of Expert Systems

Expert systems are characterized by a specific organization illustrated in Figure 1. Their organization differs from conventional computer programs. Ordinary computer programs organize knowledge on two levels as data and program. Most expert systems organize knowledge on the three levels of data(facts), rule base, and control(inference).

A system with the organization in Figure 1 is often referred to as a knowledge-based system. In the knowledge-base there are facts that are declarative knowledge about a particular problem being solved and the current state of the attempt to solve the problem. Also found in the knowledge base are rules that represent knowledge specific to solving a particular problem and are used to reason about the problem. Finally, the inference engine controls decisions on when and how to use specific problem-solving knowledge.

The organization of expert systems differs markedly from that of conventional computer programs and database systems. One of the major differences is in the treatment of rules as 'data' in the knowledge base. In the conventional computer programs domain-specific rules are embedded in the procedural knowledge coded as the program. Therefore, in conventional computer programs it is difficult to separate the rules from the inference, or control, mechanism. Database systems separate the facts to be stored in data files, whereas the domain-specific rules are combined with other considerations into the programs. This characteristic organization of expert systems is made possible through exploitation of the principles of unification and resolution (22). With the addition of rules to the collection of facts, rather complex situations can be adequately represented. It is this powerful representation of knowledge that leads some expert systems to be labelled as knowledge-based systems.

Knowledge Representation:

A well-known method of representing knowledge is by means of formulas in first-order predicate logic (17, 8). This method is commonly used to represent both facts and rules. Here simple declarative facts can be represented as instantiated predicates. For example, "parcel 435 is commercial land" can be adequately represented by LAND_USE(435, Commercial). More complicated statements may require more complex representation schemes.

Procedural knowledge can also be represented in first-order predicate logic if the logical formulas are suitably interpreted. The programming language Prolog is an example of just such an approach (13). In Prolog the formula

$$A_1 \text{ and } A_2 \text{ and } \ldots \text{ and } A_n \to B$$

can be thought of as the logical statement that B is true if A[1], A[2], ... A[n] are true. The formula above can also be thought of as a procedure for producing a state satisfying condition B. The two basic statements in Prolog are

fact: B is (fact)
rule: A_1 and A_2 and ... and A_n -> B.

B and all the A's must be predicates and all variables are considered to be universally quantified. That is to say that the statement is taken to be true for all possible values of the variables.

Another method to represent decalarative knowledge is to use semantic nets (or semantic networks). Semantic nets were introduced as a means of modeling human associative memory (20). In this method objects are represented by nodes in a graph and the relations among them by labeled arcs. The arcs are 'followed to proceed from node to node. A directed arc with label W between nodes A and B can signify that the predicate W(A,B) is true.

One of the useful aspects of semantic nets is their indexing property. The network can be constructed so objects often associated in computations, or those which are conceptually close to one another, may be represented by nodes in the network that are near each other (measured as number of arcs separating them). This property can be exploited in making analogical representations of states of affairs.

A third method of representing declarative knowledge is by using what is called frames (15). One can think of frames as data structures in which all knowledge about a particular object or event is stored together. However, the organization of knowledge can be made more modular, hence increasing its accessibility. Most variants of frame-based knowledge representation include the idea of having different types of frames for different types of objects. There are slots in each frame containing information relevant to that type of frame.

The main advantage of semantic nets or frames over the first-order predicate representation is that for each object, event, or concept, all relevant information is collected together. This makes accessing and manipulating the information easier. Also, default values can be created when information about an object or event is not explicitly given (18).

The selection of one method of representation over another appears to be a matter of suitability for a particular problem. The choice may influence the performance characteristics of the system but not its logical power. It is widely acknowledged that first-order predicate logic, semantic nets, and frames are generally equivalent forms of representation (18). Reiter (21) has gone so far as to suggest all systems descriptions include a first-order predicate logic definition of its semantics.

Knowledge Exploitation

Commonly expert systems are asked to solve problems where the precise series of steps necessary for reaching a solution are not known. It is therefore often necessary to search through a space containing many alternative paths, each potentially leading to a solution. The search space is very often extremely large. Thus, it is not practical to construct all potential paths, but rather the path is constructed only as far as necessary. Furthermore, most approaches to exploiting the knowledge in the knowledge-base are based on pattern matching.

Problem Reduction

Here the problem to be solved is decomposed into subproblems that can be solved separately. The problem is decomposed in such a way that combining the solutions to the subproblems will yield a solution to the original problem. Each subproblem can be further decomposed into smaller problems, until primitive problems that can be solved directly, are generated (18).

