MODULE 4 ARTIFICIAL INTELLIGENCE AND ITS APPLICATIONS

Unit 1 Expert System
Unit 2 Robotics

UNIT 1 EXPERT SYSTEM

CONTENTS

1.0 Introduction
2.0 Objectives
3.0 Main Content
- 3.1 What is an Expert System?
  - 3.1.1 Comparison to Problem-Solving Systems
- 3.2 Knowledge Base
  - 3.2.1 Types of Knowledge Base
- 3.3. Inference Engine
  - 3.3.1 Architecture
  - 3.3.2 The Recognize-Act Cycle
  - 3.3.3 Data-Driven Computation versus Procedural Control
  - 3.3.4 Inference Rules
  - 3.3. 5 Chaining
- 3.4 Certainty Factors
- 3.5 Real-Time Adaption
  - 3.5.1 Ability to make Relevant Inquiries
- 3.7 Knowledge Engineering
- 3.8 General Types of Problems Solved
- 3.9 Different Types of Expert System
- 3.10 Examples of Applications
- 3.11 Advantages
- 3.12 Disadvantages

4.0 Conclusion
5.0 Summary
6.0 Tutor-Marked Assignment
7.0 References/Further Reading

1.0 INTRODUCTION

Expert system is a computer program that uses artificial intelligence to solve problems within a specialized domain that ordinarily requires human expertise. The first expert system was developed in 1965 by Edward Feigenbaum and Joshua Lederberg of Stanford University in California, U.S. Dendral, as their expert system was later known, was

designed to analyze chemical compounds. Expert systems now have commercial applications in fields as diverse as medical diagnosis, petroleum engineering, financial investing make financial forecasts and schedule routes for delivery vehicles.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

Explain an expert system
Distinction between expert systems and traditional problem solving programs
Explain the term “knowledge base”.

3.0 MAIN CONTENT

3.1 What is an Expert System?

It is a computer application that performs a task that would otherwise be performed by a human expert. Some expert systems are designed to take the place of human experts, while others are designed to aid them.

To design an expert system, one needs a *knowledge engineer*, an individual who studies how human experts make decisions and translates the rules into terms that a computer can understand. In order to accomplish feats of apparent intelligence, an expert system relies on two components: a knowledge base and an inference engine.

3.1.1 Comparison to problem-solving systems

The principal distinction between expert systems and traditional problem solving.

Programs are the way in which the problem related expertise is coded. In traditional applications, problem-related expertise is encoded in both program and data structures. In the expert system approach all of the problem expertise is encoded mostly in data structures.

In an example related to tax advice, the traditional approach has data structures that describe the taxpayer and tax tables, and a program that contains rules (encoding expert knowledge) that relate information about the taxpayer to tax table choices. In contrast, in the expert system approach, the latter information is also encoded in data structures. The collective data structures are called the knowledge base . The program (inference engine) of an expert system is relatively independent of the

problem domain (taxes) and processes the rules without regard to the problem area they describe.

This organization has several benefits:

> New rules can be added to the knowledge base or altered without needing to rebuild the program. This allows changes to be made rapidly to a system (e.g., after it has been shipped to its customers, to accommodate very recent changes in state or federal tax codes).
> Rules are arguably easier for (non-programmer) domain experts to create and modify than writing code. Commercial rule engines typically come with editors that allow rule creation/modification through a graphical user interface, which also performs actions such as consistency and redundancy checks.

Modern rule engines allow a hybrid approach: some allow rules to be "compiled" into a form that is more efficiently machine-executable. Also, for efficiency concerns, rule engines allow rules to be defined more expressively and concisely by allowing software developers to create functions in a traditional programming language such as Java, which can then be invoked from either the condition or the action of a rule. Such functions may incorporate domain-specific (but reusable) logic.

3.2 Knowledge Base

A knowledge base (abbreviated KB, kb or Δ) is a special kind of database for knowledge management, providing the means for the computerized collection, organization, and retrieval of knowledge. Also, it is a collection of data representing related experiences which their results is related to their problems and solutions.

