On the use of computerised decision aids

Dijkstra, Jacob Jan

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version
Publisher's PDF, also known as Version of record

Publication date:
2006

Link to publication in University of Groningen/UMCG research database

Citation for published version (APA):

Copyright
Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

Take-down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from the University of Groningen/UMCG research database (Pure): http://www.rug.nl/research/portal. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.

Download date: 11-09-2020
Chapter 2

Expert systems

2.1 Introduction

The history of expert systems is rather short. The first expert systems were developed just 25 years ago and their basic design has not changed since. An expert system is a computer program that contains a portion of human problem solving to reason out a problem like an expert does. This chapter outlines the functioning of an expert system. It describes the main parts of the expert system: the knowledge base, the inference engine, and the user interface. To clarify the difficulties of interface design the chapter concludes with some theory about Human-Computer Interaction.

First, however, this section gives an overview of the history of expert system research. In the period 1957-1959, Newell and Simon developed the predecessor of the expert system: the General Problem Solver. Newell and Simon (1972) wanted to build a computer program that could solve problems in a way similar to humans. They tried to find general problem solving rules and implemented them in a computer program. This program, the General Problem Solver, was expected to solve all problems that humans can solve. In 1957 Newell and Simon finished their first version of the General Problem Solver and showed that human problem solving can be implemented in a computer program. However, the General Problem Solver could only resolve a small set of logical problems (Parsaye & Chignell, 1988). It was far from being a `general' problem solver.

Later, it proved to be impossible to make the General Problem Solver work out more difficult problems (Waldrop, 1987). The whole of human problem solving is too complex and diverse to be captured in a small set of computer rules. Although the General Problem Solver project has been an important research project on computer reasoning, the computer program
itself was of no practical use.

About 10 years later, Feigenbaum, Buchanan, and Lederberg came up with an answer to the limitations of the General Problem Solver (Waldrop 1987, p. 37). To construct a useful problem solver, they restricted its knowledge domain. They had noticed that human experts use specific domain knowledge to solve problems. Therefore, a computerised problem solver also needs this specific domain knowledge to solve these problems (Buchanan & Shortliffe, 1985, p. 8; Waldrop, 1987, pp. 36-37). For example, doctors use their knowledge of bacterial infections when they examine a patient who might have a bacterial infection. To diagnose bacterial infections, the computer needs this specific knowledge as well. Therefore, these researchers made `specific problem solvers' that used specific domain knowledge. These computer programs became known as expert systems. Thus, an expert system is a computer program that can solve a restricted set of problems using specific knowledge about these problems.

The first expert systems were developed during the early 1970s. Successful early expert systems were DENDRAL and MYCIN (Buchanan & Shortliffe, 1985). DENDRAL, developed by Feigenbaum and Buchanan, could analyse mass-spectrogram data of chemical structures. MYCIN, constructed by Shortliffe, could diagnose bacterial infections. The success of DENDRAL and MYCIN caused a rapid growth in expert system development. People started to build expert systems for various knowledge domains. Currently, there are expert systems for all kinds of problems, for example the configuration of computers, mineral exploration, mathematical tasks, job scheduling problems, internal medicine, hydrostatic cookers, telephone switching, etc. (Durkin, 1994; Feigenbaum, McCorduck & Nii, 1988; McGraw & Harbison-Briggs, 1989).

The expert system is one of the first products to commercialise artificial intelligence. However, there are different opinions about the commercial success of expert systems. Gill (1995) claims that only one third of the early expert systems are still more or less functioning as required and the remainder has fallen into disuse. On the other hand, Hayes-Roth and Jacobstein (1994) state that expert systems have attained a permanent and secure role in industry and will increasingly be implemented. They claim that
the actual results to date contradict the impression that the technology somehow failed.

Whatever the commercial success of expert systems, the authors mentioned above agree that expert systems are actually used. Expert system development is no longer restricted to academic research projects. People use expert systems to make decisions in various lines of work (Feigenbaum et al., 1988).

2.2 The knowledge base and the inference engine

The knowledge base and the inference engine are the most important parts of an expert system: they are the 'brain' of the expert system. The knowledge base contains the rules that the inference engine uses to make inferences. These knowledge rules are known as heuristics. A heuristic is a rule of thumb, used to make problem solving easier (Parsaye & Chignell, 1988, p. 13). People use heuristics to solve problems. An example of a heuristic is: if the earth of a pot plant is dry and the plant was last watered a long time ago then the plant needs water. This is not always true, some pot plants hardly need water, but often this heuristic is very useful to grow pot plants.