We can represent all possible decompositions of a problem using a problem reduction, or AND/OR graph. In the problem reduction graph each OR branch represents a choice of alternative decompositions, while each AND branch represents a particular way of decomposing a problem. Some decompositions of a problem may lead to solvable subproblems while others may not. To solve a problem using problem reduction, we must choose a decomposition yielding subproblems that can all be solved. To solve each of these subproblems, we again must choose decompositions that yield solvable sub-subproblems and so forth. Thus, the problem solution is represented by a solution graph.

State-space Search

State-spaces can be thought of as being represented by a graph. Walking down the graph is analogous to searching for a path to reach a state that corresponds to a solution. They can also be searched in a backward direction by starting with a goal, or solution, state and then finding a path to the initial state. The appropriateness of the approach depends on the particular problem and the nature of the state-space.

The decision to use forward chaining is determined by the data describing the current problem state. This is why the forward search is sometimes called data-driven driven search. One of the major problems with this approach lies int the identification of the facts useful for the problem at hand.

Backward search, or backward chaining, applies an inverse operator to some state, say G, and then tries to find all previous states, g', that allow it to reach goal G. This is then recursively applied to all subgoals g'. The advantage of this kind of search is that the goal is known from the start. On the other hand, this introduces additional complexity into needed pattern matching expressions.

Graph-searching is another control technique (19). These procedures explore several paths simultaneously, keeping track of several 'current states.' Examples of graph-searching procedures are -

Least-cost-first search (6) where at each iteration the path having the least accumulated cost is extended;

Breadth-first search where all paths are searched at the same speed;

Heuristic search where various heuristic criteria are used to determine which path, or paths, to extend next. Most of the known expert systems fall into this category.

Depth-first is easy to implement and thus popular. It uses a stack similar to the procedure 'call-stack' within a programming language. It explores one path as far as possible, ignoring all other paths until it fails to produce a solution. Then it backtracks to a previous state (saved on stack) and chooses another rule to extend the path in a different direction. Due to the nondeterministic nature of natural language, many parsers for analyzing natural language phrases make extensive use of backtracking (24).

Here the set of possible solutions becomes further and further constrained by rules or operators that produce 'local constraints' on what small pieces of the solution must look like. More and more rule applications are made until no more rules are applicable and only one, or some small number of, possible solution(s) is left. This technique avoids the necessity of backtracking since every existing solution must satisfy all the constraints produced by the rule applications.

Pattern Matching

Pattern matching typically plays a critical role in most methods of state-space search. That is, the selection of the next step in the path is based on the present state matching a pattern given a rule(s). For example, the use of a 'wildcare' to search a directory for a file is a simple form of pattern matching.

Pattern matching has a relatively long history in the field of artificial intelligence (12). Early Lisp-based languages such as CONNIVER (25) and PLANNER (11) and their descendants, all include methods to invoke rules based on matching patterns. Since the Japanese have selected Prolog for their Fifth Generation computer project, Prolog (4, 27) has lately become the most popular (14). It includes a sophisticated form of pattern matching call unification that is presented below.

Production Rules

Pattern-invoked programs are not called by other programs in the usual sense. Their invocation is driven by patterns in the rules being matched by the present state. One fundmenal type of pattern-invoked program is the production rule. It is a degenerate program of the form -

IF condition THEN primitive action.

The pattern is the condition that is usually a conjunction of predicates that test properties of the current state. The primitive action is some simple action that changes the current state. For example, it may change the state associate with a particular proposition from 'FALSE' to 'TRUE.'

Horn clauses are a powerful subset of first-order logic which lends itself well to automatic inferencing. It has the general form

$$\text{IF } C_1 \text{ and } C_2 \text{ and } \ldots C_n \text{ THEN } P$$

which is often written as

$$P \text{ IF } C_1 \text{ and } C_2 \text{ and } \ldots C_n .$$

This type of clause plays an important role in Prolog. The power of the Horn clause is based on exploiting the unification and resolution principles detailed in Robinson's (22) seminal paper on automatic theorem proving.

Unification and Resolution

A simple example will help clarify how a pattern matching program operates using unification and resolution. It is instructive to realize that the following example is one of backward chaining, depth-first searching. Lets consider that George owns parcel 435 which is classified as commercial land. Such a fact can be represented in first-order predicate logic as

parcel(435, George, Commercial)   (1).