Facts for a knowledge base must be acquired from human experts through interviews and observations. This knowledge is then usually represented in the form of “if-then” rules (production rules): “If some condition is true then the following inference can be made (or some action taken).” The knowledge base of a major expert system includes thousands of rules. A probability factor is often attached to the conclusion of each production rule, because the conclusion is not a certainty. For example, a system for the diagnosis of eye diseases might indicate, based on information supplied to it, a 90 percent probability that a person has glaucoma, and it might also list conclusions with lower probabilities. An expert system may display the sequence of rules through which it arrived at its conclusion; tracing this flow helps the

user to appraise the credibility of its recommendation and is useful as a learning tool for students.

Human experts frequently employ heuristic rules, or “rules of thumb,” in addition to simple production rules. For example, a credit manager might know that an applicant with a poor credit history, but a clean record since acquiring a new job, might actually be a good credit risk. Expert systems have incorporated such heuristic rules and increasingly have the ability to learn from experience. Nevertheless, expert systems remain supplements, rather than replacements, for human experts.

3.2.1 Types Of Knowledge Base

Knowledge bases are essentially closed or open information repositories and can be categorized under two main headings:

> Machine-readable knowledge bases store knowledge in a computer-readable form, usually for the purpose of having automated deductive reasoning applied to them. They contain a set of data, often in the form of rules that describe the knowledge in a logically consistent manner. An ontology can define the structure of stored data - what types of entities are recorded and what their relationships are. Logical operators, such as *And* (conjunction), *Or* (disjunction), *material implication* and *negation* may be used to build it up from simpler pieces of information. Consequently, classical deduction can be used to reason about the knowledge in the knowledge base. Some machine-readable knowledge bases are used with artificial intelligence, for example as part of an expert system that focuses on a domain like prescription drugs or customs law. Such knowledge bases are also used by the semantic web.
> Human-readable knowledge bases are designed to allow people to retrieve and use the knowledge they contain. They are commonly used to complement a help desk or for sharing information among employees within an organization. They might store troubleshooting information, articles, white papers, user manuals, knowledge tags, or answers to frequently asked questions. Typically, a search engine is used to locate information in the system, or users may browse through a classification scheme.

A text based system that can include groups of documents including hyperlinks between them is known as Hypertext Systems. Hypertext systems support the decision process by relieving the user of the significant effort it takes to relate and remember things." Knowledge

bases can exist on both computers and mobile phones in a hypertext format. Knowledge base analysis and design (also known as KBAD) is an approach that allows people to conduct analysis and design in a way that result in a knowledge base, which can later be used to make informative decisions. This approach was first implemented by Dr. Steven H. Dam

3.3 Inference Engine

In computer science, and specifically the branches of knowledge engineering and artificial intelligence, an inference engine is a computer program that tries to derive answers from a knowledge base. It is the "brain" that expert systems use to reason about the information in the knowledge base for the ultimate purpose of formulating new conclusions. Inference engines are considered to be a special case of reasoning engines, which can use more general methods of reasoning.

3.3.1 Architecture

The separation of inference engines as a distinct software component stems from the typical production system architecture. This architecture relies on a data store:

1. An interpreter. The interpreter executes the chosen agenda items by applying the corresponding base rules.
2. A scheduler. The scheduler maintains control over the agenda by estimating the effects of applying inference rules in light of item priorities or other criteria on the agenda.
3. A consistency enforcer. The consistency enforcer attempts to maintain a consistent representation of the emerging solution

3.3.2 The Recognize-Act Cycle

The inference engine can be described as a form of finite state machine with a cycle consisting of three action states: *match rules*, *select rules*, and *execute rules*. Rules are represented in the system by a notation called predicate logic.

In the first state, match rules, the inference engine finds all of the rules that are satisfied by the current contents of the data store. When rules are in the typical *condition-action* form, this means testing the conditions against the working memory. The rule matching that are found are all candidates for execution: they are collectively referred to as the *conflict set*. Note that the same rule may appear several times in the conflict set

if it matches different subsets of data items. The pair of a rule and a subset of matching data items are called an *instantiation* of the rule.
In many applications, where large volumes of data are concerned and/or when performance time considerations are critical, the computation of the conflict set is a non-trivial problem.