To be useful to an expert system, heuristics need to be translated into an artificial language since the inference engine cannot reason with natural language. Often, expert systems use production rules as an artificial language to encode their heuristics. The general form of a production rule is: IF premise THEN conclusion. By combining rules, and using the conclusions of rules in the premises of other rules, the expert system makes inferences. The expert system uses its inference engine to make these inferences (see Figure 2.1).

Figure 2.1 shows an example of the functioning of an expert system for growing pot plants. The knowledge base contains a production rule (rule 1) that, roughly translated, states: if the earth is dry and the plant was last watered more than 2 days ago then the plant should be watered. The inference engine uses this rule to give advice. When a user consults the
Figure 2.1 The components of an expert system.
expert system to get advice on how to nurse a specific plant, the expert system takes the following steps. First, it asks the user whether the earth of the plant is dry. Then, it consults a database to find out whether this plant was last watered more than 2 days ago. Because the earth is dry and the water supply was 3 days ago, the premise of rule 1 is true. Consequently, the conclusion of rule 1 becomes also true. Finally, the expert system gives this conclusion as an advice to the user: water the plant.

Figure 2.1 presents a very simple example of an expert system. Real expert systems are far more complex. Often, a knowledge base contains many rules to cover a knowledge domain. For instance, the knowledge base of MYCIN contained about 500 production rules concerning bacterial infections. Figure 2.2 gives an example of a production rule from MYCIN (shown in LISP and its English translation).

If the expert system cannot evaluate the applicability of a rule by using other rules, then it will ask the user to provide additional information. In the example of Figure 2.1, the user has to tell whether the earth of the pot plant is dry. If MYCIN has no rules to deduct the morphology of the organism (needed in rule009, see Figure 2.2) then it will ask the user to give the morphology.

<table>
<thead>
<tr>
<th>RULE009</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREMISE: (SAND (SAME CNTXT GRAM GRAMNEG) (SAME CNTXT MORPH COCCUS))</td>
</tr>
<tr>
<td>ACTION: (CONCLUDE CNTXT IDENTITY NEISSERIA TALLY 800)</td>
</tr>
</tbody>
</table>

Translation of rule009
IF: 1) The gram stain of the organism is gramneg, and
2) The morphology of the organism is coccus
THEN: There is strongly suggestive evidence (.8) that the identity of the organism is Neisseria

Figure 2.2 A production rule from MYCIN and its translation in natural language (from Scott, Clancey, Davis & Shortliffe, 1985, p. 344).
To reduce the number of questions that users have to answer, an expert system often uses information from a database. The database of a hospital, for instance, may contain information about patients and a medical expert system can use this information when the expert system and the database are linked. In the example presented in Figure 2.1 the water supply is checked through a database. However, the user always has to give some information to describe the problem that he or she wants advice on. A medical expert system, for example, needs the most recent symptoms of a patient to make an inference about the patient's disease. In the example presented in Figure 2.1 the user had to check whether the earth of the plant was dry at the moment of consultation.

Besides production rules there are other ways to represent knowledge, for example semantic networks and frames. A thorough description of these techniques goes beyond the scope of this thesis. Parsaye and Chignell (1988) give an overview of several of these knowledge representation techniques. The next section focuses on the construction of an expert system.

2.3 Knowledge engineering and knowledge acquisition

Knowledge engineering is the overall process of developing an expert system (McGraw & Harbison-Briggs, 1989, p. 5). Usually, the knowledge engineer chooses the knowledge representation technique, puts knowledge into the knowledge base, helps designing the user interface, and tests the expert system. However, the most important and difficult part of knowledge engineering is to obtain the knowledge needed to build the expert system. This extraction and formulation of knowledge is called knowledge acquisition (Turban, 1992).

During the knowledge acquisition process the knowledge engineer interviews experts, studies handbooks, and examines other relevant knowledge sources. To build an expert system for criminal law, for instance, the knowledge engineer interviews lawyers and studies handbooks about criminal law, the law itself, and criminal law cases. After completing the acquisition the knowledge engineer translates the knowledge obtained into a
format that can be used by the expert system. This is called the
implementation of the knowledge. With this formalised knowledge the expert
system can make inferences about criminal law cases.