Also in the knowledge base we include the general rule that parcel 'p' is used for activity 'u' and is owned by 'o'

own_use(u,o) if parcel(p,o,u)   (2).

Now we test the proposition that the owner of commercial land is 'o' by stating

own_use(<u>Commercial</u>, o).

The first step in the resolution process is to posit the negation

(not own_use(<u>Commercial</u>, o)).

Using (2) we substitute (u = Commercial) to have this predicate match with the first part of (2) and get

own_use(Commercial, o) if parcel(p,o, Commercial)

and then apply <u>modus tollens</u> (which says given that a implies b and not b therefore conclude not a) so that

(not own_use(<u>Commercial</u>, o)).

and

own_use(Commercial, o) if parcel(p,o, Commercial)

implies   (not parcel(p, o, <u>Commercial</u>)).

Then using (1) the following substitutions are made to achieve a match

o = George   ,   p = 435

Now we have

(not parcel(435, George, <u>Commercial</u>))

and   parcel(435, George, Commercial)

which is a contradiction. Thus, (not own_use(<u>Commercial</u>, p)) cannot be proven. Therefore, own_use(Commercial, p) can be shown with p = George. We can conclude that George owns commercial property.

In the formal definition of resolution the process of matching that occurred above is called unification. The completeness of the resolution principle is a very nice mathematical property. Essentially, Robinson (22) shows that if some fact follows from our proposition then we are able to prove its truth by showing the inconsistency of its negation.

Constructing Expert Systems

Knowledge acquisition is the transfer and transformation of problem solving expertise from some knowledge source to a form suitable for automatic exploitation (3). Typical knowledge sources are human experts, textbooks, scientific journals, etc. The process of knowledge acquisition is generally recognized as the most serious 'bottleneck' in the development of expert systems, although some domains are more difficult than others. Specifically, applications that depend on a narrow domain of knowledge are easier than those for which one needs creative and/or commonsense reasoning to solve problems (29). In the traditional model for knowledge acquisition, a knowledge engineer (an expert system expert) works with a domain expert to develop and test facts and rules for inclusion in the knowledge base. The knowledge engineer decides what type of facts to express using a chosen paradigm. The task is somewhat similar to a software engineer designing the logic of a large software system, but less is known about how to proceed, nor is it clear which paradigm is most efficient for a specific domain of interest. Thus, the knowledge engineer is concerned with the fundamental questions of what is the base of facts and rules, and what is to be deduced.

Once the area of application is clearly specified and delimited, a form for the organization of knowledge in smaller groups must be selected. Then the process of acquiring the desired knowledge from human experts in the field can begin. For existing systems this has taken a considerable length of time (several report many man-years of effort). This phase of building an expert system is presently very poorly understood. It demands the human expert to be very patient and willing to learn quite alot about the internal workings of an expert system. Davis and Lenat (5) suggest a direct interaction between domain expert and program, without the knowledge engineer, in much the same manner as a teacher interacts with a student, providing new problems and observing how the system attacks them. Thus, the domain expert 'debugs' the system, providing new knowledge as needed.

One approach to the construction of an expert system is to use a so-called expert system 'shell' or 'skeleton' to build around. Such shells allow the designer to focus on the knowledge base rather than the details of the internal workings of an expert system. Two examples of shells that have been used with some success are EMYCIN (for Empty MYCIN) and KAS (Knowledge Acquisition System). Both systems are derived from expert systems, MYCIN and PROSPECTOR respectively, with the domain-specific knowledge removed. In such systems all domain-specific knowledge is represented as explicit rules rather than being hard-coded into the inference engine. While greatly simplifying the task, shells diminish in value if the representation, rule language, or structure is inappropriate to the new domain (1).

Expert system shells provide the builder with a number of tools for effective use of the inference engine. They typically help during the input and editing of facts and rules, allow browsing through the knowledge base, and facilitate debugging of rules in order to detect combinations that are not intended. They further may contain facilities to ask a user for additional facts that become necessary during an inference and must include a system to explain to a user how certain conclusions are reached with an increasing degree of detail. The shell should provide a means to organize knowledge and present them to the expert in a coherent and logical manner without bothering him with more detail than is required.

Some Concluding Comments

Inference systems as known today do not efficiently support large scale data collection. Currently we see three major limitations -

(1) The organization of the knowledge base is inefficient for large collections of facts and rules. Here we are speak of a large collection as being composed of more than 1 million facts which is probably the lower limit for a GIS of any practical use.

