Artificial Intelligence

Software is now being developed which will enable computers to **learn** and **reason**.
egg some chess game programs get better the more they are played - the computer remembers 'when I made that move, I lost...therefore find an alternative'.

AI is difficult to define. Alan Turing (a prominent mathematician) developed a simple test (1950) to determine if a computer possessed intelligence :
Suppose there are two identical terminals in a room, one connected to a computer, and the other operated by a person. If someone using the two terminals is unable to tell which is connected to the computer and which is operated by the person, then the computer can be credited with intelligence.

AI research includes -

- **language processing** - understanding and speaking languages as well as humans.
- **computer vision** - recognising and analysing objects.

Neural networks.
The current technology for this is only in its infancy. Nothing to do with computer networks, **neural networks** try to mimic the way that the human brain works (neurons, synapses etc)
Work on neural networks is being carried out in the field of image analysis, pattern analysis, financial trends etc...

Aspirin is a language used in neural networks (Using a **MIGRAINES** interface!).

Requirements

Hardware
Parallel processors with powerful capabilities to handle numbers very quickly
Software

- Languages such as OCCAM

How does a parallel processor handle an instruction
Instruction is split up and each processor does part of it
Egg 4 processors adding up 8 numbers

P1	P2	P3	P4
1+2	3+4	5+6	7+8
P1+ P2	P3 +P4		

Result of above

Expert Systems

An expert system is a computer system that emulates the decision-making ability of a human expert.

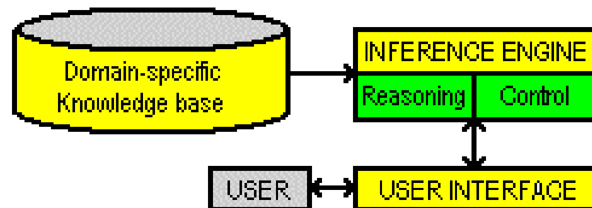
A **knowledge-based system** which attempts to replace a human 'expert' in a particular field.

It diagnoses problems and gives advice on that the cause of those problems are. They can also give advice on solutions.

Components of a rule-based expert system

A typical rule-based expert system integrates

1. **A problem-domain-specific knowledge base** that stores the encoded knowledge to support one problem domain such as diagnosing why a car won't start. In a rule-based expert system, the knowledge base includes the if-then rules and



additional specifications that control the course of the interview.

2. **An inference engine**
a set of rules for making **deductions** from the data and that implements the reasoning mechanism and controls the interview process. The inference engine might be generalized so that the same software is able to process many different knowledge bases.
3. **The user interface**
requests information from the user and outputs intermediate and final results. In some expert systems, input is acquired from additional sources such as data bases and sensors.

An **expert system shell** consists of a generalized inference engine and user interface designed to work with a knowledge base provided in a specified format. A shell often includes tools that help with the design, development and testing of the knowledge base. With the shell approach, expert systems representing many different problem domains may be developed and delivered with the same software environment. .

There are special high level languages used to program expert systems egg **PROLOG**

The user interacts with the system through a *user interface* which may use menus, natural language or any other style of interaction). Then an *inference engine* is used to reason with both the *expert knowledge* (extracted from our friendly expert) and data specific to the particular problem being solved. The expert knowledge will typically be in the form of a set of IF-THEN rules. The *case specific data* includes both data provided by the user and partial conclusions (along with certainty measures) based on this data. In a simple forward chaining rule-based system the case specific data will be the elements in *working memory*.

How an expert system works

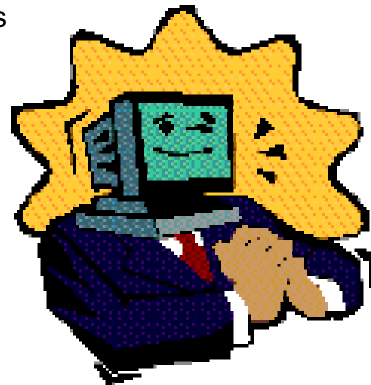
Car engine diagnosis

1. IF engine_getting_petrol
   AND engine_turns_over
   THEN problem_with_spark_plugs

2. IF NOT engine_turns_over
   AND NOT lights_come_on
   THEN problem_with_battery

3. IF NOT engine_turns_over
   AND lights_come_on
   THEN problem_with_starter

4. IF petrol_in_fuel_tank
   THEN engine_getting_petrol



There are three possible problems with the car:

- **problem_with_spark_plugs,**
- **problem_with_battery,**
- **problem_with_starter.**

The system will ask the user:

Is it true that there's petrol in the fuel tank?