3.3.3 Data-Driven Computation versus Procedural Control

The inference engine control is based on the frequent re - evaluation of the data store states, not on any static control structure of the program. The computation is often qualified as *data-driven* or *pattern-directed* in contrast to the more traditional procedural control. Rules can communicate with one another only by way of the data, whereas in traditional programming languages procedures and functions explicitly call one another. Unlike instructions, rules are not executed sequentially and it is not always possible to determine through inspection of a set of rules which rule will be executed first or cause the inference engine to terminate.

In contrast to a procedural computation, in which knowledge about the problem domain is mixed in with instructions about the flow of control—although object-oriented programming languages mitigate this entanglement—the inference engine model allows a more complete separation of the knowledge (in the rules) from the control (the inference engine).

3.3.4 Inference Rules

An inference rule is a conditional statement with two parts namely; if clause and a then clause.

This rule is what gives expert systems the ability to find solutions to diagnostic and prescriptive problems. An example of an inference rule is:

If the restaurant choice includes French and the occasion is
romantic, Then the restaurant choice is definitely Paul Bocuse.

An expert system's rule base is made up of many such inference rules. They are entered as separate rules and it is the inference engine that uses them together to draw conclusions. Because each rule is a unit, rules may be deleted or added without affecting other rules - though it should affect which conclusions are reached. One advantage of inference rules over traditional programming is that inference rules use reasoning which more closely resembles human reasoning.

Thus, when a conclusion is drawn, it is possible to understand how this conclusion was reached. Furthermore, because the expert system uses knowledge in a form similar to the that of the expert, it may be easier to retrieve this information directly from the expert.

3.3.5 Chaining

Two methods of reasoning when using inference rules are forward chaining and backward chaining.

Forward chaining starts with the data available and uses the inference rules to extract more data until a desired goal is reached. An inference engine using forward chaining searches the inference rules until it finds one in which the if clause is known to be true . It then concludes the then clause and adds this information to its data. It continues to do this until a goal is reached. Because the data available determines which inference rules are used, this method is also classified as data driven. Backward chaining starts with a list of goals and works backwards to see if there is data which will allow it to conclude any of these goals. An inference engine using backward chaining would search the inference rules until it finds one which has a then clause that matches a desired goal. If the if clause of that inference rule is not known to be true, then it is added to the list of goals. For example, suppose a rule base contains:

(1) IF X is green THEN X is a frog. (Confidence Factor: +1%)
(2) IF X is NOT green THEN X is NOT a frog. (Confidence Factor: +99%)
(3) IF X is a frog THEN X hops. (Confidence Factor: +50%)
(4) IF X is NOT a frog THEN X does NOT hop. (Confidence Factor +50%)

Suppose a goal is to conclude that Fritz hops. Let X = "Fritz". The rule base would be searched and rule (3) would be selected because its conclusion (the then clause) matches the goal. It is not known that Fritz is a frog, so this "if" statement is added to the goal list. The rule base is again searched and this time rule (1) is selected because its then clause matches the new goal just added to the list. This time, the if clause (Fritz is green) is known to be true and the goal that Fritz hops is concluded. Because the list of goals determines which rules are selected and used, this method is called goal driven.

However, note that if we use confidence factors in even a simplistic fashion - for example, by multiplying them together as if they were like soft probabilities - we get a result that is known with a confidence factor of only one-half of 1%. (This is by multiplying 0.5 x 0.01 = 0.005). This

is useful, because without confidence factors, we might erroneously conclude with certainty that a sea turtle named Fritz hops just by virtue of being green. In Classical logic or Aristotelian term logic systems, there are no probabilities or confidence factors; all facts are regarded as certain. An ancient example from Aristotle states, "Socrates is a man. All men are mortal. Thus Socrates is mortal."

In real world applications, few facts are known with absolute certainty and the opposite of a given statement may be more likely to be true ("Green things in the pet store are not frogs, with the probability or confidence factor of 99% in my pet store survey"). Thus it is often useful when building such systems to try and prove both the goal and the opposite of a given goal to see which is more likely.