The knowledge engineer can also obtain knowledge by observing
experts when they solve problems. In such a case the expert may have to
think aloud and his or her behaviour is recorded on video. Afterwards, the
knowledge engineer examines how the expert solved the problems. Other
techniques to collect knowledge from a human expert are: task analysis,
protocol analysis, repertory grid, card sort, multidimensional scaling,
hierarchical cluster analysis, and network scaling. See Cooke (1994) and
McGraw and Harbison-Briggs (1989) for an overview of these techniques.

Expert systems rely heavily on knowledge and the quality of the
knowledge base depends on the level of success achieved in the knowledge
acquisition process. This makes knowledge acquisition a very important part
of knowledge engineering.

Knowledge acquisition is a difficult task (Turban, 1992). The
knowledge engineer has to specify all the knowledge needed to make the
expert system function correctly (Parsaye & Chignell, 1988, p. 300). Even
for small knowledge domains this problem is seldom solved completely.
Therefore, knowledge acquisition is the bottleneck of expert system
development (Waern, 1989).

The first problem of knowledge acquisition is to define the knowledge
domain. What knowledge does the expert system need to achieve its goals?
Only a restricted knowledge domain is suitable for the construction of an
expert system. For instance, the knowledge domain of MYCIN covered only
the diagnosis of bacterial infections that can occur in the blood. To restrict
the knowledge domain can be difficult since knowledge domains are often
linked. For example, an expert system regarding criminal attempt needs
knowledge about criminal law in general to ‘determine’ whether a criminal
attempt occurred. In other words, to judge an attempted murder case, the
expert system needs a definition of murder. If the knowledge engineer
started to define murder and all the other crimes one can attempt, he or she
would try to build an expert system for the whole of criminal law. Such a
knowledge domain is too broad.
A more fundamental problem is the problem of knowledge representation. A knowledge model is a representation of knowledge, and a representation is never the same thing as the thing being represented. A representation (or model) is a simplification of the reality. The critical thing in a representation is to represent all the important aspects and not the unimportant. When the relevant aspects are modelled, then the representation provides substantive power to enhance the ability to reason and think. When the relevant aspects are not modelled, then the representation can be misleading, causing people to ignore critical aspects of the event or perhaps form misguided conclusions (Norman, 1993, p. 49). The fact that a representation is a simplification implies that there is always a chance that the knowledge engineer has missed something that might be important one day. Therefore, one probably never gets the knowledge model of an expert system `correct'. Being a simplification of reality, a model is always incomplete (Radermacher, 1994). Or, to put it more carefully, since all models fail to correspond to the world in indefinitely many ways, they sometimes will not correspond to what is the case in a crucial and relevant way (Smith, 1995, p. 465).

Another problem is that experts can have different opinions about certain topics. For instance, lawyers sometimes disagree about what makes an attempt criminal (see Chapter 5). Sometimes it is impossible to make a consistent knowledge base when experts differ in the way they think about the knowledge domain (McGraw & Harbison-Briggs, 1989; Chapter 9). Experts can even change their minds, and when knowledge changes over time the knowledge engineer has to update the expert system.

Even more difficult is the use of common sense knowledge. Common sense is the vast amount of everyday experience and knowledge that humans have accumulated during their life. Expert systems have no `everyday experience' and therefore they have no common sense knowledge. An expert system knows nothing unless the knowledge engineer puts it into the knowledge base. For instance, when building an expert system for solving criminal law cases, the knowledge engineer needs to make a common sense definition of `attempt', if he or she wants to use the common sense meaning of the word `attempt'. This can be very difficult because common sense
knowledge is often vague. Sometimes, the knowledge engineer cannot find definitions or adequate descriptions of the common sense knowledge that he or she wants to use. Consequently, the user of the expert system has to make up for all the things the expert system does not know.

Besides these knowledge problems, knowledge acquisition itself is a difficult job. Knowledge engineers have to deal with such problems as trouble with management, uncooperative experts, experts who cannot explain their knowledge, experts who are too expensive, and time and money limits (Cleal & Heaton, 1990, Chapter 7).

To conclude, all the problems mentioned above can undermine the knowledge acquisition process and, consequently, this can corrupt the contents of the knowledge base. Therefore, the knowledge of an expert system is probably incomplete and sometimes incorrect. In addition, it is important to realise that the knowledge of an expert system, like the opinion of an expert, is subjective. Expert systems are subjective and questionable rather than objective and infallible (Boden, 1990, p. 17). As a result, an expert system ought not be treated as an oracle. It is an advisory system and the user should judge the applicability of the advice (Waldrop, 1987, p. 198).