Let's say that the answer is yes. This answer would be recorded, so that the user doesn't get asked the same question again. Anyway, the system now has proved that the engine is getting petrol, so now wants to find out if the engine turns over. As the system doesn't yet know whether this is the case, and as there are no rules which conclude this, the user will be asked:

Is it true that the engine turns over?

Lets say this time the answer is no. There are no other rules which can be used to prove ``problem_with_spark_plugs'' so the system will conclude that this is not the solution to the problem, and will consider the next hypothesis: problem_with_battery. It is true that the engine does not turn over (the user has just said that), so all it has to prove is that the lights don't come one. It will ask the user

Is it true that the lights come on?

Suppose the answer is no. It has now proved that the problem is with the battery. Some systems might stop there, but usually there might be more than one solution, (e.g., more than one fault with the car), or it will be uncertain which of various solutions is the right one. So usually all hypotheses are considered. It will try to prove ``problem_with_starter'', but given the existing data (the lights come on) the proof will fail, so the system will conclude that the problem is with the battery. A complete interaction with our very simple system might be:

System: Is it true that there's petrol in the fuel tank?
User: Yes.
System: Is it true that the engine turns over?
User: No.
System Is it true that the lights come on?
User: No.
System: I conclude that there is a problem with battery.

Note that in general, solving problems using backward chaining involves *searching* through all the possible ways of proving the hypothesis, systematically checking each of them.

Questions

1. ``Briefly describe the basic architecture of a typical expert system, mentioning the function of each of the main components.''

   2. ``A travel agent asks you to design an expert system to help people choose where to go on holiday. Design a set of decisions to help you give advice on which holiday to take.

Expert System Use

Expert systems are used in a variety of areas, and are still the most popular developmental approach in the artificial intelligence world. The table below depicts the percentage of expert systems being developed in particular areas:

Area	Percentage
Production/Operations Mgmt	48%
Finance	17%
Information Systems	12%
Marketing/Transactions	10%
Accounting/Auditing	5%
International Business	3%
Human Resources	2%
Others	2%

- Medical screening for cancer and brain tumours
- Matching people to jobs
- Training on oil rigs
- Diagnosing faults in car engines
- Legal advisory systems
- Mineral prospecting

Medical diagnosis

The computer does not take the place of the doctor but can be used to help the doctor make decisions.
An expert system would have information about diseases and their symptoms, the drugs used in treatments etc.

A patient is asked by a doctor about symptoms and the replies are input to the expert system. The computer searches its database, uses its rules and makes suggestions about the disease and its treatments. Sometimes probabilities are assigned to diagnoses.

Mycin was one of the earliest expert systems, and its design has strongly influenced the design of commercial expert systems and expert system shells.

Mycin was an expert system developed at Stanford in the 1970s. Its job was to diagnose and recommend treatment for certain blood infections. To do the diagnosis ``properly'' involves growing cultures of the infecting organism. Unfortunately this takes around 48 hours, and if doctors waited until this was complete their patient might be dead! So, doctors have to come up with quick guesses about likely problems from the available data, and use these guesses to provide a ``covering'' treatment where drugs are given which should deal with any possible problem.

Mycin was developed partly in order to explore how human experts make these rough (but important) guesses based on partial information. However, the problem is also a potentially important one in practical terms - there are lots of junior or non-specialised doctors who sometimes have to make such a rough diagnosis, and if there is

an expert tool available to help them then this might allow more effective treatment to be given. In fact, Mycin was never actually used in practice. This wasn't because of any weakness in its performance - in tests it outperformed members of the Stanford medical school. It was as much because of ethical and legal issues related to the use of computers in medicine - if it gives the wrong diagnosis, who do you sue?

Anyway Mycin represented its knowledge as a set of **IF-THEN rules** with certainty factors. The following is an English version of one of Mycin's rules:

> **IF** the infection is pimary-bacteremia
> **AND** the site of the culture is one of the sterile sites
> **AND** the suspected portal of entry is the gastrointestinal tract
> **THEN** there is suggestive evidence (0.7) that infection is bacteroid.