3.4 Certainty Factors

One method of operation of expert systems is through a quasi-probabilistic approach with certainty factors: A human, when reasoning, does not always make statements with 100% confidence: he might venture, "If Fritz is green, then he is probably a frog" (after all, he might be a chameleon). This type of reasoning can be imitated using numeric values called confidences. For example, if it is known that Fritz is green, it might be concluded with 0.85 confidence that he is a frog; or, if it is known that he is a frog, it might be concluded with 0.95 confidence that he hops. These Certainty factor (CF) numbers quantify uncertainty in the degree to which the available evidence supports a hypothesis. They represent a degree of confirmation, and are not probabilities in a Bayesian sense. The CF calculus, developed by Shortliffe & Buchanan, increases or decreases the CF associated with a hypothesis as each new piece of evidence becomes available. It can be mapped to a probability update, although degrees of confirmation are not expected to obey the laws of probability. It is important to note, for example, that evidence for hypothesis H may have nothing to contribute to the degree to which Noth is confirmed or disconfirmed (e.g., although a fever lends some support to a diagnosis of infection, fever does not disconfirm alternative hypotheses) and that the sum of CFs of many competing hypotheses may be greater than one (i.e., many hypotheses may be well confirmed based on available evidence).

The CF approach to a rule-based expert system design does not have a widespread following, in part because of the difficulty of meaningfully assigning CFs a priori. (The above example of green creatures being likely to be frogs is excessively naive.) Alternative approaches to quasi-probabilistic reasoning in expert systems involve fuzzy logic, which has a firmer mathematical foundation. Also, rule-engine shells such as

Drools and Jess do not support probability manipulation: they use an alternative mechanism called salience, which is used to prioritize the order of evaluation of activated rules.

In certain areas, as in the tax-advice scenarios discussed below, probabilistic approaches are not acceptable. For instance, a 95% probability of being correct means a 5% probability of being wrong. The rules that are defined in such systems have no exceptions: they are only a means of achieving software flexibility when external circumstances change frequently. Because rules are stored as data, the core software does not need to be rebuilt each time changes to federal and state tax codes are announced.

3.5 Real-Time Adaption

Industrial processes, data networks, and many other systems change their state and even their structure over time. Real time expert systems are designed to reason over time and change conclusions as the monitored system changes. Most of these systems must respond to constantly changing input data, arriving automatically from other systems such as process control systems or network management systems.

Representation includes features for defining changes in belief of data or conclusions over time. This is necessary because data becomes stale. Approaches to this can include decaying belief functions, or the simpler validity interval that simply lets data and conclusions expire after specified time period, falling to "unknown" until refreshed. An often-cited example (attributed to real time expert system pioneer Robert L. Moore) is a hypothetical expert system that might be used to drive a car. Based on video input, there might be an intermediate conclusion that a stop light is green and a final conclusion that it is OK to drive through the intersection. But that data and the subsequent conclusions have a very limited lifetime. You would not want to be a passenger in a car driven based on data and conclusions that were, say, an hour old.

The inference engine must track the times of each data input and each conclusion, and propagate new information as it arrives. It must ensure that all conclusions are still current. Facilities for periodically scanning data, acquiring data on demand, and filtering noise, become essential parts of the overall system. Facilities to reason within a fixed deadline are important in many of these applications.

An overview of requirements for a real-time expert system shell is given in. Examples of real time expert system applications are given in and.

Several conferences were dedicated to real time expert system applications in the chemical process industries, including.

3.5.1 Ability to Make Relevant Inquiries

An additional skill of an expert system is the ability to give relevant inquiries based on previous input from a human user, in order to give better replies or other actions, as well as working faster, which also pleases an impatient or busy human user - it allows a priori volunteering of information that the user considers important.

Also, the user may choose not to respond to every question, forcing the expert system to function in the presence of partial information.

Commercially viable systems will try to optimize the user experience by presenting options for commonly requested information based on a history of previous queries of the system using technology such as forms, augmented by keyword-based search. The gathered information may be verified by a confirmation step (e.g., to recover from spelling mistakes), and now act as an input into a forward-chaining engine. If confirmatory questions are asked in a subsequent phase, based on the rules activated by the obtained information, they are more likely to be specific and relevant. Such abilities can largely be achieved by control flow structures.

In an expert system, implementing the ability to learn from a stored history of its previous use involves employing technologies considerably different from that of rule engines, and is considerably more challenging from a software-engineering perspective. It can, however, make the difference between commercial success and failure. A large part of the revulsion that users felt towards Microsoft's Office Assistant was due to the extreme naivete of its rules ("It looks like you are typing a letter: would you like help?") and its failure to adapt to the user's level of expertise over time (e.g. a user who regularly uses features such as Styles, Outline view, Table of Contents or cross-references is unlikely to be a beginner who needs help writing a letter).