The notion that the user should evaluate the advice makes the user interface an important part of the expert system. The user should have the possibility to judge the advice. Often, the user-interface of the expert system provides the explanation facilities needed to make this judgement. The next section considers user-interface aspects of expert systems.

2.4 The user interface and explanation facilities

Users interact with expert systems through a user interface. The interface passes data back and forth between the user and the expert system. It is the part of the expert system program that is visible to the user (McGraw, 1992). Through the interface the expert system asks the user questions, presents its advice, and provides several explanation functions (see Figure 2.1).
The expert system can also query the user. Whenever the expert system needs information to make an inference, it will ask the user to provide the necessary information. It is important that the user understands these questions and answers them properly. If a user gives the expert system the wrong facts to a problem, it would probably provide an inappropriate advice.

The explanation facilities should be able to provide the user with information that clarifies the questions the expert system asked. For this purpose almost every expert system has a `why' function, which gives the user information about questions asked by the expert system. The why-function is a help function that explains to the user why the expert system asked a certain question (Cleal & Heaton, 1990; Dhaliwal & Benbasat, 1996). It usually shows the production rule that invoked the question (see the explanation screen in Figure 2.1). This gives the user the context of the question, and the user may gain insight into the effect that his or her answer will have on the inference process. Sometimes the production rule is translated into natural language before it is shown to the user. Figure 2.3 gives an example of a screen from the why-function of MYCIN. The user has asked MYCIN why it wants to know if the patient has a fever of an unknown origin. To answer this question, MYCIN shows a translation of a production rule in which this fact (point 4) is needed to deduce whether the patient has a specific infection (infective-endocarditis).

After completing the inference process the expert system advises the user through the interface (see the advice screen in Figure 2.1). The user now has to determine whether the advice is useful. Often, an expert system has a `how' function to help the user with this judgement. A how-function is an explanation function that shows the inference process that invoked the advice (Cleal & Heaton, 1990; Dhaliwal & Benbasat, 1996). The user can evaluate the inference process by examining the rules, the facts, and the conclusions used to generate the advice. The user may also check the accuracy of the production rules. Figure 2.4 gives an example of a screen from the how-function of MYCIN. The user has asked how MYCIN decided that the organism might be pseudomas aeruginosa. To answer this question MYCIN shows the information which was used for this conclusion (the site
WHY DO YOU ASK WHETHER THE PATIENT HAS A FEVER OF UNKNOWN ORIGIN?

The rules listed below use:

whether the patient has a true fever of unknown origin

RULE180

IF:  
1) The site of the culture is blood, and
2) The number of the cultures in the series including the culture is greater than or equal to 1, and
3) The number of cultures in this series which were positive for the organism is less than or equal to 1, and
4) The patient has a true fever of unknown origin, and
5) Cardiac-surgery is a relevant item from the history of the patient

THEN: There is suggestive evidence (.6) that the infection is infective-endocarditis

**Figure 2.3** An example of the why-function from MYCIN (from Scott et al., 1985, p. 356).

of the culture, the gram stain of the organism, and the morphology of the organism). Furthermore, MYCIN gives five production rules from its knowledge base which conclude that the identity of the organism is pseudomas aeruginosa. A translation of one of these rules (rule 184) is also shown, and its conclusion gives weak evidence that the identity of the organism is pseudomas aeruginosa.

Sometimes, the expert system has a text-function as well. A text-function is an explanation function that displays the original text on which a production rule is based. This text is the original piece of knowledge which the knowledge engineer has translated into a production rule. Thus, the text can be from a handbook, an expert, or another knowledge source. The text validates the production rule and gives the user a context for the production rule. Moreover, for the user it is often easier to read the original text than to read the production rule.

In all, the interface, including the explanation facilities, is important for the interaction between the user and the expert system. For appropriate use of the expert system the interchange of information between the expert system and its user has to be adequate. Thus, interface design is an important aspect of expert system building. The next section provides some
aspects of interface design.