(2) The number of logical inferences processed per second (LIPS) is too small for practical applications to large knowledge base collections. This limitation is of particular importance in applying expert system technology to problems of geographic information processing.

(3) Generally, the knowledge base must be stored in in main memory and not in secondary memory (disk). Even the use of large virtual memory spaces as provided in modern operating systems will not alleviate this shortcoming. Storage structures suitable for secondary storage must minimize the number of slow disk page accesses. Present systems assume uniform access time. Thus, they are inefficient if some of the knowledge base is stored in main memory and other portions in secondary memory.

Some further considerations when constructing expert systems are essentially time, money, and manpower. Construction of an expert system sometimes takes as many as 10-25 man-years and costs as much as $1 to $2 million. A contributing factor is the lack of software tools for implementing expert systems. However, progress is being made in this direction. A number of companies are now selling expert system development tools (23) Another obstacle is the amount of time required to take the knowledge from an expert in some problem domain and encode it in a knowledge base. Again this problem is partially due to a lack of tools for the task. It also reflects the large gaps remaining in our understanding of human problem solving.

Acknowledgements

We would like to acknowledge the hard work of Mr. Dominique Thongs who conducted many a library search for needed material. Also, we would like to acknowledge the partial support of a research grant from PSC-CUNY and National Science Foundation Grant IST-8412406.

Appendix I. - References

1. Barstow, D.R. et al, "Expert System Tools," in Building Expert Systems, F. Hayes-Roth, D.A. Waterman, and D.B. Lenat (eds.), Addison-Wesley, Reading, 1983.

2. Brachman, R.J. et al, "What Are Expert Systems?," in Building Expert Systems, F. Hayes-Roth, D.A. Waterman, and D.B. Lenat (eds.), Addison-Wesley, Reading, 1983.

3. Buchanan, B.G. et al, "Constructing an Expert System," in Building Expert Systems, F. Hayes-Roth, D.A. Waterman, and D.B. Lenat (eds.), Addison-Wesley, Reading, 1983.

4. Clocksin, W. and Mellish, C., Programming in PROLOG, Springer-Verlag, New York, 1981.

5. Davis, R. and Lenat, D.B., Knowledge Based Systems in Artificial Intelligence, McGraw-Hill, New York, 1982.

6. Dijkstra, E.W., "A Note on Two Problems in Connection with Graphs," Numerical Mathematics, Vol 1, 1959, pp. 269-271.

7. Forsyth, R., "The Architecture of Expert Systems," in Expert Systems: Principles and Case Studies, R. Forsyth (ed.), Chapman and Hall, London, 1984.

8. Gallaire, H., Minker, J. and Nicolas, J.M., "Logic and Databases: A Deductive Approach," ACM Computing Surveys, Vol. 16, 1984, pp. 153-185.

9. Gevarter, W.B., An Overview of Artificial Intelligence and Robotics, NASA Technical Memorandum 85836, National Aeronautics and Space Administration, Washington, D.C., 1983.

10. Hayes-Roth, F., "Knowledge-based Expert Systems," Computer, Vol. 17 1984, pp. 263 - 273.

11. Hewitt, C., 'PLANNER: A Language for Proving Theorems in Robots,' International Journal of Conference on Artificial Intelligence, Vol. -2, 1969, pp. 167-182.

12. Jackson, P.C., Introduction to Artificial Intelligence, 2nd ed., Dover Publications, New York, 1985.

13. Kowalski, R.A., Logic for Problem Solving, North-Holland, New York, 1979.

14. McDermott, D., "The PROLOG Phenomenon," Sigart Newsletter, No. 72, 1980, pp. 16-20.

15. Minsky, M., "A Framework for Representing Knowledge," in The Psychology of Computer Vision, P.H. Winston (ed.), McGraw-Hill, New York, 1975, pp. 211-277.

16. Mooneyhan, D.W. 'The Potential of Expert Systems for Remote Sensing,' Proceedings, 17th International Symposium on Remote Sensing of Environment, Ann Arbor, Michigan, 1983 .

17. Mylopoulos, J. "An Overview of Knowledge Representation," Proceedings Workshop Data Abstraction, Databases, and Conceptual Modeling, 1980, pp. 5-12.

18. Nau, D.S., "Expert Computer Systems," Computer, Vol 16, 1983, pp. 63-85.

19. Nilsson, N., Problem-solving Methods in Artificial Intelligence, McGraw-Hill, New York, 1971.

20. Quillian, M.R., 'Semantic Memory,' in Semantic Information Processing, M. Minsky (ed.), Cambridge, MA. MIT Press, 1968.