3.6 Explanation System

Another major distinction between expert systems and traditional systems is illustrated by the following answer given by the system when the user answers a question with another question, "Why", as occurred in the above example. The answer is:

A. I am trying to determine the type of restaurant to suggest. So far Indian is not a likely choice. It is possible that French is a likely choice. If I know that if the diner is a wine drinker, and the preferred wine is French, then there is strong evidence that the restaurant choice should include French.

It is very difficult to implement a general explanation system (answering questions like "Why" and "How") in a traditional computer program. An expert system can generate an explanation by retracing the steps of its reasoning. The response of the expert system to the question "Why" exposes the underlying knowledge structure. It is a rule; a set of antecedent conditions which, if true, allow the assertion of a consequent. The rule references values, and tests them against various constraints or asserts constraints onto them. This, in fact, is a significant part of the knowledge structure. There are values, which may be associated with some organizing entity. For example, the individual diner is an entity with various attributes (values) including whether they drink wine and the kind of wine. There are also rules, which associate the currently known values of some attributes with assertions that can be made about other attributes. It is the orderly processing of these rules that dictates the dialogue itself.

3.7 Knowledge Engineering

The building, maintaining and development of expert systems are known as knowledge engineering. Knowledge engineering is a "discipline that involves integrating knowledge into computer systems in order to solve complex problems normally requiring a high level of human.

There are generally three individuals having an interaction in an expert system. Primary among these is the end-user, the individual who uses the system for its problem solving assistance. In the construction and maintenance of the system there are two other roles: the problem domain expert who builds the system and supplies the knowledge base, and a knowledge engineer who assists the experts in determining the representation of their knowledge, enters this knowledge into an explanation module and who defines the inference technique required to solve the problem. Usually the knowledge engineer will represent the problem solving activity in the form of rules. When these rules are created from domain expertise, the knowledge base stores the rules of the expert system.

3.8 General Types of Problems Solved

Expert systems are most valuable to organizations that have a high-level of know-how experience and expertise that cannot be easily transferred to other members. They are designed to carry the intelligence and information found in the intellect of experts and provide this knowledge to other members of the organization for problem-solving purposes.

Typically, the problems to be solved are of the sort that would normally be tackled by a professional, such as a medical professional in the case of clinical decision support systems. Real experts in the problem domain (which will typically be very narrow, for instance "diagnosing skin conditions in teenagers") are asked to provide "rules of thumb" on how they evaluate the problem — either explicitly with the aid of experienced systems developers, or sometimes implicitly, by getting such experts to evaluate test cases and using computer programs to examine the test data and derive rules from that (in a strictly limited manner). Generally, expert systems are used for problems for which there is no single "correct" solution which can be encoded in a conventional algorithm — one would not write an expert system to find the shortest paths through graphs, or to sort data, as there are simpler ways to do these tasks.

Simple systems use simple true/false logic to evaluate data. More sophisticated systems are capable of performing at least some evaluation, taking into account real-world uncertainties, using such methods as fuzzy logic. Such sophistication is difficult to develop and still highly imperfect.

3.9 Different Types of Expert System are

Rule-Based expert system
Frames-Based expert system
Hybird system
Model-based expert system
Ready-made system
Real-Time expert system

3.10 Examples of Applications

Expert systems are designed to facilitate tasks in the fields of accounting, medicine, process control, financial service, production, human resources among others. Typically, the problem area is complex enough that a more simple traditional algorithm cannot provide a proper solution, The foundation of a successful expert system depends on a