**Figure 2.4** An example of the how-function from MYCIN (from Scott et al., 1985, p. 357).

### 2.5 Interface design and Human-Computer Interaction research

A good interaction between the user and the expert system is critical for proper use of the expert system, and the design of the user interface regulates this interaction. Theory about interface design can be found in the Human-Computer Interaction (HCI) literature, which is strongly related with cognitive psychology research. Cognitive psychologists seek to discover fundamental principles of information processing (Booth, 1992). HCI research tries to link human information processing with computer information processing (Monk, 1984). The interface of a computer program
embodies this link.

When studying the interface of a computer, HCI researchers distinguish between the physical and the cognitive interface (Card, Moran & Newell, 1983). The physical interface refers to perceptual and motorical interaction. Sometimes researchers restrict the terms ergonomics and human factors to refer to the physical aspects of man-machine interaction (Booth, 1992). Physical aspects of the interface include display devices, audio devices, and input devices such as a mouse or a keyboard. Cognitive interface research deals with the mental activities involved during interaction with a computer. Norman (1986) developed a theory that is characteristic for this type of research.

Norman's theory offers seven stages of action as a model of human-computer interaction. These seven stages are: forming the goal, forming the intention, specifying the action, executing the action, perceiving the system state, interpreting the system state, and evaluating the outcome (see Figure 2.5). The stages describe the interaction at a low level. For instance, when a user wants to know why an expert system asked a specific question, the user has set a goal. He or she can realise this goal (intention) by specifying the corresponding action: push the `why-button' (when the expert system has a why-function). When the user pushes the expert system's why-button (execution), the explanation screen of the why-function pops up (expert system's activity). The user perceives and interprets the contents of this screen and when the information is satisfactory (evaluation), he or she closes the explanation screen (another goal, intention, action, etc.) and continues answering the questions prompted by the expert system (yet another goal, intention, etc.). Thus, Norman's theory gives a framework to describe all user activities at a low level.

Another well-known HCI-model, the goals, operators, methods, and selection rules model (GOMS; Card et al., 1983) reflects a similar approach (Norman, 1986, p. 38). The GOMS model works by breaking the task down into a goal stack, and specifying the operators, methods, and rules for selecting between alternative methods to reach the goals. The model can be used to predict times and users' routes through tasks. However, to be accurate, GOMS assumes expert behaviour and substantially error-free
performance (Booth, 1992, p. 85).

Figure 2.5 The seven stages of user activities involved in the performance of a task (Norman, 1986, p. 42).

Cognitive interface research also deals with mental models. To interact with a computer program the user must have an idea about the functionality of the program; this is the user's mental model of the program. For instance, a user might see a word processor as a kind of typewriter. Such a person will have difficulties to manage word processing functions that are atypical
for a typewriter (for instance, moving a block of text).

Besides the user's mental model, the designer of the interface also models the functionality of the computer program. The interface design reflects this model. Often, the interface designer is a programmer and therefore the interface probably reflects the functions and procedures in the source code of the computer program.

For a good interaction between the user and the computer program it is important that the user's mental model matches the designer's interface model. Either the interface should reflect the user's mental model or the user should learn the designer's model. If there is a mismatch between the user's mental model and the interface design, then the user will have difficulties specifying allowable actions that reflect his or her intentions. Furthermore, the user will have difficulties to interpret the system's state. Norman (1986) calls the first mismatches the gulf of execution and the latter the gulf of evaluation.

The designer of the expert system can try to prevent a mismatch between the interface model and the user's mental model. First, he or she can attempt to implement the user's mental model into the interface. The interface model will then reflect the user's mental model. Secondly, the designer (or the provider) of the expert system can try to teach users the interface model. The mental model of the user will then reflect the interface model.

To summarise, the main concern of HCI research is how the user can easily and effectively operate the computer program. HCI research concentrates on the user friendliness of a computer program. Most HCI research concerns interface styles such as menu interaction, command language and object manipulation with graphical interfaces (see for instance Shneiderman, 1992). However, lately new voices are entering the HCI discussion, urging a strong social and contextual orientation in what has been largely a cognitive psychology project (Carroll, 1997). The research presented in this thesis can be seen as a part of this development within HCI research.
2.6 Conclusions

An expert system is a computer program that can help its users to solve problems but it is not an oracle. Because the expert system is not infallible the user should evaluate the applicability of the expert system's advice. Therefore, the expert system has explanation facilities that can provide the user with information about the advice. Furthermore, to make this information easily accessible the interface of the expert system should be carefully designed. However, useful explanation functions with a well-designed interface do not guarantee that the user actually will consult them. The next chapter discusses the question whether or not the user wants to study the advice of an expert system.