21. Reiter, R., 'Towards a Logical Reconstruction of Relational Database Theory,' in On Conceptual Modeling, M. Brodie, J. Mylopoulos, and J.W. Schmidt (eds.), Springer-Verlag, Berlin, 1984.

22. Robinson, J.A., 'A Machine-oriented Logic Based on the Resolution Principle,' Journal of Association of Computing Machinery, Vol. 12, 1965, pp. 23-41.

23. Schlosberg, J., 'Almost Human,' Digital Review, 1985, pp. 37-42.

24. Sowa, J.F., Conceptual Structures: Information Processing in Mind and Machine, Addison-Wesley: Reading, 1984.

25. Sussman, G.J. and McDermott, D.V. The CONNIVER reference manual. AI Memo No. 259. M.I.T. Artificial Intelligence Laboratory : Cambridge, Mass, 1972.

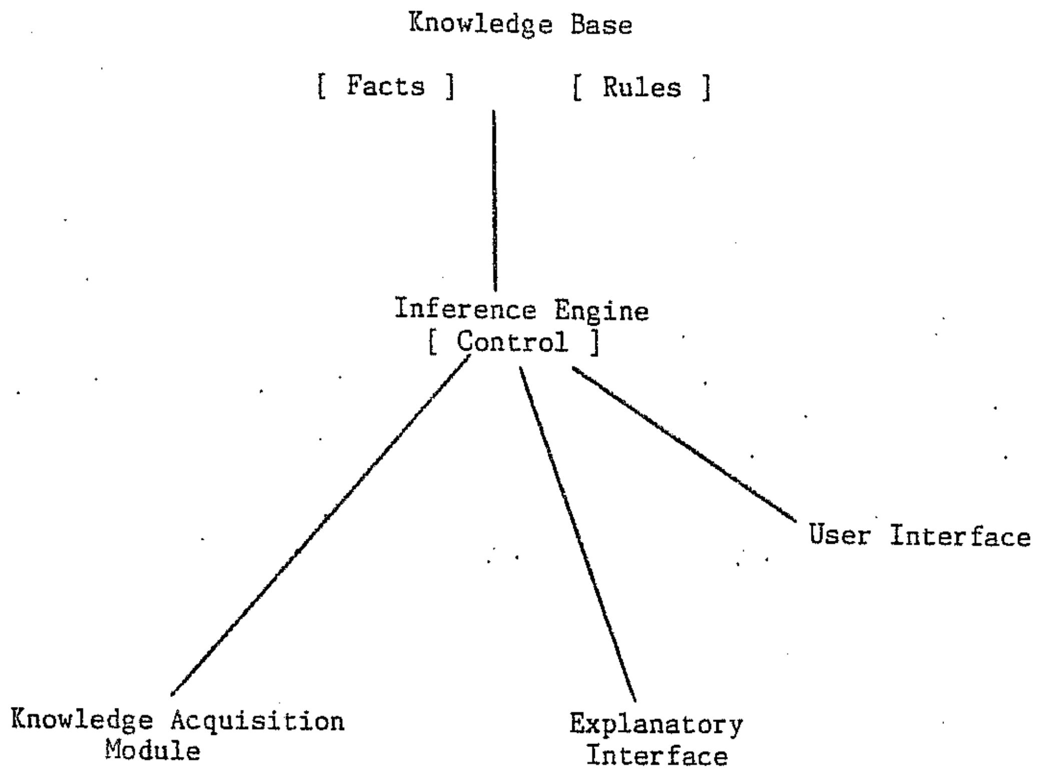
26. Thompson, B.A. and Thompson, W.A., 'Inside an Expert System,' Byte, Vol. 10, 1985, pp. 315-332.

27. van Emden, M.H. and Kowalski, R.A., "The Semantics of Predicate Logic as a Programming Language," Journal of Association of Computing Machinery, vol 23, 1976, pp. .

28. Weiss, S.M. and Kulikowski, C.A., A Practical Guide to Designing Expert Systems, Rowman and Allenheld, Totawa, NJ., 1984.

29. Yazdani, M. , "Knowledge Engineering in Prolog," in Expert Systems: Principles and Case Studies, R. Forsyth (ed.), Chapman and Hall, London, 1984.

Figure 1. Essential Components of an Expert System.


Knowledge Base
[Facts]
[Rules]
Inference Engine
[Control]
User Interface
Knowledge Acquisition Module
Explanatory Interface