series of technical procedures and development that may be designed by technicians and related experts. As such, expert systems do not typically provide a definitive answer, but provide probabilistic recommendations. An example of the application of expert systems in the financial field is expert systems for mortgages. Loan departments are interested in expert systems for mortgages because of the growing cost of labour, which makes the handling and acceptance of relatively small loans less profitable. They also see a possibility for standardised, efficient handling of mortgage loan by applying expert systems, appreciating that for the acceptance of mortgages there are hard and fast rules which do not always exist with other types of loans. Another common application in the financial area for expert systems is in trading recommendations in various marketplaces. These markets involve numerous variables and human emotions which may be impossible to deterministically characterize, thus expert systems based on the rules of thumb from experts and simulation data are used. Expert system of this type can range from ones providing regional retail recommendations, like Wishabi, to ones used to assist monetary decisions by financial institutions and governments. Another 1970s and 1980s application of expert systems, which we today would simply call AI, was in computer games. For example, the computer baseball games Earl Weaver Baseball and Tony La Russa Baseball each had highly detailed simulations of the game strategies of those two baseball managers. When a human played the game against the computer, the computer queried the Earl Weaver or Tony La Russa Expert System for a decision on what strategy to follow. Even those choices where some randomness was part of the natural system (such as when to throw a surprise pitch-out to try to trick a runner trying to steal a base) were decided based on probabilities supplied by Weaver or La Russa. Today we would simply say that "the game's AI provided the opposing manager's strategy."

3.11 Advantages

Compared to traditional programming techniques, expert-system approaches provide the added flexibility (and hence easier modifiability) with the ability to model rules as data rather than as code. In situations where an organization's IT department is overwhelmed by a software-development backlog, rule-engines, by facilitating turnaround, provide a means that can allow organizations to adapt more readily to changing needs.

In practice, modern expert-system technology is employed as an adjunct to traditional programming techniques, and this hybrid approach allows the combination of the strengths of both approaches. Thus, rule engines allow control through programs

(and user interfaces) written in a traditional language, and also incorporate necessary functionality such as inter-operability with existing database technology.

3.12 Disadvantages

- The Garbage In, Garbage Out (GIGO) phenomenon: A system that uses expert-system technology provides no guarantee about the quality of the rules on which it operates. All self-designated "experts" are not necessarily so, and one notable challenge in expert system design is in getting a system to recognize the limits to its knowledge.
- Expert systems are notoriously narrow in their domain of knowledge— as an amusing example, a researcher used the "skin disease" expert system to diagnose his rust bucket car as likely to have developed measles — and the systems are thus prone to making errors that humans would easily spot. Additionally, once some of the mystique had worn off, most programmers realized that simple expert systems were essentially just slightly more elaborate versions of the decision logic they had already been using. Therefore, some of the techniques of expert systems can now be found in most complex programs without drawing much recognition.

  An expert system or rule-based approach is not optimal for all problems, and considerable knowledge is required so as to not misapply the systems.

  Ease of rule creation and rule modification can be double-edged. A system can be sabotaged by a non-knowledgeable user who can easily add worthless rules or rules that conflict with existing ones. Reasons for the failure of many systems include the absence of (or neglect to employ diligently) facilities for system audit, detection of possible conflict, and rule lifecycle management (e.g. version control, or thorough testing before deployment). The problems to be addressed here are as much technological as organizational.

An example and a good demonstration of the limitations of an expert system is the Windows operating system troubleshooting software located in the "help" section in the taskbar menu. Obtaining technical operating system support is often difficult for individuals not closely involved with the development of the Operating System. Microsoft has designed their expert system to provide solutions, advice, and suggestions to common errors encountered while using their operating systems.

4.0 CONCLUSION

Expert systems now have commercial applications in fields as diverse as medical diagnosis, petroleum engineering, financial investing make financial forecasts and schedule routes for delivery vehicles.

5.0 SUMMARY

In this unit, you learnt:

Definition of an Expert System
Knowledge Base and Types of Knowledge Base
Inference Engine
Certainty factors
Real-time adaption
Knowledge Engineering
General types of problems solved
Different types of expert system

6.0 TUTOR-MARKED ASSIGNMENT

i. Explain expert system.
ii. Mention and explain two methods of reasoning when using inference rules.
iii. Describe two type of knowledge bases.
iv. Mention two advantages of expert system.

7.0 REFERENCES/FURTHER READING

Argumentation in Artificial Intelligence by Iyad Rahwan, Guillermo R. Simari.

OWL DL Semantics. http://www.obitko.com/tutorials/ontologies-semantic-web/owl-dl-semantics.html.

Marakas, George. Decision Support Systems in the 21st Century. Prentice Hall, 1999, p.29.