The 0.7 is roughly the certainty that the conclusion will be true given the evidence. If the evidence is uncertain the certainties of the bits of evidence will be combined with the certainty of the rule to give the certainty of the conclusion.

Mycin was written in Lisp, and its rules are formally represented as Lisp expressions. The action part of the rule could just be a conclusion about the problem being solved, or it could be an arbitary lisp expression. This allowed great flexibility, but removed some of the modularity and clarity of rule-based systems, so using the facility had to be used with care.

Anyway, Mycin is a (primarily) goal-directed system, using the basic backward chaining reasoning strategy that we described above. However, Mycin used various heuristics to control the search for a solution (or proof of some hypothesis). These were needed both to make the reasoning efficient and to prevent the user being asked too many unnecessary questions.

One strategy is to first ask the user a number of more or less preset questions that are always required and which allow the system to rule out totally unlikely diagnoses. Once these questions have been asked the system can then focus on particular, more specific possible blood disorders, and go into full backward chaining mode to try and prove each one. This rules out alot of unecessary search, and also follows the pattern of human patient-doctor interviews.

The other strategies relate to the way in which rules are invoked. The first one is simple: given a possible rule to use, Mycin first checks all the premises of the rule to see if any are known to be false. If so there's not much point using the rule. The other strategies relate more to the certainty factors. Mycin will first look at rules that have more certain conclusions, and will abandon a search once the certainties involved get below 0.2.

There are three main stages to the dialogue with Mycin .

- In the first stage, initial data about the case is gathered so the system can come up with a very broad diagnosis.
- In the second more directed questions are asked to test specific hypotheses. At the end of this section a diagnosis is proposed.
- In the third section questions are asked to determine an appropriate treatment, given the diagnosis and facts about the patient. This obviously concludes with a treatment recommendation.

At any stage the user can ask why a question was asked or how a conclusion was reached, and when treatment is recommended the user can ask for alternative treatments if the first is not viewed as satisfactory.

A new expert system called PUFF was developed using EMYCIN in the new domain of heart disorders.

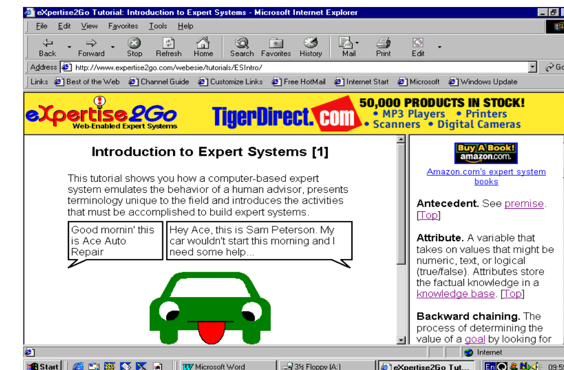
A later version called NEOMYCIN had an explicit disease classification to represent facts about different kinds of diseasesAnd system called NEOMYCIN was developed for training doctors, which would take them through various example cases, checking their conclusions and explaining where they went wrong.

Advantages.

- The computer can store far **more information** than a human. It can draw on a wide variety of sources such as stored knowledge from books case studies to help in diagnosis and advice.
- The computer does not '**forget**' or make mistakes.
- Data can be kept **up-to-date**.
- The expert system is always **available** 24 hours a day and will never 'retire'.
- The system can be used at a **distance** over a network. So rural areas or even poorer third world countries have access to experts.
- Provides accurate predictions with probabilities of all possible problems with more accurate advice.
- Some people prefer the privacy of talking to a computer.

Limitations / Disadvantages of expert systems

- **Over reliance upon computers**
- Some ‘ experts’ could loose their jobs or not be given training if computers are available to do the job.
- Lacks the 'human touch'! – lack of personal contact
- Dependent upon the correct information being given. If data or rules wrong the wrong advice could be given.
- Expert systems have no "common sense". They have no understanding of what they are for, nor of what the limits of their applicability are, nor of how their recommendations fit into a larger context. If MYCIN were told that a patient who has received a gunshot wound is bleeding to death, the program would attempt to diagnose a bacterial cause for the patient's symptoms.
- Expert systems can make absurd errors, such as prescribing an obviously incorrect dosage of a drug for a patient whose weight and age are accidentally swapped by the clerk.

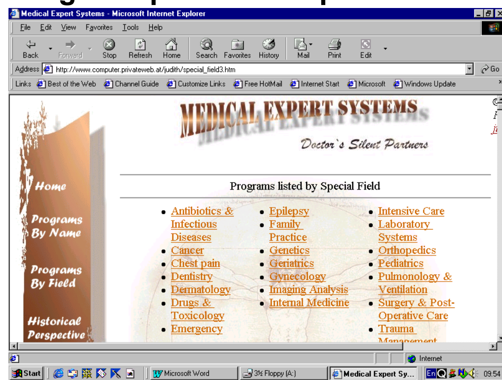


The knowledge base of an expert system is small and therefore manageable--a few thousand rules at most. Programmers are able to employ simple methods of searching and updating the KB which would not work if the KB were large. Furthermore, micro-world programming involves extensive use of what are called "domain-specific tricks"--dodges and shortcuts that work only because of the circumscribed nature of the program's "world". More general

simplifications are also possible. One example concerns the representation of time. Some expert systems get by without acknowledging time at all. In their micro-worlds everything happens in an eternal present. If reference to time is unavoidable, the micro-world programmer includes only such aspects of temporal structure as are essential to the task--for example, that if a is before b and b is before c then a is before c. This rule enables the expert system to merge suitable pairs of before-statements and so extract their implication (e.g. that the patient's rash occurred before the application of penicillin). The system may have no other information at all concerning the relationship "before"--not even that it orders events in time rather than space.

The problem of how to design a computer program that performs at human levels of competence in the full complexity of the real world remains open.

http://www.aaai.org/AITopics/html/expert.html



Delivering expertise without the expert's physical presence

The scenario we just examined used a telephone to provide remote access to an expert mechanic. Books and manuals provide other examples of packaged expertise. Methods for delivering advice without the expert's presence that include a stronger goal orientation include checklists**,** flowcharts **and** decision tables**:**

AUTO DIAGNOSTIC CHECKLIST
SECTION 1
1. Does the starter operate?
(GO TO SECTION 2)
(GO TO SECTION 3)
SECTION 2

A **checklist** for diagnosing why a car won't start might begin like this. The branching nature of the problem could result in a complex questionnaire.

START
Starter runs? No → Headlights lite? No
Starter runs? Yes → Fuel in tank? No
Headlights lite? Yes
Fuel in tank? Yes

Graphical representations of diagnostic procedures like this **flowchart**, provide an alternative to complex checklists.

Rule	**1**	**2**	**3**	**...**
Starter runs?	Y	Y	N	
Smell gas?	Y	N	.	
Dead battery	.	.	X	
Out of gas	.	X	.	
Flooded	X	.	.	

Decision tables can provide procedural guidance for complex problems. Attributes of the problem are listed in the **condition stub** (green) and recommendations or intermediate results in the **action stub** (yellow). **Rules** (read vertically) specify the action to take for any combination of conditions.

Representing knowledge in rule-based systems

RULE 1: **If** the result of switching on the headlights is nothing happens or the result of trying the starter is nothing happens **Then** the recommended action is recharge or replace the battery	RULE 2: **If** the result of trying the starter is the car cranks normally and a gas smell is not present when trying the starter **Then** the gas tank is empty with 90% confidence
RULE 3: **If** the gas tank is empty **Then** the recommended action is refuel the car	RULE 4: **If** the result of trying the starter is the car cranks normally and a gas smell is present when trying the starter **Then** the recommended action is wait 10 minutes, then restart flooded car

Each rule consists of an **if** part called the **premise** or **antecedent** (shown in blue) and a **then** part called the **consequent** or **conclusion** (shown in green). When the **if** part is true, the rule is said to **fire** and the **then** part is **asserted** -- it is considered to be a fact. Rule results are often combined to reach a conclusion. The goal of the auto diagnosis is to find a recommended action: what to do to get the car started. Rule 3 tells what to do if the gas tank is empty and rule 2 could prove that the gas tank is empty. If rule 2 fires, rule 3 will also fire and provide a recommended course of action.

The consequent in rule 2 is asserted with 90% confidence. This means that if the rule's premise is true, we are only 90% certain that the car is out of gas. Our computer-based expert might be willing to accept this level of confidence to fire rule 3 and recommend an action.